Comparing Calculi of Explicit Substitutions with Eta-reduction

Mauricio Ayala-Rincón⋆[1], Flavio L. C. de Moura⋆⋆[1], and Fairouz Kamareddine[2]

[1] Departamento de Matemática, Universidade de Brasília, Brasília D.F., Brasil ayala@mat.unb.br, flavio@mat.unb.br
[2] Computing and Electrical Engineering, Heriot-Watt University, Edinburgh, Scotland fairouz@cee.hw.ac.uk

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 λs_e which have proved useful for solving practical problems like higher order unification, and the suspension calculus related to the implementation of the language λProlog. 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 λs_e as well as $\lambda\sigma$ and the suspension calculus are non comparable while λs_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 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:

1. The $\lambda\sigma$-style [1] which introduces two different sets of entities: one for terms and one for substitutions.
2. The suspension calculus [28, 26], denoted λ_{SUSP}, which introduces three different sets of entities: one for terms, one for environments and one for lists of environments.
3. The λs-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 λ-calculus (cf. [18]).

The desired properties of explicit substitution calculi are a) simulation of β-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 λ-calculus. $\lambda\sigma$ satisfies a), b) and d), λs satisfies a)..e) but not c). λs has an extension λs_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 β-reduction of [22].
- Extending the suspension calculus with eta-reduction resulting in λ_{SUSP}. Eta-reduction for $\lambda\sigma$ was used in [13] to deal with HOU and was introduced in [2] for the same purpose in λs_e.

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

2 Preliminaries

We assume familiarity with λ-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.

⋆ Corresponding author suported by CNPq brazilian research council
⋆⋆ Author supported by CAPES brazilian foundation

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 $a \rightarrow^* b$ we say that there exists a **derivation** from a to b. By $a \rightarrow^n b$, 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, $(A, \rightarrow_R)$, 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, θ, from its domain $\mathcal{X}$ to the domain $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is called the **grafting** of θ. As usual, valuations and their corresponding graftings are denoted by the same Greek letter. The application of a valuation θ 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 θ, 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 θ, 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 1. *Terms* $\Lambda(\mathcal{V})$*, of the* **λ-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* λ*-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 **β-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, β-reduction would be defined by $(\lambda_x.a\ \ b) \rightarrow a\{x \mapsto b\}$.

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

1. 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.
2. 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, β-reduction, should be defined in a way that takes care of renaming bound variables when necessary to avoid harmful capture of variables.

The λ-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 β-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 λ-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].

α-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. **β-reduction** *is the rewriting relation defined by the rewrite rule* (β) *and* **η-reduction** *is the rewriting relation defined by the rewrite rule* (η)*, where:*

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

$\mathcal{F}var(a)$ 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 3. *Terms* $\Lambda(\mathcal{V},\mathcal{X})$*, of the* **λ-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 λ-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 λ-calculus is an effective way of clarifying the meaning of λ-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, λ-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 λ-terms with $\ldots\lambda_z.\lambda_y.\lambda_x.\_$.

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

Definition 4. *The set* $\Lambda_{dB}(\mathcal{X})$ *of* **λ-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 Λ to denote $\Lambda_{dB}(\mathcal{X})$.

Defining β-reduction in de Bruijn notation's as $(\lambda a\ \ b) \to \{\underline{\mathtt{1}}/b\}a$ (where $\{\underline{\mathtt{1}}/b\}a$ is the substitution of the index 1 in a with b) fails: 1) when eliminating the leading abstractor all indices associated with free variable occurrences in a should be decremented; 2) when propagating the substitution $\{\underline{\mathtt{1}}/b\}$ crossing abstractors through a 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.

Definition 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)* $\underline{\mathtt{n}}^{+i} = \begin{cases} \underline{\mathtt{n}} + \underline{\mathtt{1}}, & \text{if } n > i \\ \underline{\mathtt{n}}, & \text{if } n \le i \end{cases}$ *for* $n \in \mathbb{N}$*.*

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

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

Definition 7. **β-reduction** *in the λ-calculus with de Bruijn indices is defined as* $(\lambda a\ \ b) \to \{\underline{\mathtt{1}}/b\}a$*.*

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

In the λ-calculus with names, the η-reduction rule is defined as $\lambda_x.(a\ \ x) \to a$, 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 8. **η-reduction** *in the λ-calculus with de Bruijn indices is:* $\lambda(a\ \ 1) \to b$ *if* $\exists b\ b^{+} = a$*.*

3 Calculi à la $\lambda\sigma$, λs_e and λ_{susp}

3.1 The $\lambda\sigma$-calculus

The $\lambda\sigma$-calculus works on 2-sorted terms: *(proper) terms*, and *substitutions* (over which $s, t, \ldots$ range).

Definition 9. *The $\lambda\sigma$-calculus is defined as the calculus of the rewriting system $\lambda\sigma$ of Table 1 where*

TERMS $a \ ::= \ \underline{1} \mid X \mid (a \ \ a) \mid \lambda a \mid a[s]$, *where* $X \in \mathcal{X}$ *and* SUBS $s \ ::= \ id \mid \ \uparrow \ \mid a.s \mid s \circ s$

Table 1. The $\lambda\sigma$ Rewriting System of the $\lambda\sigma$-calculus with Eta-rule

(Beta)	$(\lambda a \ \ b) \longrightarrow a\,[b \cdot id]$	*(Id)*	$a[id] \longrightarrow a$
(VarCons)	$\underline{1}\,[a \cdot s] \longrightarrow a$	*(App)*	$(a \ \ b)[s] \longrightarrow (a\,[s])\,(b\,[s])$
(Abs)	$(\lambda a)[s] \longrightarrow \lambda a\,[\underline{1} \cdot (s \circ \uparrow)]$	*(Clos)*	$(a\,[s])[t] \longrightarrow a\,[s \circ t]$
(IdL)	$id \circ s \longrightarrow s$	*(IdR)*	$s \circ id \longrightarrow s$
(ShiftCons)	$\uparrow \circ\, (a \cdot s) \longrightarrow s$	*(Map)*	$(a \cdot s) \circ t \longrightarrow a\,[t] \cdot (s \circ t)$
(Ass)	$(s \circ t) \circ u \longrightarrow s \circ (t \circ u)$	*(VarShift)*	$\underline{1} \cdot \uparrow \longrightarrow id$
(SCons)	$\underline{1}[s] \cdot (\uparrow \circ\, s) \longrightarrow s$	*(Eta)*	$\lambda(a \ \ \underline{1}) \longrightarrow b$ if $a =_\sigma b[\uparrow]$

For every substitution s we define the *iteration of the composition of* s inductively as $s^1 = s$ and $s^{n+1} = s \circ s^n$. We use s^0 to denote id. Note that the only de Bruijn index used is $\underline{1}$, but we can code $\underline{n}$ by $\underline{1}[\uparrow^{n-1}]$.

