Comparing and Implementing Calculi of Explicit Substitutions with Eta-Reduction

Mauricio Ayala-Rincón¹, Flávio L. C. de Moura²

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

Fairouz Kamareddine

Mathematical and Computer Sciences, 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. It has also been shown that eta-reduction is useful for adapting substitution calculi for practical problems like higher order unification. This paper concentrates on rewrite rules for eta-reduction in three different styles of explicit substitution calculi: \( \lambda \sigma \), \( \lambda s_e \) and the suspension calculus. Both \( \lambda \sigma \) and \( \lambda s_e \) when extended with eta-reduction rules, have proved useful for solving higher order unification. We enlarge the suspension calculus with an adequate eta-reduction rule which we show to preserve termination and confluence of the associated substitution calculus and to correspond to the eta rules of the other two calculi. 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 in simulating one step beta-reduction.

After defining the eta-reduction rule in the suspension calculus, and after comparing these three calculi of explicit substitutions (all with an eta rule), we then concentrate on the implementation of the rewrite rules of eta-reduction in these calculi. We note that it is usual practice when implementing the eta rule for substitution calculi, to mix isolated applications of eta-reduction with the application of other rules of the corresponding substitution calculi. The main disadvantage of this practice is that the eta rewrite rules so obtained are unclean because they have an operational semantics different from that of the eta-reduction of the \( \lambda \)-calculus. For the three calculi in question enlarged with adequate eta rules we show how to implement these eta rules. For the \( \lambda s_e \) we build a clean implementation of the eta rule and we prove that it is not possible to follow the same approach for the \( \lambda \sigma \) and \( \lambda_{\text{susp}} \).

Key words: Explicit substitutions, \( \lambda \)-calculi, Eta-Reduction.
1 Introduction

Recent years have witnessed an explosion of work on explicit substitutions [1,7,9,16,17,21,23,4] and on its usefulness for: automated deduction and theorem proving [32,33], proof theory [40], programming languages [8,20,28,34] and higher order unification [3,15]. This paper studies three 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 [37,34], which introduces three different sets of entities: one for terms, one for environments and one for lists of environments.

(3) The $\lambda s$-style [23] which uses a philosophy of de Bruijn’s Automath [38] elaborated in the new item notation [22]. 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. [22]).

Desired properties of explicit substitution calculi include 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. The $\lambda\sigma$-calculus (without eta) satisfies a), b), d) and satisfies c) only when the set of open terms is restricted to those which admit metavariables of sort term. The $\lambda s$-calculus (without eta) satisfies a)...e) but not c). However, the $\lambda s$-calculus has an extension $\lambda s_e$ (again without eta) for which a)...c) holds, but e) fails and d) is unknown. The suspension calculus (which does not have eta) satisfies a) and when restricted to well formed terms it also satisfies b)...d). For the suspension calculus, e) is unknown.

The above discussion holds for these calculi without eta-reduction. However, work on higher order unification (HOU) in $\lambda s_e$ and $\lambda\sigma$ established the importance of combining eta-reduction (as well as expansion) with explicit substitutions. This has provided extensions of $\lambda s_e$ and $\lambda\sigma$ with eta-reduction rules also referred to by $\lambda s_e$ and $\lambda\sigma$ (cf. [15,3]). In fact, due to the importance of eta-reduction, calculi of explicit substitutions (including $\lambda\sigma$) have been extended with eta rules earlier than the application of $\lambda\sigma$ to HOU [19,39,12,29]. Eta-reduction (as well as expansion) is necessary for working with functions.

* Research partially supported by Grant 47488/01-6 of CNPq.

Email addresses: ayala@mat.unb.br (Mauricio Ayala-Rincón), flavio@mat.unb.br (Flávio L. C. de Moura), fairouz@macs.hw.ac.uk (Fairouz Kamareddine).

1 Author partially supported by the CNPq Brazilian Council.
2 Author supported by the Brazilian High Education Council CAPES.
and programs, since one needs to express functional or extensional equality. In particular, when the application of two lambda terms \( a \) and \( b \) to any term \( c \) yields the same result, then \( a \) and \( b \) should be considered equal.

Although \( \lambda s_e \) and \( \lambda \sigma \) have already been extended with eta-reduction, the suspension calculus still has not. This paper fills the gap and gives the first extension of the rewriting system of the suspension calculus with an eta-reduction rule bringing to it the advantages of the use of eta-reduction in substitutions calculi. Once the suspension calculus is extended with this eta-reduction rule, one can then compare these three calculi and assess the way eta-reduction should be implemented in each of them. This paper deals with three useful notions for these three calculi:

- Extending the suspension calculus with an eta-reduction rule resulting in \( \lambda \text{susp} \). We show the soundness of this rule and the confluence and strong normalisation of the underlying substitution calculus with eta.
- Comparing the adequacy of the reduction process of these three substitution calculi extended with eta-reduction, using the efficient simulation of \( \beta \)-reduction of [26] which showed that \( \lambda s_e \) and \( \lambda \sigma \) are non comparable. In this paper we show that \( \lambda s_e \) and \( \lambda \sigma \) as well as \( \lambda \sigma \) and \( \lambda \text{susp} \) are non comparable, that \( \lambda s_e \) is more adequate than \( \lambda \text{susp} \) for simulating one step beta-reduction.
- Reflecting on the correct definition and adequate implementation of the eta-reduction rewrite rules in these calculi. It is usual practice when implementing the eta rule for substitution calculi [11,2], to mix isolated applications of eta-reduction with the application of other rules of the corresponding substitution calculi. The main disadvantage of this practice is essentially that the eta rewrite rules so obtained are unclean because they have an operational semantics different from the one of the eta-reduction rule of the \( \lambda \)-calculus: the notion of functional equivalence embedded in the eta-reduction should be interpreted modulo the semantics of the corresponding substitution calculus. For the three calculi enlarged with adequate eta rules we show how to implement in practice these eta rules without mixing the isolated application of the eta-reduction with the application of other rules of the associated substitution calculi. The definition of a successful implementation depends on an effective specification of a practical method for evaluating the conditions of these eta rules which are conditional rules of the rewriting systems of the three treated calculi. For each of these explicit substitution calculi, our implementation consists basically of a linear verification along a term of the nonexistence of occurrences of the free variable of the eta-reduction while simultaneously upgrading all other free de Bruijn indices and without applying any additional rewrite rule of the corresponding substitution calculus. The three implementations are proved complete in the sense that they effectively simulate eta-reduction over pure lambda terms.
After including the necessary notations and motivation about explicit substitutions, in the second section, we present the $\lambda\sigma$, the $\lambda s_e$, and the suspension calculus. We enlarge the latter with an eta-reduction rule which is proved sound in the third section. Then, in the fourth and fifth sections, we compare the adequacy of these calculi in simulating one step beta-reduction and the appropriateness of the defined eta rewriting rules. Finally, and before concluding, we discuss the clean implementation of these eta rules in the sixth and seventh sections.

2 Preliminaries

We assume familiarity with the notion of term algebra $T(F, X)$ built on a (countable) set of variables $X$ and a set of operators $F$. Variables in $X$ are denoted by $X, Y, \ldots$ and for a term $a \in T(F, X)$, $\text{var}(a)$ 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 $\Rightarrow_R$ 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 $a \rightarrow_R^n b$, 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 $R_{-}\text{nf}(t)$ denotes its normal form. As usual we use indiscriminately either “noetherian” or “terminating” instead of SN.

A valuation is a mapping from $X$ to $T(F, X)$. The homomorphic extension of a valuation, $\theta$, from its domain $X$ to the domain $T(F, X)$ 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 $a \in T(F, X)$ will be written in postfix notation $a_\theta$. The domain of a grafting $\theta$, is defined by $\text{Dom}(\theta) = \{X \mid X\theta \neq X, X \in X\}$. Its range, is defined by $\text{Ran}(\theta) = \cup_{X \in \text{Dom}(\theta)} \text{var}(X\theta)$. We let $\text{var}(\theta) = \text{Dom}(\theta) \cup \text{Ran}(\theta)$. For explicit representations of a valuation and its corresponding grafting $\theta$, we use the notation $\theta = \{X \mapsto X\theta \mid X \in \text{Dom}(\theta)\}$. Note that the notion of grafting, usually called first order substitution, corresponds to simple syntactic substitution without renaming.

We use notations from [6] for the $\lambda$-calculus. Let $V$ be a (countable) set of variables denoted by lowercase last letters of the Roman alphabet $x, y, \ldots$.
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 $x \in \mathcal{V}$. We call $\lambda x.a$ resp. $(a b)$ abstraction resp. application terms.

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

But in this context problems arise forcing the use of $\alpha$-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 $(\lambda x.x)\theta = \lambda z.b$ 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, $\beta$-reduction, should be defined in a way that takes care of renaming bound variables when necessary to avoid harmful capture of variables.

$\alpha$-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 important to note 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 $\theta$ to a term $a$, which we denote in prefix notation $\theta a$ for discriminating substitution from grafting, can be conceived as a class of equivalence of 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
(η), where:

\[(\beta) \quad (\lambda_x.a \ b) \rightarrow \{x/b\}(a)\]

\[(\eta) \quad \lambda_x.(a \ x) \rightarrow a, \text{ if } x \notin \mathcal{F} \text{var}(a),\]

where \(\mathcal{F} \text{var}(a)\) denotes the free variables occurring in \(a\).

Note that our notion of substitution is not completely satisfactory because fresh variables depend on the history of the renaming process. \(\lambda\)-terms with meta-variables or open \(\lambda\)-terms are given by:

**Definition 2.3** Terms \(\Lambda(\mathcal{V}, \mathcal{X})\), of the \(\lambda\)-calculus with names and meta-variables are inductively defined by:

\[\Lambda(\mathcal{V}, \mathcal{X}) := x \mid X \mid \Lambda(\mathcal{V}, \mathcal{X}) \Lambda(\mathcal{V}, \mathcal{X}) \mid \lambda_x.\Lambda(\mathcal{V}, \mathcal{X}), \text{ where } x \in \mathcal{V} \text{ and } X \in \mathcal{X}.\]

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 \(\Lambda(\mathcal{V}, X)\) 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 is an effective way of clarifying the meaning of \(\lambda\)-terms and, for the unification process, of eliminating redundant renaming. De Bruijn proposed in [14] that names of bound variables be 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., \(T(\{(\_, \_), \lambda\} \cup \mathbb{N})\).

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 notation of de Bruijn is defined inductively as:

\[\Lambda_{dB}(\mathcal{X}) := n \mid X \mid (\Lambda_{dB}(\mathcal{X}) \Lambda_{dB}(\mathcal{X})) \mid \lambda_n.\Lambda_{dB}(\mathcal{X}), \text{ where } X \in \mathcal{X} \text{ and } n \in \mathbb{N} \setminus \{0\}.\]

\(\Lambda_{dB}(\mathcal{X})\)-terms without meta-variables are called closed \(\lambda\)-terms.
We write de Bruijn indices as $1, 2, 3, \ldots, n, \ldots$, to distinguish them from scripts. Since all considered calculi of explicit substitutions are built over the language of $\Lambda_{dB}(X)$, we will use $\Lambda$ to denote $\Lambda_{dB}(X)$.