The equational theory associated with the rewriting system $\lambda\sigma$ defines a congruence denoted $=_{\lambda\sigma}$. The congruence obtained by dropping *Beta* and *Eta* is denoted $=_\sigma$. When we restrict reduction to these rules, we will use expressions such as σ-reduction, σ-normal form, etc, with the obvious meaning.

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] as: 1) λa, where a is a normal term; 2) $a_1 \ldots a_p . \uparrow^n$, where $a_1, \ldots, a_p$ are normal terms and $a_p \neq \underline{n}$ or 3) $(a \ b_1 \ldots b_n)$, where a is either $\underline{1}$, $\underline{1}[\uparrow^n]$, X or $X[s]$ for s a substitution term different from id in normal form.

In the λ-calculus with names or de Bruijn indices, the rule $X\{y/a\} = X$, where y is an element of $\mathcal{V}$ or a de Bruijn index, respectively, is necessary because there is no way to suspend the substitution $\{y/a\}$ until X is instantiated. In the $\lambda\sigma$-calculus, the application of this substitution can be delayed, since the term $X[s]$ does not reduce to X. The fact that the application of a substitution to a meta-variable can be suspended until the meta-variable is instantiated will be used to code the substitution of variables in $\mathcal{X}$ by "$\mathcal{X}$-grafting" and explicit lifting. Consequently a notion of $\mathcal{X}$-substitution in the $\lambda\sigma$-calculus is unnecessary. Observe that the condition $a =_\sigma b[\uparrow]$ of the *Eta* rule is stronger than the condition $a = b^+$ given in Definition 8 as $X = X^+$, but there exists no term b such that $X =_\sigma b[\uparrow]$. Note that $\lambda\sigma$-reduction is compatible with first order substitution or grafting and hence $\mathcal{X}$-grafting and $\lambda\sigma$-reduction commute.

3.2 The λs_e-calculus

The λs_e-calculus of [21] is an extension of the λs-calculus ([19]) which is CR on open terms and insists on remaining close to the syntax of the λ-calculus. Next to abstraction and application, substitution (σ) and updating (φ) operators are introduced. A term containing neither σ nor φ is called a *pure lambda term.*

Definition 10. *Terms of the* λs_e**-calculus**, *whose set of rules is presented in Table 2, are given by:*

$\Lambda s_e \ ::= \ X \mid \mathbb{N} \mid \Lambda s_e \Lambda s_e \mid \lambda \Lambda s_e \mid \Lambda s_e\, \sigma^j \Lambda s_e \mid \varphi^i_k \Lambda s_e$, *where* $j, i \geq 1$, $k \geq 0$, $X \in \mathcal{X}$

The equational theory associated to the rewriting system λs_e defines a congruence $=_{\lambda s_e}$. The congruence obtained by dropping *σ-generation* and *Eta* is denoted by $=_{s_e}$. The λs-calculus is the one associated with the first eight rules of the λs_e and without the meta variables X standing for open terms in the set of terms.

We can describe the operators of the λs_e-calculus over the signature of a first order sorted term algebra $\mathcal{T}_{\lambda s_e}(\mathcal{X})$ built on $\mathcal{X}$, the set of variables of sort TERM and its subsort NAT$\subset$TERM:

Table 2. The Rewriting System of the λs_e-calculus with Eta-rule

(σ-generation)	$(\lambda a\ \ b) \longrightarrow a\,\sigma^1\, b$	
(σ-λ-transition)	$(\lambda a)\,\sigma^i b \longrightarrow \lambda(a\,\sigma^{i+1}\, b)$	
(σ-app-transition)	$(a_1\ \ a_2)\,\sigma^i b \longrightarrow ((a_1\,\sigma^i b)\ \ (a_2\,\sigma^i b))$	
(σ-destruction)	$\underline{\mathtt{n}}\,\sigma^i b \longrightarrow \begin{cases} \underline{\mathtt{n-1}} & \text{if } n > i \\ \varphi^i_0\, b & \text{if } n = i \\ \underline{\mathtt{n}} & \text{if } n < i \end{cases}$	
(φ-λ-transition)	$\varphi^i_k(\lambda a) \longrightarrow \lambda(\varphi^i_{k+1}\, a)$	
(φ-app-transition)	$\varphi^i_k(a_1\ \ a_2) \longrightarrow ((\varphi^i_k\, a_1)\ \ (\varphi^i_k\, a_2))$	
(φ-destruction)	$\varphi^i_k\, \underline{\mathtt{n}} \longrightarrow \begin{cases} \underline{\mathtt{n+i-1}} & \text{if } n > k \\ \underline{\mathtt{n}} & \text{if } n \le k \end{cases}$	
(Eta)	$\lambda(a\ \ \underline{1}) \longrightarrow b$	if $a =_{s_e} \varphi^2_0 b$
(σ-σ-transition)	$(a\,\sigma^i b)\,\sigma^j\, c \longrightarrow (a\,\sigma^{j+1}\, c)\ \sigma^i\, (b\,\sigma^{j-i+1}\, c)$	if $i \le j$
(σ-φ-transition 1)	$(\varphi^i_k\, a)\,\sigma^j\, b \longrightarrow \varphi^{i-1}_k\, a$	if $k < j < k+i$
(σ-φ-transition 2)	$(\varphi^i_k\, a)\,\sigma^j\, b \longrightarrow \varphi^i_k(a\,\sigma^{j-i+1}\, b)$	if $k+i \le j$
(φ-σ-transition)	$\varphi^i_k(a\,\sigma^j\, b) \longrightarrow (\varphi^i_{k+1}\, a)\,\sigma^j\, (\varphi^i_{k+1-j}\, b)$	if $j \le k+1$
(φ-φ-transition 1)	$\varphi^i_k\,(\varphi^j_l\, a) \longrightarrow \varphi^j_l\,(\varphi^i_{k+1-j}\, a)$	if $l+j \le k$
(φ-φ-transition 2)	$\varphi^i_k\,(\varphi^j_l\, a) \longrightarrow \varphi^{j+i-1}_l\, a$	if $l \le k < l+j$

$$\begin{array}{llrcll} \underline{\mathtt{n}} & : & & \to & \text{NAT}, & \forall n \in \mathbb{N}\setminus\{0\} \\ \_\sigma^i\_ & : & \text{TERM}\times\text{TERM} & \to & \text{TERM}, & \forall i \in \mathbb{N}\setminus\{0\} \\ \varphi^i_k\_ & : & \text{TERM} & \to & \text{TERM}, & \forall i \in \mathbb{N}, k \in \mathbb{N}\setminus\{0\} \end{array} \qquad \begin{array}{llrcl} (\_\ \_) & : & \text{TERM}\times\text{TERM} & \to & \text{TERM} \\ \lambda\_ & : & \text{TERM} & \to & \text{TERM} \end{array}$$

The λs_e-calculus has been proved in [21] to be CR on open terms; to simulate β-reduction: let $a,\, b \in \Lambda$, if $a \to_\beta b$ then $a \to^*_{\lambda s_e} b$; to be sound: let $a,\, b \in \Lambda$, if $a \to^*_{\lambda s_e} b$ then $a \to^*_\beta b$; and its associated substitution calculus, that is the s_e-calculus, to be WN and CR. The characterization of the λs_e-normal forms was given in [21, 2] as: a term $a \in \Lambda s_e$ is a λs_e-nf if and only if one of the following holds:
1. $a \in \mathcal{X} \cup \mathbb{N}$;
2. $a = (b\ \ c)$, where b, c are λs_e-nf and b is not an abstraction λd;
3. $a = \lambda b$, where b is a λs_e-nf excluding applications of the form $(c\ \ \underline{1})$ where $\varphi^2_0 d =_{s_e} c$ for some d;
4. $a = b\sigma^j c$, where b, c are λs_e-nf and b is of the form: (a) X or (b) $d\sigma^i e$, with $j < i$ or (c) $\varphi^i_k d$, with $j \le k$
5. $a = \varphi^i_k b$, where b is a λs_e-nf of the form: (a) X or (b) $c\sigma^j d$, with $j > k+1$ or (c) $\varphi^j_l c$, with $k < l$;

3.3 The suspension calculus

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

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

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 β_s, that initiates the simulation of a β-reduction (as for the $\lambda\sigma$ and the λs_e, respectively, the *Beta* and the *σ-generation* rules do) and two sets of rules used for handling the suspended terms. The first set, the r rules, for reading suspensions and the second one, the m rules, for merging suspensions. These rules are given in the Table 3.

As in [28] we denote by $\rhd_{rm}$ the reduction relation defined by the r- and m-rules in the Table 3. The associated substitution calculus, denoted as SUSP, is the one given by the congruence $=_{rm}$.

Table 3. Rewriting rules of the suspension calculus

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

Definition 11 ([28]). *The* length $len(e)$ *of an environment* e *is given by:*

$$len(nil) := 0 \qquad len(et :: e') := len(e') + 1 \qquad len(\{\!\{ e_1, i, j, e_2 \}\!\}) := len(e_1) + (len(e_2) \mathbin{\dot{-}} i).$$

The index $ind(et)$ *of an environment term* et*, and the* l-th index $ind_l(e)$ *of the environment* e *and* $l \in \mathbb{N}$*, are simultaneously defined by induction on the structure of expressions:*

$$ind(@m) = m{+}1 \qquad ind((t', m)) = m \qquad ind(\langle\!\langle et', j, k, e \rangle\!\rangle) = \begin{cases} ind_m(e) + (j \mathbin{\dot{-}} k) & \text{if } len(e) > j \mathbin{\dot{-}} ind(et') = m \\ ind(et') & \textit{otherwise} \end{cases}$$

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

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

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

Definition 12 ([28]). *An expression of the suspension calculus is said to be* well-formed *if the following conditions hold over all its subexpressions* s*:*

- *if* s *is* $\llbracket t, ol, nl, e \rrbracket$ *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 deal only with well-formed expressions of the suspension calculus.

The suspension calculus has been proved to simulate β-reduction and its associated substitution calculus SUSP to be 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 1 ([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* λ*-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 η-reduction: the λ_{susp}-calculus

The suspension calculus was initially formulated without η-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 λ_{SUSP} and its associated substitution calculus remains as SUSP. The *Eta* rule is formulated as follows:

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

Intuitively *Eta* may be interpreted as: when it is possible to apply the η-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 1 proves the correctness of *Eta* according to this interpretation. We follow [10] and [3] for $\lambda\sigma$ and λs_e respectively, and implement the *Eta* rule of the λ_{SUSP}-calculus by introducing a dummy symbol $\Diamond$, as

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

The correctness of this implementation is explained because an η-reduction $\lambda(M\ \underline{1}) \to_\eta N$ gives us a term N, that is obtained from M by decrementing by one all free occurrences of de Bruijn indices, as previously mentioned, and that 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 2. *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.