Defining $\beta$-reduction in de Bruijn notation's as $(\lambda a \ b) \rightarrow \{1/b\}a$ (where $\{1/b\}a$ 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 $\{1/b\}$ crossing abstractors through $a$ the indices of the substitution (initially 1) and of the free variables in $b$ should be incremented.

Hence, we need new operators for detecting, incrementing and decrementing free variables.

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

1) $X^+_{i} = X$, for $X \in X$
2) $(a_1 a_2)^+_{i} = (a_1^+_{i} a_2^+_{i})$
3) $(\lambda a_1)^+_{i} = \lambda a_1^+_{i(i+1)}$
4) $n^+_{i} = \begin{cases} n + 1, & \text{if } n > i \\ n, & \text{if } n \leq i \end{cases}$

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

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

1) $\{n/b\}X = X$, for $X \in X$
2) $\{n/b\}(a_1 a_2) = \{n/b\}a_1 \{n/b\}a_2$
3) $\{n/b\}\lambda a_1 = \lambda(\{n+1/b^+\})a_1$
4) $\{n/b\}m = \begin{cases} m - 1, & \text{if } m > n \\ b, & \text{if } m = n \\ m, & \text{if } m < n \end{cases}$

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

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 $\Lambda_{dB}(X)$ by the $\beta$-reduction rule is CR.

In the $\lambda$-calculus with names, the $\eta$-reduction rule is defined by $\lambda a. (a \ x) \rightarrow a$, if $x \notin \text{Fvar}(a)$. In $\Lambda_{dB}(X)$, 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 $\Lambda_{dB}(X)$. “$a$ has no free occurrences of
\( x^n \) 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. Hence, there is in general, a term \( b \) such that \( b^+ = a \).

**Definition 2.8** \( \eta \)-reduction in the \( \lambda \)-calculus with de Bruijn indices is:

\[
\lambda(a \ 1) \rightarrow b \text{ if } b^+ = a.
\]

### 3 Calculi à la \( \lambda \sigma \), \( \lambda s_e \) and the Suspension Calculus

We present \( \lambda \sigma \), \( \lambda s_e \) and the Suspension Calculus. We enlarge the latter with an eta-reduction rule which we prove to be sound and to preserve the confluence of the suspension calculus.

#### 3.1 The \( \lambda \sigma \)-calculus

The \( \lambda \sigma \)-calculus is a first order rewriting system that contains the lambda calculus in de Bruijn notation and which makes explicit the substitutions started by \( \beta \)-reductions [1]. This calculus works on 2-sorted terms: (proper) terms (over which \( a, b, \ldots \) range), and substitutions (over which \( s, t, \ldots \) range). In this calculus, when a substitution \( \{n/b\} \) is applied to a term \( a \): \( \{n/b\}a \), we internalise this as \( a[1, \ldots, n-1. b. ↑^{n+1}] \). This means that all de Bruijn indices except \( n \) remain unchanged, while \( n \) is replaced with \( b \). Notice that \( b \) is placed at position \( n \) of the substitution list, which allows for simultaneous substitutions; for instance, \( a[b_1, b_2, \ldots ] \) replaces \( 1, 2, \ldots \) with \( b_1, b_2, \ldots \), respectively. Operationally, this calculus applies this kind of substitution decrementing by one the size of the substitution list as well as the de Bruijn indices. When doing that the operator \( ↑ \) is reached, \( a[↑^k] \) internalises the \( k \)-lifting of the term \( a \). In this calculus only \( 1 \) is used and the other de Bruijn indices are coded by lifting \( 1 \) as we will explain below. For details see [1].

**Definition 3.1** The \( \lambda \sigma \)-calculus is defined as the calculus of the rewriting system \( \lambda \sigma \) of Table 1 where

\[
\text{TERMS} \quad a ::= 1 \mid X \mid (a a) \mid \lambda a \mid a[s], \text{ where } X \in \mathcal{X}
\]

\[
\text{SUBSTITUTIONS} \quad s ::= id \mid ↑ \mid a.s \mid s \circ s
\]

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 \( 1 \), but we can code \( n \) by \( 1[↑^{n-1}] \).
Table 1
The \(\lambda\sigma\) Rewriting System of the \(\lambda\sigma\)-calculus with Eta rule

\[
\begin{array}{lll}
\text{(Beta)} & (\lambda a) b & \rightarrow a [b \cdot id] \\
\text{(App)} & (a b)[s] & \rightarrow (a [s]) (b [s]) \\
\text{(Abs)} & (\lambda a)[s] & \rightarrow \lambda(a [1 \cdot (s \circ \uparrow)]) \\
\text{(Clos)} & (a [s])[t] & \rightarrow a [s \circ t] \\
\text{(VarCons)} & 1 [a \cdot s] & \rightarrow a \\
\text{(Id)} & a[id] & \rightarrow a \\
\text{(Assoc)} & (s \circ t) \circ u & \rightarrow s \circ (t \circ u) \\
\text{(Map)} & (a \cdot s) \circ t & \rightarrow a [t] \cdot (s \circ t) \\
\text{(IdL)} & id \circ s & \rightarrow s \\
\text{(IdR)} & s \circ id & \rightarrow s \\
\text{(ShiftCons)} & \uparrow \circ (a \cdot s) & \rightarrow s \\
\text{(VarShift)} & 1 \cdot \uparrow & \rightarrow id \\
\text{(SCons)} & 1[s] \cdot (\uparrow \circ s) & \rightarrow s \\
\text{(Eta)} & \lambda(a 1) & \rightarrow b \text{ if } a =_\sigma b[\uparrow] \\
\end{array}
\]

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 \(=_\sigma\). We use \(\sigma\)-reduction, \(\sigma\)-normal form, etc., with the obvious meaning, in the case when reduction is restricted to the \(\sigma\)-rules.

The rewriting system \(\lambda\sigma\) is locally confluent [1], CR on substitution-closed terms (i.e., terms without substitution variables) [39] and not CR on open terms (i.e., terms with term and substitution variables) [13]. The possible forms of a \(\lambda\sigma\)-term in \(\lambda\sigma\)-normal form were given in [39] by:

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

In the \(\lambda\)-calculus with names or de Bruijn indices, the rule \(X \{y/a\} = X\), where \(y\) is an element of \(V\) 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 $a = a_\sigma b^{[\uparrow]}$ of the Eta rule is stronger than the condition $a = b^+$ given in Definition 2.8 as $X = X^+$, but there exists no term $b$ such that $X = a_\sigma b^{[\uparrow]}$. 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 Calculi à la $\lambda s$ and the $\lambda s_e$-calculus

Calculi à la $\lambda s$ avoid introducing two different sets of entities and insist on remaining close to the syntax of the $\lambda$-calculus using de Bruijn indices.

A term containing neither substitution nor updating operators is called a pure term. The role of the substitution operator is to internalise the substitution. Essentially, $aa^\sigma b$ makes operational the application of the substitution $\{n/b\}$ to $a$. This operator is propagated into the body of the abstractors, while all free de Bruijn indices (greater than $n$) are decreased by one. Once an occurrence of $n$ is found, $b$ is adequately modified (lifted) by the updating operator. The operational effect of $\varphi^i_j b$ is the $(i-1)$-lifting of all de Bruijn indices in $b$ greater than $j$. For details see [23,24].

Definition 3.2 (The $\lambda s$-calculus) Terms of the $\lambda s$-calculus are given by:

$$\Lambda s ::= N \mid \Lambda s \Lambda s \mid \lambda \Lambda s \mid \Lambda s \sigma^i \Lambda s \mid \varphi^i_k \Lambda s \quad \text{where} \quad i \geq 1, \ k \geq 0.$$  

The set of rules $\lambda s$ is given in Table 2.

The $\lambda s$-calculus was introduced in [23] with the aim of providing a calculus that preserves strong normalisation and has a confluent extension on open terms [24]. In [23,25], we establish the properties of these calculi which we list in the following theorem.

Theorem 3.3 The $s$-calculus is SN, the $\lambda s$-calculus is confluent on closed terms and satisfies PSN. Moreover, the $\lambda s$-calculus simulates $\beta$-reduction, is sound and has a confluent extension on open terms.

---

3 It can be argued that because we use de Bruijn indices, we remain close to de Bruijn’s philosophy rather than to the syntax of the $\lambda$-calculus and that instead it is calculi like $\lambda x$ of [10] and $\lambda \chi$ of [30] that remain close to the syntax of the lambda calculus. So, we need to explain here that by staying with the syntax of the $\lambda$-calculus we mean that we do not introduce substitutions and other category of operators separately as in $\lambda\sigma$, but that a term for us is either an abstraction term, an application term, a substitution term or an updating term.
We introduce the open terms and the rules that extend \( \lambda s \) to obtain the \( \lambda s_e \)-calculus.

**Definition 3.4** The set of open terms, noted \( \Lambda_{spo} \), is given as follows:

\[
\Lambda_{spo} ::= V | N | \Lambda_{spo} \Lambda_{spo} | \lambda \Lambda_{spo} \sigma^i \Lambda_{spo} | \varphi_k \Lambda_{spo} \quad \text{where} \quad i \geq 1, \ k \geq 0
\]

and where \( V \) stands for a set of variables, over which \( X, \ Y, \ ... \) range. We take \( a, b, c \) to range over \( \Lambda_{spo} \). Furthermore, closures, pure terms and compatibility are defined as for \( \Lambda s \).

Working with open terms one loses confluence as shown by the following counterexample:

\[
((\lambda X)Y)\sigma^1 \lambda \rightarrow (X\sigma^1 Y)\sigma^1 \lambda \quad ((\lambda X)Y)\sigma^1 \lambda \rightarrow ((\lambda X)\sigma^1 \lambda)(Y\sigma^1 \lambda)
\]

and \( (X\sigma^1 Y)\sigma^1 \lambda \) and \( ((\lambda X)\sigma^1 \lambda)(Y\sigma^1 \lambda) \) have no common reduct. Moreover, the above example shows that even local confluence is lost. But since

\[
((\lambda X)\sigma^1 \lambda)(Y\sigma^1 \lambda) \rightarrow (X\sigma^2 \lambda)\sigma^1 (Y\sigma^1 \lambda)
\]

the solution to the problem seems at hand if one has in mind the properties of meta-substitutions and updating functions of the \( \lambda \)-calculus in the Bruijn notation. These properties are equalities which can be given a suitable orientation and the new rules, thus obtained, added to \( \lambda s \) yield a rewriting system which happens to be locally confluent. For instance, the rule corresponding to the meta-substitution lemma is the \( \sigma-\sigma \)-transition rule. The addition of this rule solves the critical pair in our counterexample, since now we have
(Xσ^1Y)σ^1 1 → (Xσ^2 2)σ^1(Y σ^1 1).

**Definition 3.5** The set of rules λs_e is obtained by adding the rules given in Table 3 to the set λs. The λs_e-calculus is the reduction system (Λs_op, →_λs_e)

Table 3

<table>
<thead>
<tr>
<th>The new rules of the λs_e-calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td>σ-σ-transition</td>
</tr>
<tr>
<td>σ-ϕ-transition 1</td>
</tr>
<tr>
<td>σ-ϕ-transition 2</td>
</tr>
<tr>
<td>ϕ-σ-transition</td>
</tr>
<tr>
<td>ϕ-ϕ-transition 1</td>
</tr>
<tr>
<td>ϕ-ϕ-transition 2</td>
</tr>
</tbody>
</table>

where →_λs_e is the least compatible reduction on Λs_op generated by the set of rules λs_e. The calculus of substitutions associated with the λs_e-calculus is the rewriting system generated by the set of rules s_e = λs_e − {σ-generation} and we call it s_e-calculus.