- $\boldsymbol{A = \underline{n}}$. If $n > k$ then $[\![\underline{n}, k, k+1, @k :: \ldots :: @1 :: nil]\!] \to_{r_5}^{k} [\![\underline{n-k}, 0, k+1, nil]\!] \to_{r_2} \underline{n+1}$. If $n \leq k$ then $[\![\underline{n}, k, k+1, @k :: \ldots :: @1 :: nil]\!] \to_{r_5}^{n-1} [\![\underline{1}, k-n+1, k+1, @k-n+1 :: \ldots :: @1 :: nil]\!] \to_{r_3} \underline{n}$;
- $\boldsymbol{A = (B\ C)}$. 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 $[\![(\lambda B), k, k+1, @k :: \ldots :: @1 :: nil]\!] \to_{r_7} \lambda[\![B, k+1, k+2, @k+1 :: \ldots :: @1 :: nil]\!]$, 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 $\rhd_{rm}$-normalized and by the Lemma 1 the obtained term is of one of the forms analysed in the previous cases. □

Proposition 1 (Soundness of the *Eta* rule). *Every application of the Eta rule of λ_{SUSP} to the redex $\lambda(t_1\ \underline{1})$ 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.

- $\boldsymbol{t_2 = \underline{n}}$. $[\![\underline{n}, 0, 1, nil]\!] \to_{r_2} \underline{n+1} =_{rm} t_1$.
- $\boldsymbol{t_2 = (A\ B)}$. Without loss of generality we can assume that both A and B are $\rhd_{rm}$-normalized. Observe that $[\![(A\ B), 0, 1, nil]\!] \to_{r_6} [\![A, 0, 1, nil]\!]\ [\![B, 0, 1, nil]\!]$. 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 $\rhd_{rm}$-nf. Note that $[\![(\lambda A), 0, 1, nil]\!] \to_{r_7} (\lambda[\![A, 1, 2, @1 :: nil]\!])$. By applying the Lemma 2 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 $\rhd_{rm}$-normal form then $[\![t, i, j, e]\!] \rhd_{rm}^{*} t'$, where t' is a pure λ-term in de Bruijn notation by Lemma 1. 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 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]. See Appendix A. □

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

Lemma 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 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 3 (minus (β_s)) and *Eta*: namely, the one between *Eta* and (r_7).

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

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.

- $\boldsymbol{t_1 = \underline{n}}$. 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, $\lambda([\![\underline{n}, ol+1, nl+1, @nl :: e]\!]\ \underline{1}) \rightarrow^{ol+1}_{r_5} \lambda([\![\underline{\texttt{n-ol-1}}, 0, nl+1, nil]\!]\ \underline{1}) \rightarrow_{r_2} \lambda(\underline{\texttt{n-ol+nl}}\ \underline{1}) \rightarrow_{Eta} \underline{\texttt{n-ol+nl-1}}$. On the other side, $t_1 =_{rm} [\![t_2, 0, 1, nil]\!]$, hence $t_2 = \underline{\texttt{n-1}}$ and we have $[\![\underline{\texttt{n-1}}, ol, nl, e]\!] \rightarrow^{ol}_{r_5} [\![\underline{\texttt{n-1-ol}}, 0, nl, nil]\!] \rightarrow_{r_2} \underline{\texttt{n-ol+nl-1}}$.
  - $ol + 1 \geq n$. On the one side, $\lambda([\![\underline{n}, ol+1, nl+1, @nl :: e]\!]\ \underline{1}) \rightarrow^{n-1}_{r_5} \lambda([\![\underline{1}, ol-n+2, nl+1, e_1 :: e']\!]\ \underline{1})$ and the subsequent derivation depends on the structure of e_1:
    when $e_1 = @l$ we apply r_3 obtaining $\lambda(\underline{\texttt{nl+1-l}}\ \underline{1}) \rightarrow_{Eta} \underline{\texttt{nl-l}}$ and on the other side, $[\![\underline{\texttt{n-1}}, ol, nl, e]\!] \rightarrow^{n-2}_{r_5} [\![\underline{1}, ol-n+2, nl, @l :: e']\!] \rightarrow_{r_3} \underline{\texttt{nl-l}}$;
    when $e_1 = (t, l)$, where without loss of generality t is suppossed to be in $\rhd_{rm}$-nf, we have $\lambda([\![\underline{1}, ol-n+2, nl+1, (t,l) :: e']\!]\ \underline{1}) \rightarrow_{r_4} \lambda([\![t, 0, nl-l+1, nil]\!]\ \underline{1}) \rightarrow_{Eta}$ $\rhd_{rm}\text{-}nf([\![[\![t, 0, nl{+}1{-}l, nil]\!], 1, 0, (\lozenge, 0) :: nil]\!]) \rightarrow_{m_1} \rhd_{rm}\text{-}nf([\![t, 0, nl{-}l, \{\!\{nil, nl{+}1{-}l, 1, (\lozenge, 0) :: nil\}\!\}]\!])$ $\rightarrow_{m_3} \rhd_{rm}\text{-}nf([\![t, 0, nl-l, \{\!\{nil, nl-l, 0, nil\}\!\}]\!]) \rightarrow_{m_2} \rhd_{rm}\text{-}nf([\![t, 0, nl-l, nil]\!])$ and on the other side, $[\![\underline{1}, ol-n+2, nl, (t,l) :: e']\!] \rightarrow_{r_4} [\![t, 0, nl-l, nil]\!]$.
    Since $\rhd_{rm}\text{-}nf([\![t, 0, nl-l, nil]\!])$ and $[\![t, 0, nl-l, nil]\!]$ are joinable we obtain the confluence.
- $\boldsymbol{t_1 = (A\ B)}$. Since the sole rule of the λ_{SUSP} that truly "applies" applications is the β_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 $\rhd_{rm}$-nf$([\![[\![(\lambda A), ol+1, nl+1, @nl{::}e]\!], 1, 0, (\lozenge, 0){::}nil]\!])$ and the term $[\![\ \rhd_{rm}\text{-nf}([\![(\lambda A), 1, 0, (\lozenge, 0){::}nil]\!]), ol, nl, e]\!]$.
  On the one side, $[\![\ \rhd_{rm}\text{-nf}([\![(\lambda A), 1, 0, (\lozenge, 0) :: nil]\!]), ol, nl, e]\!] \rhd^*_{rm}$
  $\rhd_{rm}$-nf$([\![[\![(\lambda A), 1, 0, (\lozenge, 0) :: nil]\!], ol, nl, e]\!]) \rightarrow_{r_7, r_7}$
  $\rhd_{rm}$-nf$((\lambda[\![[\![A, 2, 1, @0 :: (\lozenge, 0) :: nil]\!], ol+1, nl+1, @nl :: e]\!])) \rhd^*_{rm}$
  $(\lambda \rhd_{rm}$ -nf$([\![[\![A, 2, 1, @0 :: (\lozenge, 0) :: nil]\!], ol+1, nl+1, @nl :: e]\!])) \rightarrow_{m_1}$
  $(\lambda \rhd_{rm}$ -nf$([\![A, ol+2, nl+1, \{\!\{@0 :: (\lozenge, 0) :: nil, 1, ol+1, @nl :: e\}\!\}]\!]))$
  and we have that $\{\!\{@0 :: (\lozenge, 0) :: nil, 1, ol+1, @nl :: e\}\!\} \rightarrow_{m_5, m_5}$
  $\langle\!\langle @0, 1, ol+1, @nl :: e\rangle\!\rangle :: \langle\!\langle (\lozenge, 0), 1, ol+1, @nl :: e\rangle\!\rangle :: \{\!\{nil, 1, ol+1, @nl :: e\}\!\} \rightarrow_{m_7}$
  $@nl :: \langle\!\langle (\lozenge, 0), 1, ol+1, @nl :: e\rangle\!\rangle :: \{\!\{nil, 1, ol+1, @nl :: e\}\!\} \rightarrow_{m_{10}}$
  $@nl :: \langle\!\langle (\lozenge, 0), 0, ol, e\rangle\!\rangle :: \{\!\{nil, 1, ol+1, @nl :: e\}\!\} \rightarrow_{m_3, m_4} @nl :: \langle\!\langle (\lozenge, 0), 0, ol, e\rangle\!\rangle :: e.$