The equational theory associated to the rewriting system λs_e defines a congruence =_λs_e. The congruence obtained by dropping σ-generation and Eta (that will be defined below in Table 4) is denoted by =_s_e. The set of variables of sort TERM in a term a ∈ Tλs_e(λ) is denoted by T var(a).

We can describe the operators of the λs_e-calculus over the signature of a first order sorted term algebra Tλs_e(λ) built on λ, the set of variables of sort TERM and its subsort NAT⊂TERM by:

- \( _\text{n} \) : \( \rightarrow \text{NAT}, \forall n \in \mathbb{N} \setminus \{0\} \)
- \((_,_): \text{TERM} \times \text{TERM} \rightarrow \text{TERM}\)
- \(\sigma^i_,: \text{TERM} \times \text{TERM} \rightarrow \text{TERM}, \forall i \in \mathbb{N} \setminus \{0\}\)
- \(\lambda_,: \text{TERM} \rightarrow \text{TERM}\)
- \(\varphi^i_k_,: \text{TERM} \rightarrow \text{TERM}, \forall i \in \mathbb{N}, k \in \mathbb{N} \setminus \{0\}\)

In [24] we proved the following:

**Theorem 3.6 (WN and CR of s_e)** The s_e-calculus is weakly normalising and confluent.

**Lemma 3.7 (Simulation of β-reduction)** Let a, b ∈ λ, if \(a \rightarrow_\beta b\) then \(a \not\rightarrow_\lambda s_e b\).
Theorem 3.8 (CR of \( \lambda s_e \)) The \( \lambda s_e \)-calculus is confluent on open terms.

Theorem 3.9 (Soundness) Let \( a, b \in \lambda \), if \( a \rightarrow_{\lambda s_e} b \) then \( a \rightarrow_{\beta} b \).

In [3] we proved that:

Proposition 3.10 \( \mathcal{X} \)-grafting and \( \lambda s_e \)-reduction commute.

This calculus was originally introduced without the \( Eta \) rule that was added in [3] to deal with higher order unification problems as originally done in [15] for the \( \lambda \sigma \)-calculus.

Table 4

<table>
<thead>
<tr>
<th>The eta rule of the ( \lambda s_e )-calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td>((Eta)) ( \lambda(a , \mathbf{1}) \rightarrow b ) if ( a = s_c \varphi^2_0 b )</td>
</tr>
</tbody>
</table>

The characterization of the \( \lambda s_e \)-normal forms was given in [24,3] by: a term \( a \in \Lambda s_e \) is a \( \lambda s_e \)-nf if and only if one of the following holds:

1. \( a \in \mathcal{X} \cup \mathbb{N} \);
2. \( a = bc \) with \( b, c \) in \( \lambda s_e \)-nf and \( b \) not an abstraction \( \lambda d \);
3. \( a = \lambda b \), where \( b \) is a \( \lambda s_e \)-nf excluding applications of the form \( (c \, \mathbf{1}) \) where \( \varphi^2_0 d = s_e c \) for some \( d \);
4. \( a = b\sigma^j c \), where \( b, c \) in \( \lambda s_e \)-nf and \( b \) is of the form: \( X \) or \( d\sigma^j e \), with \( j < i \) or \( \varphi^i_k d \), with \( j < k \);
5. \( a = \varphi^i_k b \), where \( b \) is a \( \lambda s_e \)-nf of the form: \( X \) or \( c\sigma^j d \), with \( j > k + 1 \) or \( \varphi^j_k c \), with \( k < l \);

3.3 The Suspension Calculus

The suspension calculus [37,34] 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 [36].

The suspension calculus works with three different types of entities:

| SUSPENDED TERMS | \( M, N \) := \( C \mid n \mid \lambda M \mid (M \, N) \mid [M, i, j, e_1] \) |
| ENVIRONMENTS | \( e_1, e_2 \) := \( nil \mid et :: e_1 \mid \{e_1, i, j, e_2\} \) |
| ENVIRONMENT TERMS | \( et \) := \( @i \mid (M, i) \mid \{(et, i, j, e_1)\} \) |

where \( C \) 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. Rather than performing adjustments at each stage,
the suspension calculus notation performs the adjustments into a substitution term only at the final substitution stage. Intuitively, a suspended term of the form \([M, i, j, e_1]\) means that the first \(i\) variables of the \(\lambda\)-term \(M\) must be substituted in a way determined by the environment \(e_1\) and its remaining bound variables must be renumbered according to the fact that \(M\) used to appear within \(i\) abstractions but now appears within \(j\) of them.

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 5.

As in [37] we denote by \(\triangleright_{rm}\) the reduction relation defined by the \(r\)- and \(m\)-rules in Table 5. The associated substitution calculus, denoted by \(\text{SUSP}\), is the one given by the congruence \(\equiv_{rm}\).

**Definition 3.11 ([37])** The length \(\text{len}(e)\) of an environment \(e\) is given by:

\[
\text{len}(\text{nil}) := 0; \quad \text{len}(e :: e') := \text{len}(e') + 1 \quad \text{and}
\]

\[
\text{len}(\{e_1, i, j, e_2\}) := \text{len}(e_1) + (\text{len}(e_2) - i).
\]

The index \(\text{ind}(et)\) of an environment term \(et\), and the \(l\)-th index \(\text{ind}_l(e)\) of environment \(e\) and natural number \(l\), are simultaneously defined by induction on the structure of expressions:

\[
\begin{align*}
\text{ind}(\@m) &= m + 1; \\
\text{ind}((t', m)) &= m; \\
\text{ind}(\{t', j, k, e\}) &= \begin{cases} \\
\text{ind}_m(e) + (j - k) & \text{if } \text{len}(e) > j - \text{ind}(et') = m \\
\text{ind}(et') & \text{otherwise} \end{cases} \\
\text{ind}_0(\text{nil}) &= 0; \\
\text{ind}_0(e :: e') &= \text{ind}(et) \quad \text{and} \quad \text{ind}_{l+1}(et :: e') = \text{ind}_l(e')
\end{align*}
\]

\[
\begin{align*}
\text{ind}_l(\{e_1, j, k, e_2\}) &= \begin{cases} \\
\text{ind}_m(e_2) + (j - k) & \text{if } l < \text{len}(e_1) \text{ and } \\
\text{len}(e_2) > m = j - \text{ind}_l(e_1) & \text{if } l < \text{len}(e_1) \text{ and } \\
\text{len}(e_2) \leq m = j - \text{ind}_l(e_1) & \text{if } l \geq l_1 = \text{len}(e_1)
\end{cases}
\end{align*}
\]

The index of an environment \(e\), denoted as \(\text{ind}(e)\), is \(\text{ind}_0(e)\).

**Definition 3.12 ([37])** 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 \([t, ol, nl, e]\) then \(\text{len}(e) = ol\) and \(\text{ind}(e) \leq nl\)
- if \(s\) is \(et :: e\) then \(\text{ind}(e) \leq \text{ind}(et)\)
- if \(s\) is \(\{et, j, k, e\}\) then \(\text{len}(e) = k\) and \(\text{ind}(et) \leq j\)
Lemma 3.13 ([37])

A well-formed expression of the suspension calculus characterizes the \( N \).

Nadathur conjectures that the suspension calculus preserves strong normal-suspension calculus.

The suspension calculus simulates \( \beta \)-reduction and its associated substitution calculus \( \text{SUSP} \) is CR (over closed and open terms) and SN [37]. In [34] Nadathur conjectures that the suspension calculus preserves strong normalization too but there is still no proof of this conjecture. The following lemma characterizes the \( \text{r}_{\text{rm}} \)-normal forms.

**Lemma 3.13 ([37])** A well-formed expression of the suspension calculus \( x \) is
in its $\triangledown_{rm}$-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 $\triangledown_{rm}$-nf;
3) $x$ is the environment nil or $et :: e$ for $et$ and $e$ resp. an environment term and an environment in $\triangledown_{rm}$-nf.

3.4 The suspension calculus enlarged with $\eta$-reduction: the $\lambda_{susp}$-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 $\lambda_{susp}$ and we continue to call its associated substitution calculus susp. The Eta rule is formulated in Table 6. Intuitively Eta may be interpreted as: when

Table 6

| (Eta) | $\lambda (t_1 \uparrow)$ | $\rightarrow$ | $t_2$ if $t_1 \triangledown_{rm} [t_2, 0, 1, nil]$ |

it is possible to apply the $\eta$-reduction rule to the redex $\lambda(t_1 \uparrow)$ 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 $\uparrow$ in $t_1$. Proposition 3.16 proves the correctness of Eta according to this interpretation.

Remark 3.14 The reader may wonder whether this is the best formulation of Eta in the suspension calculus. Indeed, the reader may ask this question also in connection with the formulation of Eta in both the $\lambda\sigma$- and $\lambda_{se}$-calculi. Initially, [15] intended to use $\lambda(a[\uparrow]\uparrow) \rightarrow a$ as a formulation of Eta in the $\lambda\sigma$-calculus. However, this formulation would lead to an infinite set of critical pairs. For this reason, [15] took the formulation given in Table 1. The same reason led [3] to use a formulation of Eta in the $\lambda_{se}$-calculus which uses $s_e$ convertibility (see table 4). And indeed for the suspension calculus, we also get an infinite set of critical pairs if we use $(\lambda ([t_1, 0, 1, nil] \uparrow)) \rightarrow t_1$.

We follow [11] and [2] for $\lambda\sigma$ and $\lambda_{se}$ respectively, and implement the Eta rule of the $\lambda_{susp}$-calculus by introducing a dummy symbol $\bowtie$, by:

$\lambda(M \uparrow) \rightarrow_{Eta} N$ if $N = \triangledown_{rm}$-nf$([M, 1, 0, (\bowtie, 0) :: nil])$ and $\bowtie$ does not occur in $N$.

The correctness of this implementation is explained because an $\eta$-reduction $\lambda(M \uparrow) \rightarrow_{\eta} N$ 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 \( \triangleright \text{rm} \) -normalization of the term \((\lambda M) \Diamond) \rightarrow_{\beta_s} [M,1,0,(\Diamond,0):nil]\), whenever \( \Diamond \) does not appear in this normalized term.

**Lemma 3.15** Let \( A \) be a well-formed term of the suspension calculus. Then the \( \text{susp} \)-normalization of the term \([A,k,k+1,@k::@k-1:::\vdots::@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.

- If \( A = @n \). If \( n > k \): \([@n,k,k+1,@k:::\vdots::@1::nil]\] \( \rightarrow_{r_5}^{k} \), \([@n-k,0,k+1,nil]\] \( \rightarrow_{r_5} @n+1 \).
  
  If \( n \leq k \): \([@n,k,k+1,@k:::\vdots::@1::nil]\] \( \rightarrow_{r_5}^{n-1} \), \([1,k-n+1,k+1,@k-n+1:::\vdots::@1::nil]\] \( \rightarrow_{r_5} n \).