Then we obtain the term $(\lambda \rhd_{rm}\text{-nf}([\![A, ol+2, nl+1, @nl :: \langle\!\langle(\Diamond,0),0,ol,e\rangle\!\rangle :: e]\!]))$.
On the other side, $\rhd_{rm}\text{-nf}([\![[\![(\lambda A), ol+1, nl+1, @nl :: e]\!], 1, 0, (\Diamond, 0) :: nil]\!]) \to_{r_7,r_7}$
$\rhd_{rm}\text{-nf}((\lambda[\![[\![A, ol+2, nl+2, @nl+1 :: @nl :: e]\!], 2, 1, @0 :: (\Diamond, 0) :: nil]\!])) \rhd^*_{rm}$
$(\lambda \rhd_{rm}\text{-nf}([\![[\![A, ol+2, nl+2, @nl+1 :: @nl :: e]\!], 2, 1, @0 :: (\Diamond, 0) :: nil]\!])) \to_{m_1}$
$(\lambda \rhd_{rm}\text{-nf}([\![A, ol+2, nl+1, \{\!\{@nl+1 :: @nl :: e, nl+2, 2, @0 :: (\Diamond,0) :: nil\}\!\}]\!]))$
and we have that $\{\!\{@nl+1 :: @nl :: e, nl+2, 2, @0 :: (\Diamond,0) :: nil\}\!\} \to_{m_5,m_5}$
$\langle\!\langle @nl+1, nl+2, 2, @0 :: (\Diamond,0) :: nil\rangle\!\rangle :: \langle\!\langle @nl, nl+2, 2, @0 :: (\Diamond,0) :: nil\rangle\!\rangle :: \{\!\{e, nl+2, 2, @0 :: (\Diamond,0) :: nil\}\!\}$
$\to_{m_7} @nl :: \langle\!\langle @nl, nl+2, 2, @0 :: (\Diamond,0) :: nil\rangle\!\rangle :: \{\!\{e, nl+2, 2, @0 :: (\Diamond,0) :: nil\}\!\}$
$\rhd^*_{rm}$(By the Lemma 4, since we are working with well-formed terms and then $ind(e) \le nl$)
$@nl :: \langle\!\langle @nl, nl+2, 2, @0 :: (\Diamond,0) :: nil\rangle\!\rangle :: e \to_{m_{10}}$
$@nl :: \langle\!\langle @nl, nl+1, 1, (\Diamond,0) :: nil\rangle\!\rangle :: e \to_{m_8} @nl :: (\Diamond, nl) :: e$.
Then we obtain the term $(\lambda \rhd_{rm}\text{-nf}([\![A, ol+2, nl+1, @nl :: (\Diamond, nl) :: e]\!]))$.
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 consequently these second environment terms should disapear. Then 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 1 (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 λ_{SUSP} as well as the $\lambda\sigma$ and the λs_e are non comparable. Additionally, we prove that the λs_e is more adequate than the λ_{SUSP}.

Let $a, b \in \Lambda$ such that $a \to_\beta b$. A *simulation* of this β-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 β (*beta* for $\lambda\sigma$, *σ-generation* for λs_e, β_s for λ_{SUSP}) applied to the same redex as the redex in $a \to_\beta b$. The criterion of adequacy is defined as follow:

Definition 13 (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 β-reduction) than the $\lambda\xi_2$-calculus, denoted $\lambda\xi_1 \prec \lambda\xi_2$, if*
- *for every β-reduction $a \to_\beta b$ and every $\lambda\xi_2$-simulation $a \to^n_{\lambda\xi_2} b$ there exists a $\lambda\xi_1$-simulation $a \to^m_{\lambda\xi_1} b$ such that $m \le n$;*
- *there exists a β-reduction $a \to_\beta b$ and a $\lambda\xi_1$-simulation $a \to^m_{\lambda\xi_1} b$ such that for every $\lambda\xi_2$-simulation $a \to^n_{\lambda\xi_2} b$ 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 λs are non comparable presented in [22] apply for the incomparability of $\lambda\sigma$ and λs_e since λs_e is an extension of λs for open terms.