- If \( A = (BC) \), we apply \( r_6 \) and induction hypothesis for \( B \) and \( C \);

- If \( A = (\lambda B) \). Since \( B \) is bounded by an abstractor, only its free variables greater than \( k \) should be incremented by one, the other variables remain unchanged. Since \([((\lambda B),k,k+1,@k::\vdots::@1::nil]\] \( \rightarrow_{r_7} \lambda[B,k+1,k+2,@k+1::\vdots::@1::nil], \) by applying induction hypothesis over the previous term we obtain the desired result.

- If \( A = [t,ol,nl,e] \). Without loss of generality \( A \) may be \( \triangleright \text{rm} \) -normalized and by Lemma 3.13 the obtained term is of one of the forms analysed in the previous cases.

\[ \square \]

**Proposition 3.16 (Soundness of the Eta rule)** Every application of the Eta rule of \( \lambda_{\text{susp}} \) to the redex \( \lambda(t_1 \text{1}) \) 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 \( t_1 =_{\triangleright \text{rm}} [t_2,0,1,nil] \). The effect of normalizing \([t_2,0,1,nil]\) is to increment by one all de Bruijn free indices occurring at \( t_2 \):

- \( t_2 = @n \). \([@n,0,1,nil]\] \( \rightarrow_{r_2} @n+1 =_{\triangleright \text{rm}} t_1 \).

- If \( t_2 = (AB) \). Without loss of generality we can assume that both \( A \) and \( B \) are in \( \triangleright \text{rm} \) -nf. Observe that \([((AB),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 \([((AB),0,1,nil]\).

- If \( t_2 = (\lambda A) \). As before, assume \( A \) is in \( \triangleright \text{rm} \) -nf. Note that \([((\lambda A),0,1,nil]\] \( \rightarrow_{r_7} (\lambda[A,1,2,@1::nil]. \) By applying Lemma 3.15 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.

- If \( t_2 = [t,i,j,e] \). If \( t \) is in \( \triangleright \text{rm} \) -nf then \([t,i,j,e] \triangleright \text{rm} t', where \( t' \) is a pure \( \lambda \)-term in de Bruijn notation by Lemma 3.13. Hence, the analysis given in
the previous three cases applies here too.

Noetherianity of \texttt{susp} plus the \texttt{Eta} rule enables us to apply the Newman diamond lemma and the Knuth-Bendix critical pair criterion for proving its confluence.

**Lemma 3.17 (susp+ Eta is SN)** The rewriting system associated to \texttt{susp} and the \texttt{Eta} rule is noetherian.

**Proof.** (Sketch) This is proved by showing that the \texttt{Eta} rule is also compatible with the well-founded partial ordering \(<\) that is defined and proved compatible with \(\triangledown\) in [37].

A simple environment is an environment without subexpressions of the form \(\{\ldots,\ldots,\ldots\}\) or \(\langle\langle\ldots,\ldots,\ldots\rangle\rangle\).

**Lemma 3.18 ([37])** Let \(e_1\) be a simple environment and suppose that \(\text{nl}\) and \(\text{ol}\) are naturals such that \((\text{nl} - \text{ind}(e_1)) \geq \text{ol}\). Then \(\{\text{nl, ol, e_1}\} \triangledown_{\text{rm}}^* e_1\).

**Lemma 3.19 (Local-confluence of susp+ Eta)** The rewriting system of the substitution calculus \texttt{susp} plus the \texttt{Eta} rule is locally-confluent.

**Proof.** The rewrite relation \(\triangledown_{\text{rm}}\), i.e., \texttt{susp}, was shown in [37] to be (locally) confluent. Thus for proving that the associated rewriting system enlarged with the \texttt{Eta} rule is locally-confluent, it is enough to show that all additional critical pairs built by overlapping between the \texttt{Eta} rule and the other rules of \texttt{susp} are joinable. Note that no critical pairs are generated from \texttt{Eta} and itself. Moreover, there is a unique overlapping between the set of rules in Table 5 (minus (\(\beta_s\))) and \texttt{Eta}: namely, the one between \texttt{Eta} and (\(r_7\)).

This critical pair is \(\langle\langle t_2, \text{ol}, \text{nl}, e_1\rangle\rangle, \lambda(\langle\langle n, \text{ol}, \text{nl}, e_1\rangle\rangle, \text{nl} + 1, \text{nl} + 1, @\text{nl} :: e_1\rangle\rangle\), where \(t_1 =_{\text{rm}} [t_2, 0, 1, \text{nil}]\). After applying the rules \(r_6\) and \(r_3\) the right-side term of this critical pair reduces to \(\lambda(\langle\langle n, \text{ol}, \text{nl}, e_1\rangle\rangle, 1)\).

We prove by analyzing the structure of \(t_1\) that this critical pair is joinable. We take \(t_1\) and \(t_2\) as \(\triangledown_{\text{rm}}\)-nf’s.

- \(t_1 = n\). For making possible the \texttt{Eta} application, we need that \(n > 1\). According to the length of the environment @\text{nl} :: e (i.e., \(\text{ol} + 1\)) we have the following cases:
  - \(\text{ol} + 1 < n\). On one side, \(\lambda(\langle n, \text{ol} + 1, \text{nl} + 1, \text{@nl} :: e_1\rangle) \rightarrow_{r_5}^{n+1} \lambda(\langle n, \text{ol} + 1, \text{nl} + 1, \text{nil} \rangle) \rightarrow_{r_2} \lambda(n, \text{nl} + \text{nl} + 1) \rightarrow_{\text{Eta}} n, \text{nl} + \text{nl} + 1, e\). On the other side, \(t_1 =_{\text{rm}} [t_2, 0, 1, \text{nil}]\), hence \(t_2 = n \rightarrow_{r_5} n, \text{nl} + \text{nl} + 1, \text{nil} \rightarrow_{r_2} n, \text{nl} + \text{nl} + 1\).
  - \(\text{ol} + 1 \geq n\). On one side, \(\lambda(\langle n, \text{ol} + 1, \text{nl} + 1, \text{@nl} :: e_1\rangle) \rightarrow_{r_5}^{n+1} \lambda(\langle\langle n, \text{ol} + 1, \text{nl} + 1, \text{nil} \rangle\rangle, \text{nl} + 1, @\text{nl} :: e_1 \rangle\rangle\) and the subsequent derivation depends
on the structure of $e_1$: when $e_1 = @l$ we apply $r_3$ obtaining $\lambda(nl_{1+1-1} 1)$
$\rightarrow_{Eta} \eta l_{n-2}$ and on the other side, $[n, ol, nl, e] \rightarrow_{r_3} [1, ol - n + 2, nl, @l :: e'] \rightarrow_{r_3} \eta l_{n-2}$; when $e_1 = (t, l)$, where without loss of
generality $t$ is supposed to be in $\eta l_{n-1}$, we have
$\lambda([1, ol - n + 2, nl + 1, (t, l) :: e'] 1) \rightarrow_{r_4} \lambda([1, ol - n + 1, nl, e] 1) \rightarrow_{Eta}
\eta l_{n-1}$ and then apply the induction hypothesis. That is, suppose inductively that
and $\lambda([1, ol - n + 2, nl + 1, (t, l) :: e'] 1) \rightarrow_{r_3} \eta l_{n-2}$, we can separately consider
$\eta l_{n-1}$.

• $t_1 = (A B)$. Since the sole rule of the $\lambda_{susp}$ 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
$\lambda([A, ol + 1, nl + 1, @nl :: e] 1) \rightarrow_{Eta} A''$ and $[A', ol, nl, e]$, where
$\eta l_{n-1}$ as well as $\lambda([B, ol + 1, nl + 1, @nl :: e] 1) \rightarrow_{Eta} B''$ and
$[B', ol, nl, e]$, where $B', 0, nl + 1, @nl :: e$ $\rightarrow_{r_6}$ $B$ are joinable. Then since
$\lambda([A (B), ol + 1, 1, @nl :: e] 1) \rightarrow_{r_6}$
$\lambda([A (B), ol + 1, nl + 1, @nl :: e] 1) \rightarrow_{Eta} (A'' B'')$
and $[A (B'), ol, nl, e] \rightarrow_{r_6} ([A', ol, nl, e] [B', ol, nl, e])$ we can conclude the
confluence.

• $t_1 = (\lambda A)$. By the $Eta$ rule implementation, it is enough to show the joinability
of the $Eta$-reduction of the term $\lambda(\lambda(A), ol + 1, nl + 1, @nl :: e) 1$
that is $\eta l_{n-1}$.

On the one side, $[\eta l_{n-1}, 1, 0, (\eta l :: nil)]$, $ol, nl, e]$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$. Then we obtain the term

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.

$\eta l_{n-1}$.
Then we obtain the term $(\text{lambda} \text{use} \text{-nf}([A, \text{ol} + 2, nl + 1, @nl :: (\Diamond, \text{nl}) :: e])).$

The sole difference of the obtained suspended terms is the second environment term of their environments, that is $\langle\langle (3, 0), 0, \text{ol}, e \rangle\rangle$ and $(3, \text{nl})$. But since the Eta rule applies, when propagating the substitution between these suspended terms, the dummy symbol and hence these second environment terms should disappear. 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 3.20 (Confluence of susp+ Eta)** The calculus SUSP jointly with the Eta rule, is confluent.

### 4 Comparing the adequacy of the calculi

According to the criterion of adequacy introduced in [26] we prove that the $\lambda\sigma$ and the $\lambda_{\text{sus}}$ 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 in the simulation of one step $\beta$-reduction than the $\lambda_{\text{sus}}$.

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

**Definition 4.1 ([26]) (Adequacy)** Let $\xi_1, \xi_2 \in \{\sigma, s_e, \text{SUSP}\}$. The $\lambda_{\xi_1}$-calculus is more adequate (in simulating one step $\beta$-reduction) than the $\lambda_{\xi_2}$-calculus, denoted $\lambda_{\xi_1} \prec \lambda_{\xi_2}$, if:

- for every $\beta$-reduction $a \rightarrow_{\beta} b$ and every $\lambda_{\xi_2}$-simulation $a \rightarrow_{\lambda_{\xi_2}}^n b$ there exists a $\lambda_{\xi_1}$-simulation $a \rightarrow_{\lambda_{\xi_1}}^m b$ such that $m \leq n$;
- there exists a $\beta$-reduction $a \rightarrow_{\beta} b$ and a $\lambda_{\xi_1}$-simulation $a \rightarrow_{\lambda_{\xi_1}}^m b$ such that for every $\lambda_{\xi_2}$-simulation $a \rightarrow_{\lambda_{\xi_2}}^n b$ 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 [26] apply also to the incomparability of $\lambda\sigma$ and $\lambda_{s_e}$ since $\lambda_{s_e}$ is an extension of $\lambda s$ for open terms.
Proposition 4.2 The $\lambda\sigma$- and the $\lambda s_e$-calculi are non comparable.

Lemma 4.3 Every $\lambda\sigma$-derivation of $(\lambda \lambda 2 \ 1)$ 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.

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

In the following lemmas, $(M \ 1^n)$ is a shorthand for $n$ applications of $1$, i.e., $(\ldots((M \ 1)1)\ldots1)$.

Lemma 4.4 Every $\lambdaSusp$-derivation of $(\lambda \lambda(2 \ 2)) \ 1^n$ to its $\lambdaSusp$-nf has length $4n + 5$.

Proof. In fact, note that the sole possible derivation is:

$(\lambda \lambda(2 \ 2)) \ 1^n \rightarrow_{\beta_s} (\lambda(\lambda(2 \ 2)) \ 1, 0, (1^n, 0) :: nil) \rightarrow_{\gamma_s} \lambda(n(2 \ 2), 1, 0 :: (1^n, 0) :: nil) \rightarrow_{r_6} \lambda(2, 2, 1, 0 :: (1^n, 0) :: nil) \rightarrow_{r_5} \lambda(2^n, 0, 1, nil) \rightarrow_{2^n} \lambda(2^n, 0, 1, nil) \rightarrow_{2^n} \lambda(2^n, 0, 1, nil) = (\lambda(2^n, 0, 1, nil))^n$.

Lemma 4.5 (26) There exists a derivation of $(\lambda \lambda(2 \ 2)) \ 1^n$ to its $\lambda\sigma$-nf whose length is $n + 9$.

Proof. Consider the following derivation:
Proof. By simple induction over the structure of the same lemma formulated for the \( M \) length of the \( \sigma \) and \( \lambda \text{susb}-\text{calculus are non comarable.} \)

**Proposition 4.6** Let \( \lambda A, B, C \in \Lambda \) and \( k \geq 0 \). We define the functions \( M : \Lambda \rightarrow \mathbb{N} \) and \( Q_k : \Lambda \times \Lambda \rightarrow \mathbb{N} \) by:

- \( M(\text{n}) = \text{n} \)
- \( M(\lambda A) = M(A) + 1 \)
- \( M(A, B) = M(A) + M(B) + 1 \)
- \( Q_k(\varphi^i_k(A)) = \varphi^i_k(A) + Q_k(B, C) + 1 \)
- \( Q_k(\lambda A, B) = Q_{k+1}(A, B) + 1 \)

**Lemma 4.8** Let \( A \in \Lambda \). Then all \( s_{e^i_e} \)-derivations of \( \varphi^i_k(A) \) to its \( s_{e^i_e} \)-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 [26].

**Lemma 4.9** Let \( A \in \Lambda \). Then all \( \text{susb} \)-derivations of the well-formed term \( [A, i, i, @i - 1 :: \ldots :: @0 :: \text{nil}] \) to its \( \text{susb} \)-nf have length greater than or equal to \( M(A) \).

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

- \( A = \text{n} \). If \( n > i \) then \( [\text{n}, i, i, @i - 1 :: \ldots :: @0 :: \text{nil}] \rightarrow_r^i [\text{n}, i, 0, i, \text{nil}] \) → \( r_2 \) \( \text{n} \). The length of the derivation is \( i + 1 \geq M(A) \). If \( n \leq i \) then
Proof.

• \[ (A C) \]. We have that \([B, i, i, @i - 1 : : : @0 :: nil] \rightarrow_{r_6} \]
  \([B, i, i, @i - 1 : : : @0 :: nil] [C, i, i, @i - 1 : : : @0 :: nil] \). By induction hypothesis we conclude that the length of the derivation is greater than or equal to \(1 + M(A) = M(B) = M(A) \).

• \[ A = (\lambda B) \]. We have that \([\lambda B), i, i, @i - 1 : : : @0 :: nil] \rightarrow_{r_7} \]
  \(\lambda[B, i + 1, i + 1, @i : : : @0 :: nil] \). 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 4.10 Let \(B \in \Lambda \) and \(i, j \geq 0 \). The derivation of the susp-term \([B, j, @j - 1 :: e] \) to its susp-nf has length greater than or equal to \(M(B) \).

Proof. By structural induction over \(A \).

• \[ A = n \]. If \(n = k\) then \([n, k, k - 1, @k - 2 : : : @0 :: (B, l) :: nil] \rightarrow_{r_5}^{n - 1} \]
  \([1, k - n + 1, k - 1, @k - n - 1 : : : @0 :: (B, l) :: nil] \rightarrow_{r_3} n \). This derivation has length \(n \geq Q_k(n, B) \).

If \(n = k\) then \([n, k, k - 1, @k - 2 : : : @0 :: (B, l) :: nil] \rightarrow_{r_5}^{n - 1} \]
  \([1, k - 1, (B, l) :: nil] \rightarrow_{r_4} [B, 0, k - 1 = l, nil] \). By Lemma 4.10 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 \(n + M(B) = Q_k(n, B) = Q_k(A, B) \).

If \(n > k\) then \([n, k, k - 1, @k - 2 : : : @0 :: (B, l) :: nil] \rightarrow_{r_5}^{k} [n - k, 0, k - 1, nil] \rightarrow_{r_2} n - 1 \). Derivation whose length is \(k + 1 \geq Q_k(n, B) = Q_k(A, B) \).

• \[ A = (C D) \]. \([[(C D), k, k - 1, @k - 2 : : : @0 :: (B, l) :: nil] \rightarrow_{r_6} \]
  \([C, k, k - 1, @k - 2 : : : @0 :: (B, l) :: nil] \rightarrow_{r_6} [[D, k, k - 1, @k - 2 : : : @0 :: (B, l) :: nil] \).

By induction hypothesis the derivation has length greater than or equal to \(1 + Q_k(C, B) + Q_k(D, B) = Q_k((C D), B) = Q_k(A, B) \).

• \[ A = \lambda C \]. \([[(\lambda C), k, k - 1, @k - 2 : : : @0 :: (B, l) :: nil] \rightarrow_{r_7} \]
  \(\lambda[C, k + 1, k, @k - 1 : : : @0 :: (B, l) :: nil] \). 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 4.12** Let $A, B \in \Lambda$ and $k \geq 1$. $s_e$-derivations of $A\sigma^k B$ to its $s_e\text{-}nf$ have length $\leq Q_k(A, B)$.

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

- $A = n$. By applying the $\sigma$-destruction rule, in the case $n \neq k$, we obtain either $n - 1$ or $n$ and in the case $n = k$, $\varphi^k_0 B$. In the case that $n \neq k$, the derivation has length equal to $1 \leq Q_k(n, B)$. In the other case, we apply Lemma 4.8 obtaining that the complete $s_e$-normalization has length $1 + M(B)$. In both cases the derivation has length less than or equal to $Q_k(n, B)$.

- $A = (CD)$. $(CD)\sigma^k B \rightarrow (C\sigma^k B D\sigma^k B)$. By applying the induction hypothesis we conclude that the complete derivation has length less than or equal to $1 + Q_k(C, B) + Q_k(D, B) = Q_k((CD), B)$.

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

**Theorem 4.13** ($\lambda s_e < \lambda \text{susp}$) The $\lambda s_e$ is more adequate in the simulation of one step $\beta$-reduction than the $\lambda \text{susp}$-calculus.

**Proof.** We prove the stronger result that if $A \in \Lambda$ and $A \rightarrow^* B \rightarrow^m_{susp} \text{susp-nf}(B)$ is a $\lambda \text{susp}$-simulation of a $\beta$-reduction then: $A \rightarrow^{\sigma\text{-generation}} C \rightarrow^n_{s_e} s_e\text{-nf}(C)$ has length $n + 1 \leq m + 1$.

In $\lambda \text{susp}$, for any redex of $\beta$ we have $(\lambda D) E \rightarrow^\beta [D, 1, 0, (E, 0)::nil] \rightarrow^m_{susp} \text{susp-nf}([D, 1, 0, (E, 0)::nil]).$ In the $\lambda s_e$, $(\lambda D) E \rightarrow^{\sigma\text{-generation}} D\sigma^1 E \rightarrow^n_{s_e} s_e\text{-nf}(D\sigma^1 E)$. By Propositions 4.11 and 4.12, $m \geq Q_1(D, E) \geq n$. Hence, the length of a $\lambda \text{susp}$-simulation of a $\beta$-reduction 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 $(\lambda \overline{2}) \overline{1} \rightarrow^\beta \overline{1}$. In $\lambda \text{susp}$ the only possible three steps simulation is: $(\lambda \overline{2}) \overline{1} \rightarrow^\beta [\overline{2}, 1, 0, (\overline{1}, 0)::nil] \rightarrow^r_\sigma [\overline{1}, 0, 0, nil] \rightarrow^r_\sigma \overline{1}$. In $\lambda s_e$ the only possible two steps simulation is: $(\lambda \overline{2}) \overline{1} \rightarrow^{\sigma\text{-generation}} 2\sigma^1 \overline{1} \rightarrow^{\beta\text{-destruction}} \overline{1}$. □

As mentioned in the above proof, we prove a stronger result than simple better adequacy of $\lambda s_e$ as in [26]. 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 4.11 and 4.12 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 $(\lambda(\lambda^n \overline{1})) \overline{1}$, where
\[ n \geq 0: (\lambda(\lambda^n \downarrow)) \downarrow \rightarrow_{\sigma-gen} (\lambda^n \downarrow)\sigma^1 \downarrow \rightarrow^n_{\sigma-\lambda-trans} \lambda^n (i \sigma^{n+1} \downarrow) =: t_1 \]
\[ (\lambda(\lambda^n \downarrow)) \downarrow \rightarrow_{\beta_s} [\lambda^n \downarrow, 1, 0, (dbj, 0) :: nil] \rightarrow^n_{r_7} \lambda^n \downarrow, n + 1, n, @n-1 :: \ldots :: @0 :: (j, 0) :: nil] =: t_2. \]

After that the \( \lambda_\text{se} \) complete the simulation in one or two steps by checking arithmetic inequations:

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

But in the \( \lambda_\text{susp} \) we have to destruct the environment list, environment by environment:

\[
t_2 \rightarrow^{i-1}_{r_5} \lambda^n [1, n-i + 2, n, @n-i :: \ldots :: @0 :: (j, 0) :: nil] \rightarrow_{r_3} \lambda^n i, \quad \text{if } i < n + 1 \\
\rightarrow^{n+1}_{r_5} \lambda^n [i - n - 1, 0, n, nil] \rightarrow_{r_2} \lambda^n i - 1, \quad \text{if } i > n + 1 \\
\rightarrow^{i-1}_{r_5} \lambda^n [1, 1, n, (j, 0) :: nil] \rightarrow_{r_4} \lambda^n [j, 0, n, nil] \rightarrow_{r_2} \lambda^n j + n, \quad \text{if } i = n + 1 
\]

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 [34]), since the \( \lambda_\text{se} \) has been proved to answer this question negatively in [18].

5 Relating the Eta rules

[3] established the correspondence between the Eta rules of \( \lambda \sigma \) and \( \lambda_\text{se} \) through the premises \( t[\uparrow] =_\sigma M \) and \( \varphi^2_0 t =_s M \), where \( t \in \Lambda_{dB} \). This correspondence means that the effect of applying the substitution \( [\uparrow] \), in \( \lambda \sigma \), and the upgrading \( \varphi^2_0 \), in \( \lambda_\text{se} \), to a pure \( \lambda \)-term are identical. This implies that these Eta rules are equivalent when applied to a pure \( \lambda \)-term. Hence, it remains to show that the results, in the two calculi, of applying the substitution \( [\uparrow] \) and the upgrading operator \( \varphi^2_0 \) to a \( \lambda \)-term \( t \) are equal (up to the codification of the term in the internal language of the calculus). This is the case \( k = 0 \) of the third item of the following lemma.