Proposition 2. *The $\lambda\sigma$- and the λs_e-calculi are non comparable.*

Lemma 6. *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 7. *Every λ_{SUSP}-derivation of $(\lambda\lambda(\underline{2}\ \underline{2}))\ \underline{1}^n$ to its λ_{SUSP}-nf has length $4n+5$.*

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

Lemma 8 ([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 3. *The $\lambda\sigma$- and λ_{SUSP}-calculi are non comparable.*

Proof. On the one side, by the Lemmas 7 and 8, 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 λ_{SUSP}: $((\lambda\lambda\underline{2})\ \underline{1}) \to_{\beta_s} [\![(\lambda\underline{2}),1,0,(\underline{1},0) :: nil]\!] \to_{r_7} \lambda[\![\underline{2},2,1,@0 :: (\underline{1},0) :: nil]\!] \to_{r_5} \lambda[\![\underline{1},1,1,(\underline{1},0) :: nil]\!] \to_{r_4} \lambda[\![\underline{1},0,1,nil]\!] \to_{r_2} \lambda\underline{2}$.

This simulation jointly with the Lemma 6 allows us to conclude that $\lambda\sigma \not\preceq \lambda_{\text{SUSP}}$. □

To prove that λs_e is more adequate than λ_{SUSP} we need to estimate the lengths of derivations.

Definition 14. *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:*

$$M(\underline{\mathtt{n}})=1 \qquad M(\lambda A)=M(A)+1 \qquad M(A\ B)=M(A)+M(B)+1$$

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

Lemma 9. *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 λs-calculus in [22]. □

Lemma 10. *Let $A \in \Lambda$. Then all SUSP-derivations of $[\![A,i,i,@i-1 :: \ldots :: @0 :: nil]\!]$ 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 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 $1 + M(B) = M(\lambda B) = M(A)$. □

Lemma 11. *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 induction hypothesis we obtain the desired result.

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

Proposition 4. *Every* SUSP*-derivation of $[\![A, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!]$, where $A, B \in \Lambda$ and $k \geq 0$ 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 $[\![\underline{\mathtt{n}}, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \to_{r_5}^{n-1}$ $[\![\underline{\mathtt{1}}, k-n+1, k-1, @k-n-1 :: \ldots :: @0 :: (B, l) :: nil]\!] \to r_3\ \underline{\mathtt{n}}$. This derivation has length $n \geq Q_k(\underline{\mathtt{n}}, B)$. If $n = k$ then $[\![\underline{\mathtt{n}}, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \to_{r_5}^{n-1} [\![\underline{\mathtt{1}}, 1, k-1, (B, l) :: nil]\!] \to r_4$ $[\![B, 0, k-1-l, nil]\!]$. By the Lemma 11 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 $[\![\underline{\mathtt{n}}, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \to_{r_5}^{k} [\![\underline{\mathtt{n-k}}, 0, k-1, nil]\!] \to r_2\ \underline{\mathtt{n-1}}$. Derivation whose length is $k+1 \geq Q_k(\underline{\mathtt{n}}, B) = Q_k(A, B)$.

– $\boldsymbol{A = (C\ D)}$. $[\![(C\ D), k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \to_{r_6}$ $([\![C, k, k-1, @k-2 :: \ldots :: @0 :: (B, 0) :: nil]\!]\ [\![D, k, k-1, @k-2 :: \ldots :: @0 :: (B, 0) :: nil]\!])$. By induction hypothesis the derivation has length greater than or equal to $1 + Q_k(C, B) + Q_k(D, B) = Q_k((C\ D), B) = Q_k(A, B)$.

– $\boldsymbol{A = (\lambda C)}$. $[\![(\lambda C), k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \to_{r_7}$ $\lambda[\![C, k+1, k, @k-1 :: \ldots :: @0 :: (B, l) :: nil]\!]$. By induction hypothesis we can conclude that this derivation has length greater than or equal to $1 + Q_{k+1}(C, B) = Q_k(\lambda C, B) = Q_k(A, B)$. □

Proposition 5. *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 the Lemma 9 obtaining that the complete s_e-normalization has length $1 + M(B)$. In the two cases the derivation has length less than or equal to $Q_k(\underline{\mathtt{n}}, B)$.

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

Theorem 2 ($\lambda s_e \prec \lambda_{\mathbf{susp}}$). *The λs_e- is more adequate than the λ_{SUSP}-calculus.*

Proof. We prove the stronger result that if $A \in \Lambda$ and $A \to_{\beta_s} B \to_{\text{SUSP}}^m$ SUSP-nf(B) is a λ_{SUSP}-simulation of a β-reduction then: $A \to_{\sigma-generation} C \to_{s_e}^n s_e\text{-nf}(C)$ has length $n + 1 \leq m + 1$.
In λ_{SUSP}, for any redex of β_s we have $(\lambda D)\ E \to_{\beta_s} [\![D, 1, 0, (E, 0) :: nil]\!] \to_{\text{SUSP}}^m$ SUSP-nf$([\![D, 1, 0, (E, 0) :: nil]\!])$. In the λs_e, $(\lambda D)\ E \to_{\sigma-generation} D\sigma^1 E \to_{s_e}^n s_e\text{-nf}(D\sigma^1 E)$. By Propositions 4 and 5, $m \geq Q_1(D, E) \geq n$. Hence, the length of a λ_{SUSP}-simulation of a β-contraction is not shorter than that of some λs_e-simulation.

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

As mentioned in the proof above, we prove a stronger result than simple better adequacy of λs_e as in [22]. In fact, we prove that the length of all λs_e-simulations are shorter than the length of any λ_{SUSP}-simulation. Examining the proofs of Propositions 4 and 5 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, λ_{SUSP} deals with the substitutions in a less efficient way. To explain that, compare the simulations of β-reduction from the term $(\lambda(\lambda^n \underline{\mathtt{i}}))\ \underline{\mathtt{j}}$, where $n \geq 0$:

$$\begin{array}{llll} (\lambda(\lambda^n \underline{\mathtt{i}}))\ \underline{\mathtt{j}} \to_{\sigma-gen} (\lambda^n \underline{\mathtt{i}})\sigma^1 \underline{\mathtt{j}} & \to^n_{\sigma-\lambda-tra} & \lambda^n(\underline{\mathtt{i}}\sigma^{n+1}\underline{\mathtt{j}}) & =: t_1 \\ (\lambda(\lambda^n \underline{\mathtt{i}}))\ \underline{\mathtt{j}} \to_{\beta_s} [\![\lambda^n \underline{\mathtt{i}}, 1, 0, (\underline{\mathtt{j}}, 0) :: nil]\!] & \to^n_{r_7} & \lambda^n [\![\underline{\mathtt{i}}, n+1, n, @n-1 :: \ldots :: @0 :: (\underline{\mathtt{j}}, 0) :: nil]\!] & =: t_2 \end{array}$$

After that the λs_e complete the simulation in one or two steps by checking arithmetic inequations:

$$t_1 \to_{\sigma-dest} \begin{cases} \lambda^n \underline{\mathtt{i}}, & \text{if } i < n+1 \\ \lambda^n \underline{\mathtt{i-1}}, & \text{if } i > n+1 \\ \lambda^n (\varphi_0^{n+1} \underline{\mathtt{j}}) \to_{\varphi-dest} \lambda^n \underline{\mathtt{j+n}}, & \text{if } i = n+1 \end{cases}$$

But in the λ_{SUSP} we have to destruct the environment list, environment by environment:

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

These simple considerations lead us to believe that the main difference of the two calculus (at least in the simulation of β-reduction) is given by the manipulation of indices: although λ_{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 or not of strong normalization by λ_{SUSP} as conjectured positively in [26], since the λs_e has been proved to answer this question negatively in [15, 16].

6 Future Work and Conclusion

[13, 2] showed that η-reduction is of great interest for adaptating substitution calculi ($\lambda\sigma$ and λs_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 η-reduction and showed that this extended suspension calculus λ_{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 of β-reduction. We concluded that $\lambda\sigma$ and $\lambda\xi$ are mutually non comparable for $\xi \in \{s_e, \text{SUSP}\}$ but that λs_e is more adequate than λ_{SUSP}.

An immediate work to be done is to study two open questions: 1) whether the s_e-calculus has strong normalization (SN), 2) whether λ_{SUSP} preserves SN. Interesting points arise in this context since: a) λs_e is more adequate than λ_{SUSP}, b) λs_e does not preserves SN [16] and c) the substitution calculus of λ_{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 λs_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 λs_e-Style of Unification for Simply-Typed Higher Order Unification in the Pure lambda Calculus. In R.J.G.B. de Queiroz and M. Ayala-Rincón, editors, *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. $\lambda\upsilon$, 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. λ-Calculi with Explicit Substitutions and Composition which Preserve β-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 λs_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 λ-notation. *Theoretical Computer Science*, 155:85–109, 1996.
19. F. Kamareddine and A. Ríos. A λ-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 λ-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 λs-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 β-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 λProlog. 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 λ-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.

A Proof of Lemma 3: noetherianity of susp enlarged with the *Eta* rule

Definition 15 ([28]). *The* immediate subexpression(s) *of an expression x are given by:*

1) If x is a term of the form: (a) $(t_1\ t_2)$, then t_1 and t_2 are its immediate subexpressions; (b) (λt), then t is its immediate subexpression; (c) $[\![t, ol, nl, e]\!]$, then t and e are its immediate subexpressions;

2) if x is an environment of the form: (a) $et :: e$, then et and e are its immediate subexpressions; (b) $\{\!\{e_1, i, j, e_2\}\!\}$, then e_1 and e_2 are its immediate subexpressions;

3) *if x is an environment term of the form: (a) (t, l), then t is its immediate subexpression; (b) $\langle\langle et, i, j, e\rangle\rangle$, then et and e are its immediate subexpressions.*

Definition 16 ([28]). *Two expressions have the same* external structure *whenever both are constants, de Bruijn indices, abstractions, applications or suspended terms or both are of the form nil, $et :: e$, $\{\!\{e_1, i, j, e_2\}\!\}$, $@l$, (t, l) or $\langle\langle et, i, j, e\rangle\rangle$.*

Expressions with the same external structure have an obvious correspondence between their subexpressions.

Lemma 12 ([28]). *Let et be an environment term such that $ind(et) \leq nl$. Then for $j \geq 1$, $\langle\langle et, nl + j, ol + j, et_1 :: \ldots :: et_j :: e\rangle\rangle \rhd^*_{rm} \langle\langle et, nl, ol, e\rangle\rangle$.*

Proof. Simple induction over j after repeated applications of the rule m_{10}. □

Now, we build a well-ordering compatible with SUSP. Two functions δ and μ that measure the work to be done when substitutions are propagated into the suspended terms. For a suspended term $[\![t, ol, nl, e]\!]$, these measures compute the complexity of the structure of the term t, since the substitutions are to be propagated into that term, as well as the complexity of the environment e, since this is constituted by the substitutions that have to be done between the term t as a simple examination of the rule r_4 makes it evident.