**Lemma 5.1 (Eta correspondence of \( \lambda \sigma \) and \( \lambda_\text{se} \) [3])**
(1) Let $n$ be a de Bruijn index. Then, for $k \geq 0$, the $s_e$-nf of $\varphi^2_n$ and the $\sigma$-nf of $n[1,1[1],1[2],\ldots,1[t^{k-1}],t^{k+1}]$ are corresponding de Bruijn indices.

(2) Let $\lambda t$ an abstraction over $\Lambda_{dB}$. Then, for $k \geq 0$,

$$\lambda t[1,1[1],1[2],\ldots,1[t^{k-1}],t^{k+1}] \sigma$$-rewrites to $\lambda(t[1,1[1],1[2],\ldots,1[t^{k}],t^{k+2}]).$

(3) Let $t \in \Lambda_{dB}$ and $t'$ its codification in the language of $\lambda \sigma$, where all de Bruijn indices $n \in \mathbb{N}$ occurring in $t$ are replaced with $1[t^{n-1}]$. Then, for $k \geq 0$, the $\sigma$-nf of $t'[1,1[1],1[2],\ldots,1[t^{k-1}],t^{k+1}]$ corresponds to the $s_e$-nf of $\varphi^2_k t$.

Analogously to the previous lemma, in the next proposition we establish the correspondence between the rules Eta of $\lambda_{susp}$ and $\lambda s_e$; i.e., the correspondence, in the above mentioned sense, between the terms at their premises: $[t,0,1,nil]$ and $\varphi^2_k t$, for $t \in \Lambda_{dB}$. This corresponds to the case $k = 0$ of the following proposition.

**Proposition 5.2 (Eta correspondence of $\lambda_{susp}$ and $\lambda s_e$)** Let $t \in \Lambda_{dB}$. Then, for all $k \geq 0$, the $susp$-nf of the suspended term

$$[t,k,k+1,\@ k :: @ k - 1 :: \ldots :: @ 1 :: nil]$$

corresponds to the $s_e$-nf of $\varphi^2_k t$.

**Proof.** This is done by induction on the structure of $t$.

- **$t = n$.** By Lemma 3.15 we have that for all $k \geq 0$,

$$[n,k,k+1,\@ k :: @ k - 1 :: \ldots :: @ 1 :: nil] \rightarrow \begin{cases} n + 1 & \text{if } n > k \\ n & \text{if } n \leq k \end{cases}$$

This coincides with the result of applying the rule $\varphi$-dest to the term $\varphi^2_n$.

- **$t = (A B)$.** $[(A B),k,k+1,\@ k :: @ k - 1 :: \ldots :: @ 1 :: nil] \rightarrow_{\rho_0}$

$$([A,k,k+1,\@ k :: @ k - 1 :: \ldots :: @ 1 :: nil], [B,k,k+1,\@ k :: \ldots :: @ 1 :: nil]) \mathrel{IH} (\varphi^2_k A \varphi^2_k B).$$

- **$t = (\lambda A)$.** $[(\lambda A),k,k+1,\@ k :: @ k - 1 :: \ldots :: @ 1 :: nil] \rightarrow_{\tau_1}$

$$((\lambda\sigma A,k,k+1,\@ k :: @ k - 1 :: \ldots :: @ 1 :: nil)) \mathrel{IH} (\lambda\varphi^2_{k+1} A).$$

This correspondence is not obvious for open terms. In fact, let $c$ be a constant.

On one side, in $\lambda_{susp}$, we have that $[c,k,k+1,\@ k :: \ldots :: @ 1 :: nil] \rightarrow_{\tau_1} c$. On the other side, $\varphi^2_k c$ is irreducible in $\lambda s_e$. Both terms can, in a certain sense, be considered equivalent since the upgrading operator $\varphi^2_k$ does not modify the constant $c$ and this correspondence could be assumed in other practical contexts such as those of higher order unification via explicit substitutions.

The following notational conventions are useful for the rest of the paper:
Notation 5.3 Let $\xi \in \{\sigma, s_c, \text{susp}\}$, and let $\lambda \xi$ be the corresponding explicit substitution calculus. The generation rules of $\lambda \xi$ (i.e. the Beta, $\sigma$-generation or $\beta_\xi$ rules), will be denoted correspondingly by $\lambda \xi$-gen. Similarly, $\text{Eta}_\xi$ denotes the corresponding Eta rule. $\xi$ denotes the associated substitution calculus, that is given by the rewriting rules of the calculus $\lambda \xi$ except the $\xi$-gen and the $\text{Eta}_\xi$ rules. The congruence generated by the rules of the substitution calculus $\xi$ is denoted by $=_{\xi}$. By $\xi$-nf$(M)$ we denote the $\xi$-normal form of the $\lambda \xi$-term $M$. If $M$ has a $\lambda \xi$-gen redex at the root position then we denote by gen$_{\lambda \xi}(M, \text{root})$ its contractum.

Now, we establish the appropriateness of the three Eta rules of $\lambda \sigma$, $\lambda s_c$ and $\lambda \text{susp}$. By appropriateness of a specific Eta rule we understand that every pure $\lambda$-terms which contains an Eta redex is reduced to the same pure $\lambda$-term by the usual $\eta$-rule as well as by the specific Eta rule.

Lemma 5.4 (Appropriateness of the Eta rules) Let $a \in \Lambda_{dB}$. The following statements are equivalent:

(a) $\lambda(a \ 1) \rightarrow_{\eta} b$
(b) $\lambda(a \ 1) \rightarrow_{\text{Eta}_\xi} b$, where $\xi$ stands for $\sigma$, $s_c$ or $\text{susp}$.

PROOF. Suppose (a) is true. Then by structural induction on the term $a$:

(1) $\lambda \text{susp}$: We will show that $[b, 0, 1, \text{nil}] =_{\text{susp}} a$.
   • $a = \underline{n}$ ($n > 1$): $[\underline{n} - 1, 0, 1, \text{nil}] \rightarrow_2 \underline{n}$.
   • $a = (c \ d)$. $b^+ = (c \ d)$ means that $b$ is obtained from $(c \ d)$ decreasing all its free indices by one. Now note that the effect of normalizing $[b, 0, 1, \text{nil}]$ is to increase all free indices of $b$ by one as shown in the proof of the proposition 3.16.
   • $a = \lambda c$. Suppose $b^+ = \lambda c$ so $b$ is obtained from $\lambda c$ decreasing all free indices in $c$ but $\underline{1}$ by one and conclude considering the same argument of the previous item.

(2) $\lambda s_c$: This is a straightforward from previous item and proposition 5.2.

(3) $\lambda \sigma$: This is a straightforward from previous item and lemma 5.1.

Conversely, we will show that:

(1) In $\lambda \text{susp}$ that $[b, 0, 1, \text{nil}] =_{\text{susp}} b^+$:
   • $[\underline{n} - 1, 0, 1, \text{nil}] \rightarrow_{r_2} \underline{n} = (\underline{n} - 1)^+$.
   • $[(c \ d), 0, 1, \text{nil}] \rightarrow_{r_6} [c, 0, 1, \text{nil}] [d, 0, 1, \text{nil}] \overset{IH}{=} c^+ d^+ = (c \ d)^+$.
   • $[\lambda c, 0, 1, \text{nil}] \rightarrow_{r_{17}} \lambda[c, 1, 2, @1 :: \text{nil}] \overset{IH}{=} \lambda c^+ = (\lambda c)^+$.

(2) In $\lambda s_c$ that $\varphi_0^2(b) = b^+$:
   • $\varphi_0^2(\underline{n} - 1) \rightarrow_{\varphi - \text{dest}} \underline{n} = (\underline{n} - 1)^+$
   • $\varphi_0^2(c \ d) \rightarrow_{\varphi-\text{app}} \varphi_0^2(c) \varphi_0^2(d) \overset{IH}{=} c^+ d^+ = (c \ d)^+$.
   • $\varphi_0^2(\lambda c) \rightarrow_{\varphi-\lambda} \lambda(\varphi_0^2 c) \overset{IH}{=} \lambda(c^+) = (\lambda c)^+$.
(3) In $\lambda \sigma$ that $b[\dagger] = b^+$.
\begin{itemize}
  \item $1[\dagger^{n-2}][\dagger] \rightarrow_{Clos} 1[\dagger^{n-1}] = (n - 1)^+.$
  \item $(c \ d)[\dagger] \rightarrow_{App} c[\dagger] d[\dagger] \equiv c^+ d^+ = (c \ d)^+.$
  \item $(\lambda c)[\dagger] \rightarrow_{Abs} \lambda (c[\dagger^{1^2}]) \equiv \lambda c^{+1} = (\lambda c)^+.$
\end{itemize}

6 Usual Implementations of Eta

In the sequel we use “$\eta$” for $\eta$-reduction, and “Eta” for the Eta-reduction rules of the explicit substitutions calculi. By an “implementation” of the Eta rule of any of the three treated calculi of explicit substitutions we understand an effective computational mechanism for evaluation of the premiss of the conditional rewriting Eta rule, which allows for deciding the occurrence of Eta-redices and their subsequent reduction. In other words, an implementation is an effective mechanism for deciding the one step Eta-reduction relation.

When implementing the one step reduction of these calculi one has to take into account that the given Eta rule and its suggested implementation are not clean in the sense that one application of Eta-reduction can involve applications of other rules of the substitution calculus.

In an explicit substitutions calculus $\lambda \xi$, a clean implementation of the $\eta$-reduction does not apply additional rules of the associated substitution calculus $\xi$ during a one step application of the implemented $\eta$-reduction.

Definition 6.1 (Clean Implementations of $\eta$-reduction) An implementation of $\eta$-reduction, say $\text{ImEta}_\xi$, in an explicit substitution calculus $\lambda \xi$ is said to be clean if for any $\lambda \xi$-term $M$, whenever we obtain $N$ from $M$ by applying this implementation of the $\eta$-reduction, denoted by $M \rightarrow_{\text{ImEta}_\xi} N$, there is no $N'$ such that $M \rightarrow_{\text{Eta}_\xi} N'$ and $N' \rightarrow^{*}_{\xi} N$. An implementation of $\eta$-reduction that is not clean is called unclean.

Lemma 6.2 (The Eta rules are unclean) The implementations of $\eta$-reduction directly from the Eta$_\xi$ rewriting rules of the three treated calculi are unclean.

PROOF. Counterexamples are easy to formulate (e.g. see proof of Lemma 6.4) because the equational premise of all the three rules is given in terms of the corresponding $\xi$ congruence $=_{\xi}: a =_{\sigma} b[\dagger], a =_{s_{\xi}} \varphi_{b}^{2}(b)$ and $a =_{\text{susp}} [b, 0, 1, \text{nil}]$, respectively.