Definition 17 ([28]). *The measures δ, over expressions, and μ, over terms, are given by the table below.*

Category	*Expression*	$\delta(exp)$	$\mu(exp)$
TERM	*constant or* $\Diamond$	*0*	*1*
	$\underline{\mathtt{i}}$	*0*	*1*
	$(t_1\ t_2)$	$max(\delta(t_1), \delta(t_2))$	$max(\mu(t_1), \mu(t_2)) + 1$
	(λt)	$\delta(t)$	$\mu(t) + 1$
	$[\![t, ol, nl, e]\!]$	$\mu(t) + \delta(e)$	$\mu(t) + \delta(e) + 1$
ENVIRONMENT	nil	*0*	-
	$et :: e$	$max(\delta(et), \delta(e))$	-
	$\{\!\{e_1, nl, ol, e_2\}\!\}$	$\delta(e_1) + \delta(e_2) + 1$	-
ENVIRONMENT TERM	$@l$	*0*	-
	(t, l)	$\mu(t)$	-
	$\langle\langle et, nl, ol, e\rangle\rangle$	$\delta(et) + \delta(e) + 1$	-

Lemma 13 ([28]). *Let x, y be expressions of the same syntactical category and such that $\delta(x) \geq \delta(y)$ and if x and y are terms $\mu(x) \geq \mu(y)$. If y' results from x' by the replacement of subexpressions x with y, then $\delta(x') \geq \delta(y')$ and if x' and y' are terms, then $\mu(x') \geq \mu(y')$.*

Unfortunately, δ does not constitute the desired well-ordering because: in first place, for some rules, in particular for r_5, m_1, m_3, m_5 and m_{10}, we do not have a simplification of the expressions obtained because left-side and right-side terms of each rule have equal δ. And in second place, when we replace a subexpression t_1 with t_2 of t with $\delta(t_2) < \delta(t_1)$ not necessarily we obtain a expression t' with $\delta(t') < \delta(t)$. In order to resolve these problems δ is extended over the structure of expression as follows.

Definition 18 ([28]). *Given two expressions x_1 and x_2, we say that $x_1 \sqsupset x_2$ whenever either $\delta(x_1) > \delta(x_2)$ or $\delta(x_1) = \delta(x_2)$ and one of the following conditions holds:*

1. *$x_1 = \underline{\mathtt{i}}$ and $x_2 = \underline{\mathtt{j}}$, where $i > j$;*
2. *$x_1 = [\![t_1, ol_1, nl_1, e_1]\!]$, $x_2 = [\![t_2, ol_2, nl_2, e_2]\!]$ and $\delta(t_1) > \delta(t_2)$;*
3. *$x_1 = \{\!\{e_1, nl, ol, e_2\}\!\}$, $x_2 = et :: e$ and $x_1 \sqsupset e$;*
4. *x_1 and x_2 have the same external structure and identical corresponding immediate subexpressions except for a pair of immediate subexpressions x'_1 of x_1 and x'_2 of x_2 such that $x'_1 \sqsupset x'_2$;*
5. *x_2 is an immediate subexpression of x_1.*

Since $\sqsupset$ is not transitive, this relation is not a partial order; i.e., a transitive and anti-reflexive relation.

Definition 19 ([28]). *The relation $\succ$ is defined over expressions of the suspension calculus as the transitive closure of the relation $\sqsupset$.*

Proposition 6 ([28]). *The relation $\succ$ is a well-founded partial ordering over the expressions of the suspension calculus.*

Lemma 14 ([28]). *If x_1, x_2 are expressions such that $x_1 \triangleright_{rm} x_2$, then $\delta(x_1) \geq \delta(x_2)$.*

Lemma 15 ([28]). *If $l \rightarrow r$ is an instance of some of the rules of* SUSP *then $l \succ r$.*

Lemma 16 ([28]). *Let x_1 and x_2 be well-formed expressions of the language of the suspension calculus. If $x_1 \triangleright_{rm} x_2$ then $x_1 \succ x_2$.*

Lemma 17. *Let M be a well-formed term of the suspension calculus. Then $\lambda(M\ \underline{1}) \rightarrow_{Eta} N$ implies that $\mu(\lambda(M\ \underline{1})) \geq \mu(N)$.*

Proof. According to the *Eta*-rule implementation we have to consider the term $\triangleright_{rm}\text{-}nf(⟦M, 1, 0, (\Diamond, 0) :: nil⟧)$. We have that $\mu(\lambda(M\ \underline{1})) = max(\mu(M), 1) + 2 = \mu(M) + 2$ and $\mu(⟦M, 1, 0, (\Diamond, 0) :: nil⟧) = \mu(M) + \delta((\Diamond, 0) :: nil) + 1 = \mu(M) + max(\delta((\Diamond, 0), \delta(nil)) + 1 = \mu(M) + \delta((\Diamond, 0)) + 1 = \mu(M) + \mu(\Diamond) + 1 = \mu(M) + 2$. Now, by applying Lemma 16 we have that $⟦M, 1, 0, (\Diamond, 0) :: nil⟧ \succ \triangleright_{rm}\text{-}nf(⟦M, 1, 0, (\Diamond, 0) :: nil⟧) = N$ and from Lemma 13 we can conclude the proof. □

Lemma 18 (Compatibility of susp plus *Eta* and $\succ$). *Let x_1 and x_2 be well-formed expressions of the language of the λ_{SUSP}-calculus and suppose that either $x_1 \rightarrow_{\text{SUSP}} x_2$ or $x_1 \rightarrow_{Eta} x_2$. Then $x_1 \succ x_2$.*

Proof. By induction on the structure of expressions. If $x_1 \rightarrow_{\text{SUSP}} x_2$ apply Lemma 16. If $x_1 \rightarrow x_2$ is an instance of the *Eta* rule, it holds as immediate consequence of Lemma 17. If x_1 and x_2 have the same external structure, then by Lemma 13 we have that $\delta(x_1) \geq \delta(x_2)$. If $\delta(x_1) > \delta(x_2)$ the result is directly obtained from the definitions of $\sqsupset$ and $\succ$. When $\delta(x_1) = \delta(x_2)$, by induction hypothesis we have that applying the relation SUSP there exists an immediate subexpression x_1' of x_1 and corresponding immediate subexpression x_2' of x_2, such that $x_1' \succ x_2'$ and any other immediate subexpression of x_1 is identical to the corresponding immediate subexpression of x_2. Consequently, we can conclude that $x_1 \succ x_2$ □

Lemma 3 (susp plus *Eta* is SN). The rewriting system associated to SUSP and the *Eta* rule is noetherian.

Proof. Immediate consequence of Proposition 6 and Lemma 18. □