□
6.1 Rule implementation for λσ

We used OCAML, a variation of the ML language, for implementing the rewriting rules of the three treated calculi. The code of this implementation is available at http://www.mat.unb.br/~ayala/TCgroup/. For λσ, consider for example the rule Abs. We have to remark that λσ works with two different entities: terms (TERMS) and substitutions (SUBS), which should be discriminated in any implementation. λσ-terms of the form 1, λM, (M N) and M[S] are respectively represented as One, L(M), A(M,N) and Sb(M,S) and λσ-substitutions of the form id, ↑, M.S and S◦T as Id, Up, Pt(M,S) and Cp(S,T). Applications of the rules are implemented in two steps: the first one of detection of redices and the second one, after selection of a possible redex, of true reduction. Detection of redices for this rule is implemented as in Table 7. Note that the search for redices is divided in the search over terms and substitution entities. Once a redex at position pr of the term exp is detected (and selected) the application of Abs is done by means of the function specified in Table 8. Analogously, the application is divided in parts for terms and substitutions. All other rules are similarly implemented.

Table 7
Detection of redices for Abs of λσ

```ocaml
let rec matchingAbs exp l pos =
  match exp with
  Dummy -> l | One -> l | Vr c -> l |
  A(e1,e2) -> append (matchingAbs e1 l (append pos [1]))
  (matchingAbs e2 [] (append pos [2])) |
  L(e1) -> matchingAbs e1 l (append pos [1]) |
  Sb(L(e1),sb) -> pos::append(matchingAbs e1 l (append pos [1;1]))
  (matchingAbsSb sb [] (append pos [2]))|
  Sb(e1,sb) -> append (matchingAbs e1 l (append pos [1]))
  (matchingAbsSb sb [] (append pos [2]))
and matchingAbsSb subs l pos =
  match subs with
  Up -> l | Id -> l |
  Pt(e1,sb) -> append (matchingAbs e1 l (append pos [1]))
  (matchingAbsSb sb [] (append pos [2])) |
  Cp(s1,s2) -> append (matchingAbsSb s1 l (append pos [1]))
  (matchingAbsSb s2 [] (append pos [2]));
```

6.2 Rule implementation for λσe

The implementation for λσe is simpler since we have to consider a sole entity, that is the one of (lambda) terms. λσe-terms are of the form n, (M N), λM, Mσ1N and φkM and are represented in OCAML respectively as DB n, A(M,N),
let rec absreduction exp pr =  
  match pr with [] -> (match exp with Sb(L(e1),sb) ->  
    L(Sb(e1,Pt(One,Cp(sb,Up)))) | _ -> exp) |  
  1 :: tail -> (match exp with  
    Dummy -> exp | One -> exp | Vr c -> exp |  
    A(e1,e2) -> A((absreduction e1 tail),e2) |  
    L(e1) -> L(absreduction e1 tail) |  
    Sb(e1,s2) -> Sb((absreduction e1 tail),s2))|  
  2 :: tail -> (match exp with  
    Dummy -> exp | One -> exp | Vr c -> exp |  
    L(e1) -> exp |  
    A(e1,e2) -> A(e1,(absreduction e2 tail)) |  
    Sb(e1,s2) -> Sb(e1,(absreductionSb s2 tail)))|  
    _ -> exp  
and absreductionSb subs pr =  
  match pr with [] -> subs |  
  1 :: tail -> (match subs with  
    Id -> subs | Up -> subs |  
    Cp(s1,s2) -> Cp((absreductionSb s1 tail),s2) |  
    Pt(e1,s2) -> Pt((absreduction e1 tail),s2)) |  
  2 :: tail -> (match subs with  
    Id -> subs | Up -> subs |  
    Cp(s1,s2) -> Cp(s1,(absreductionSb s2 tail)) |  
    Pt(e1,s2) -> Pt(e1,(absreductionSb s2 tail))) |  
    _ -> subs;;

L(M), S(i,M,N) and P(k,i,M). Searching for redices of the \(\sigma\)-\(\lambda\)-transition and its application for a selected redex \(pr\) are given in Tables 9 and 10, respectively.

6.3 Rule implementation for \(\lambda_{\text{SUSP}}\)

Expressions in \(\lambda_{\text{SUSP}}\) can be of three different types: (suspended) terms, environments and environment terms. Terms of the form \(C, n, (M N), \lambda M\) and \([t, i, j, e]\) are represented by \(Vr c, DB n, A(M, N), L(M)\) and \(Sp(t, i, j, e)\); environments of the form \(nil, et : e\) and \(\{env1, i, j, env2\}\) by \(Nilen, Con(et,e)\) and \(Ck(env1,i,j,env2)\); and environment terms of the form \(@n, (t, l)\) and \(\langle\langle envt,i,j,env\rangle\rangle\) by \(Ar(n), Paar(t, l)\) and \(LG(envt,i,j,env)\), respectively. The search for redices of the rule \((r_7)\) is given in Table 11 and for its application in a selected position in Table 12. Note that the search for redices and the application of the rule is divided in the search over suspended terms,
Table 9
Detection of redices for $\sigma$-$\lambda$-transition of $\lambda$s_e

let rec matchingSLtransition exp l pos =
match exp with Dummy -> l | DB i -> l | Vr c -> l |
A(e1,e2) -> append(matchingSLtransition e1 l (append pos [1])) |
L(e1) -> (matchingSLtransition e1 l (append pos [1])) |
S(i,L(e1),e2) -> pos :: append |
(matchingSLtransition e1 l (append pos [1;1])) |
(matchingSLtransition e2 [] (append pos [2]))) |
S(i,e1,e2) -> append |
(matchingSLtransition e1 l (append pos [1])) |
(matchingSLtransition e2 [] (append pos [2]))) |
P(j,k,e1) -> (matchingSLtransition e1 l (append pos [1]));;

Table 10
Application of $\sigma$-$\lambda$-transition of $\lambda$s_e

let rec sltransition exp pr =
match pr with [] -> (match exp with |
S(i,L(e1),e2) -> L(S(i+1,e1,e2)) | _ -> exp) |
1 :: tail -> (match exp with |
A(e1,e2) -> A((sltransition e1 tail),e2) | |
L(e1) -> L(sltransition e1 tail) |
S(i,e1,e2) -> S(i,(sltransition e1 tail),e2) |
P(j,k,e1) -> P(j,k,(sltransition e1 tail)) |
_ -> exp ) |
2 :: tail -> (match exp with |
A(e1,e2) -> A(e1,(sltransition e2 tail)) | |
S(i,e1,e2) -> S(i,e1,(sltransition e2 tail)) |
_ -> exp ) | _ -> exp;;

environments and environment terms.

6.4 Implementations by $\xi$-normalization of Eta are unclean

Observe that except for the Eta rule, deciding the applicability of all other rewrite rules of the three calculi (cf. Table 1 for $\lambda\sigma$; 2, 3 and 4 for $\lambda$s_e; 5 and 6 for $\lambda$susp) is straightforward, since these rules are either non conditional rules or their premises are simple arithmetic conditions easy to decide by means of built-in arithmetic mechanisms that are embedded in all modern computational systems.

Nevertheless, the applicability of the Eta rules of the three calculi depends on
Table 11
Detection of redices for $r_7$ of $\lambda_{\text{susp}}$

\begin{verbatim}
let rec matching_r7 exp l pos = match exp with
  Dummy -> l | DB i -> l | Vr c -> l |
  A(e1,e2) -> append (matching_r7 e1 l (append pos [1]))
    (matching_r7 e2 [] (append pos [2])) |
  L(e1) -> (matching_r7 e1 l (append pos [1])) |
  Sp(L(e1),_,_,env) -> pos ++ append
    (matching_r7 e1 l (append pos [1;1]))
    (matchingEnv_r7 env [] (append pos [2])) |
  Sp(e1,_,_,env) -> append
    (matching_r7 e1 l (append pos [1]))
    (matchingEnv_r7 env [] (append pos [2]))
and matchingEnv_r7 env l pos = match env with Nilen -> l |
  Con(envt, env1) -> append
    (matchingEt_r7 envt l (append pos [1]))
    (matchingEnv_r7 env1 [] (append pos [2])) |
  Ck(env1,_,_,env2) -> append
    (matchingEnv_r7 env1 l (append pos [1]))
    (matchingEnv_r7 env2 [] (append pos [2]))
and matchingEt_r7 envt l pos = match envt with Ar i -> l |
  LG(envt1,_,_,env1) -> append
    (matchingEt_r7 envt1 l (append pos [1]))
    (matchingEnv_r7 env1 [] (append pos [2])) |
  Paar(e1,i) -> (matching_r7 e1 l (append pos [1]));;
\end{verbatim}

checking a condition over the congruence of the rewrite system, which can, in
the first instance be implemented following a suggestion by Borovanský in [11]
for $\lambda\sigma$ and used in [2] for $\lambda_{se}$. Note that the $\eta$-reduction $\lambda(M \downarrow) \rightarrow_{\eta} N$
gives a term $N$ resulting from $M$ by decrementing all its free de Bruijn indices
by one. And the suggestion is that this corresponds to the normalization, after
the application at the root position of the generation rule of the considered calculus
of the term $((\lambda M) \downarrow)$ whenever $\downarrow$ does not occur in this normalization. The
implementation of this suggestion is presented for the three calculi in the
following definition.

**Definition 6.3 (ξ-nf implementation of the η-reduction)** For the three
treated calculi, the direct implementation of the rewrite rule: $\lambda(M \downarrow) \rightarrow_{nfEt_\xi} N$
if $N = \xi-nf(gen_\xi(((\lambda M) \downarrow), root))$ and $\downarrow$ does not occur in $N$, is called
the implementation by $\xi$-normalization of the $\eta$-reduction, denoted by $nfEt_\xi$.

This implementation is sound for $\lambda\sigma$ (cf. [11]) as well as for $\lambda_{se}$ (cf. [2]).
However this implementation is unclean because during $\xi$-normalization, rules
of the substitution calculi not strictly involved in $\eta$-reduction can be applied.
For instance, the $\lambda_{se}$-term $\lambda((4\sigma^1 \downarrow) \downarrow) \rightarrow_{nfEt_{\xi se}} 2$, but $\lambda((4\sigma^1 \downarrow) \downarrow) \not\rightarrow_{\eta} 2$. 

32
let rec r7_reduction exp pr =
match pr with
[] -> (match exp with
    Sp(L(e1),i,j,env) -> L(Sp(e1,i+1,j+1,Con(Ar(j),env))) |
    _ -> exp ) |
1 :: tail -> (match exp with
    A(e1,e2) -> A((r7_reduction e1 tail),e2) |
    L(e1) -> L(r7_reduction e1 tail) |
    Sp(e1,i,j,env) -> Sp((r7_reduction e1 tail),i,j,env) |
    _ -> exp) |
2 :: tail -> (match exp with
    A(e1,e2) -> A(e1,(r7_reduction e2 tail)) |
    Sp(e1,i,j,env) -> Sp(e1,i,j,(r7_reductionEnv env tail)) |
    _ -> exp)
and r7_reductionEnv env pr = match pr with
1 :: tail -> (match env with
    Con(envt,env1) -> Con((r7_reductionEt envt tail),env1) |
    Ck(env1,i,j,env2) -> Ck((r7_reductionEnv env1 tail),i,j,env2) |
    _ -> env) |
2 :: tail -> (match env with
    Con(envt,env1) -> Con(envt,(r7_reductionEnv env1 tail)) |
    Ck(env1,i,j,env2) -> Ck(env1,i,j,(r7_reductionEnv env2 tail)) |
    _ -> env)
and r7_reductionEt envt pr = match pr with
1 :: tail -> (match envt with
    Paar(e1,i) -> Paar((r7_reduction e1 tail),i) |
    LG(envt1,i,j,env1) -> LG((r7_reductionEt envt1 tail),i,j,env1) |
    _ -> envt) |
2 :: tail -> (match envt with
    LG(envt1,i,j,env1) -> LG(envt1,i,j,(r7_reductionEnv env1 tail)) |
    _ -> envt);

Of course, \(\lambda((4\sigma^1\downarrow)\downarrow)\to_{\sigma-dest} \lambda(\downarrow)\to_{\eta} 2\) (as well as \(\lambda(\downarrow)\to_{nfEta_{\xi}} 2\)).
Observe here that the Eta rule (table 4) does not correspond to the intended operational semantics of the \(\eta\) rule: \(\lambda(M)\to_{\eta} N\) means that \(M\) and \(N\) are functionally equivalent.

**Lemma 6.4 (\(nfEta_{\xi}\) implementations of the \(\eta\)-reduction are unclean)**
The implementations of the \(\eta\)-reduction by \(\xi\)-normalization for the three treated calculi are unclean.

**PROOF.**

- For the \(\lambda\sigma\), consider the reduction \(\lambda((\downarrow[\uparrow^3][\downarrow[\uparrow.id]]\downarrow)\downarrow)\to_{nfEta_{\xi}} \downarrow[\uparrow] = 2\). But
\( \lambda((\lambda[1\uparrow^2][1\uparrow.id])1) \rightarrow_{\text{Eta}} 1[\uparrow^2][1.id] \rightarrow_\sigma^* 2 \).

- For the \( \lambda s_c \), consider the reduction \( \lambda((4\sigma^1 2)1) \rightarrow_{nf_{\text{Eta}}} 2 \). But \( \varphi^2_0(3\sigma^1 1) = s_c \) \( \beta \) and so \( \lambda((4\sigma^1 2)1) \rightarrow_{\text{Eta}_s} 3\sigma^1 1 \rightarrow_{s_c} 2 \).

- For the \( \lambda_{\text{susp}} \), consider the reduction \( \lambda([4, 1, 0, 2, 0] :: nil) 1 \rightarrow_{nf_{\text{Eta}_{\text{susp}}}} 2 \). But \( [[3, 1, 0, 1, 0] :: nil], 0, 1, nil] = \text{susp} \ [4, 1, 0, 2, 0] :: nil \) and so \( \lambda([4, 1, 0, 2, 0] :: nil) 1 \rightarrow_{\text{Eta}_{\text{susp}}} 3, 1, 0, 1, 0 :: nil \rightarrow_{\text{susp}} 2 \).

In the sequel, we present a cleaner way to implement the \( \text{Eta} \) rules avoiding the application of other rules of the substitution calculi than the ones strictly involved in the \( \eta \)-reduction.

### 7 Clean implementations of Eta

We will adapt the above implementation idea, but will restrict the \( \xi \)-normalization of the term \( \text{gen}_{\lambda \xi}((\lambda M) \diamond) \). The restricted \( \xi \)-normalization, called \( \xi \)-pseudo-normalization, should propagate the dummy symbol between the structure of the term \( M \) without applying extra rules of the substitution calculus.

Essentially the idea for avoiding the application of extra rules of the substitution calculi during the verification of the premise via pseudo-normalization is to apply rules only when occurrences of \( \diamond \) are detected:

\[
i \rightarrow r \text{ if } \diamond \text{ occurs in } i
\]

As for all the other rules previously illustrated, our OCAML implementation divides the application of an \( \text{Eta} \) rule in two parts: detection of redices and reduction. For \( \lambda \sigma \), \( \text{gen}_{\lambda \sigma}((\lambda M) \diamond) = M[\diamond.id] \). The \( \sigma \)-pseudo-nf(M[\diamond.id]) has been implemented as the function \text{sig-norm} in Table 13, where the \text{occursdummy} check, search in linear time the occurrence of \text{Dummy} in \text{exp}. Note that in \text{sig-norm} except for the rules \text{IdL}, \text{IdR} and \text{Clos}, non trivial reductions are possible only if \( \diamond \) occurs. In case these rules had been conditioned like the others, it should be impossible to normalize very simple terms as for instance, \( 1[\uparrow \diamond.id] \) that are necessary for pseudo-normalizations as \( ((\lambda[\uparrow^2]) \diamond) \rightarrow_\beta 1[\uparrow^2 \diamond.id] \rightarrow_{\text{Clos}} 1[\uparrow^2 \circ(\diamond.id)] \rightarrow_{\text{Assoc}} 1[\uparrow \circ(\uparrow \circ(\diamond.id))] \rightarrow_{\text{ShiftCons}} 1[\uparrow \circ \diamond.id] \rightarrow_{\text{IdR}} 1[\uparrow] \). Since our objective is to propagate the dummy symbol between the structure of the normalized term that non restricted application of these rules may be pointed out as a deficiency because extra rules may be applied during the \( \sigma \)-pseudo-normalization.

For \( \lambda s_c \), we have \( \text{gen}_{\lambda s_c}((\lambda M) \diamond) = M\sigma^1 \diamond \). And the \( s_c \)-pseudo-normalization of a \( \lambda s_c \)-term, \text{exp}, is given by the function \text{se-norm} in Table 14. This pseudo-normalization is simpler than the previous one, since we are dealing with a sole entity and additionally the \( \lambda s_c \) rewrite rules preserve, in a certain way,
Table 13
σ-pseudo-normalization

let rec sig-norm exp = match exp with
  Dummy -> Dummy | One -> One | Vr c -> Vr c |
(*App*) Sb(A(e1,e2),sb) -> (if occurdummy1sb(sb) then
  A(Sb(e1,sig-normsb(sb)),Sb(e2,sig-normsb(sb))) else exp)
(*Abs*) Sb(L(e1),sb) -> (if occurdummy1sb(sb) then
  L(Sb(e1,sig-normsb(Pt(One,Cp(sb,Up))))) else exp)
(*Clos*) Sb(Sb(e1,s1),s2) -> Sb(e1,sig-normsb(Cp(s1,s2)))
(*VarCons*) Sb(One,Pt(e1,sb)) -> (if (occurr dummy1(e1) ||
  occurr dummy1sb(sb)) then sig-norm(e1) else exp)
(*Id*) Sb(e1,Id) -> (if occurr dummy1(e1) then sig-norm(e1) else exp)
and sig-normsb subs = match subs with Up -> Up | Id -> Id |
(*SCons*) Pt(Sb(One,s1),Cp(Up,s2)) -> (if ((s1 = s2) &&
  occurr dummy1sb(s1)) then sig-normsb(s1) else subs)
(*ShiftCons*) Cp(Up,Pt(e1,sb)) -> (if (occurr dummy1(e1) ||
  occurr dummy1sb(sb)) then sig-normsb(sb) else subs)
(*IdL*) Cp(Id,sb) -> sig-normsb(sb)
(*IdR*) Cp(sb,Id) -> sig-normsb(sb)
(*Map*) Cp(Pt(e1,s1),s2) -> (if (occurr dummy1(e1) ||
  occurr dummy1sb(s1)) || occurr dummy1sb(s2)) then
  sig-normsb(Pt(sig-norm(Sb(e1,s2)),sig-normsb(Cp(s1,s2))))
  else subs)
(*Assoc*) Cp(Cp(s1,s2),sb3) -> (if (occurr dummy1sb(s1) ||
  occurr dummy1sb(s2)) || occurr dummy1sb(sb3)) then
  sig-normsb(Cp(s1,sig-normsb(Cp(s2,sb3)))) else subs)
_ -> subs;;

the structure of terms: the symbol ◊ remains always as last argument of the
term to be normalized. As a consequence of this regularity, implementation
of the pseudo-normalization is done via unconditional rewrite rules (without
premises “if occurr dummy”). Clearly, this represents an advantage over the
other two calculi.

In λ susp this implementation is very similar to the one of λσ. We have that
gen_{λ susp}((λM ◊) = [M,1,0,(◊,0)::nil]. The function susp-norm in Table
15 implements the susp-pseudo-normalization of a λ susp expression exp.
Observations done for the sig-norm of λσ apply for the susp-norm of λ susp:
except for three rules, one step reduction is decided via the occurr dummy’s
check that runs in linear time on the size of exp. Rules r2 and r3 should be
implemented without any Dummy. As for λσ, this implies that other rules than
those essential for the propagation of the ◊ symbol may be applied during this
pseudo-normalization.

One may think there is a tradeoff because of the inclusion conditionals, but
let rec se-norm exp = match exp with
  Dummy -> Dummy | DB i -> DB i | Vr c -> Vr c | S(i,Vr c,Dummy) -> exp |
  (*si-dest*) S(i,DB j,Dummy) ->
    (if j<i then DB j else (if j>i then (DB (j-1))
    else P(0,i,Dummy))) |
  (*si-app*) S(i,A(e1,e2),Dummy) ->
    A((se-norm (S(i+1,e1,Dummy))),(se-norm (S(i+1,e2,Dummy)))) |
  (*si-lambda*) S(i,L(e1),Dummy) -> L(se-norm (S(i+1,e1,Dummy))) |
  (*si-si*) S(i,S(j,e1,e2),Dummy) -> (if i >= j then
    S(j,(se-norm(S(i+1,e1,Dummy))),(se-norm(S(i-j+1,e2,Dummy))))
    else exp) |
  (*si-phi*) S(i,P(k,n,e),Dummy) ->
    (if i>=k+n then P(k,n,(se-norm(S(i-n+1,e,Dummy))))
    else (if i>k then P(k,n-1,e) else exp)) |
  _ -> exp;;

let rec susp-norm exp = match exp with
  Dummy -> Dummy | DB i -> DB i | Vr c -> Vr c |
  (*r1*) Sp(Dummy,i,j,env) -> Dummy |
  (*r2*) Sp(DB i,0,j,Nilen) -> DB (i+j) |
  (*r3*) Sp(DB 1,i,j,Con(Ar(k),env)) -> DB (j-k) |
  (*r4*) Sp(DB 1,i,j,Con(Paar(e1,k),env)) ->
    (if (occurdummy3 e1) then susp-norm(Sp(e1,0,j-k,Nilen))
    else exp) |
  (*r5*) Sp(DB i,j,k,Con(envt,env)) ->
    (if ((occurdummy3_Et envt) || (occurdummy3_Env env))
      then susp-norm(Sp(DB (i-1),j-1,k,env))else exp) |
  (*r6*) Sp(A(e1,e2),i,j,env) ->
    (if ((occurdummy3 e1)||(occurdummy3 e2)||(occurdummy3_Env env))
      then A(susp-norm(Sp(e1,i,j,env)),susp-norm(Sp(e2,i,j,env)))
    else exp) |
  (*r7*) Sp(L(e1),i,j,env) ->
    (if ((occurdummy3 e1) || (occurdummy3_Env env))
      then L(susp-norm(Sp(e1,i+1,j+1,Con(Ar(j),env))))
    else exp) |
  _ -> exp;;

the verification of occurrences of the Dummy symbol can be performed simultaneously when solving the matching without additional cost.

Definition 7.1 (ξ-pse-nf implementation of the η-reduction) For the calculi λσ, λse and λsusp the previously proposed implementation of the η-reduction, that is formulated as the rewrite rule: \( \lambda(M,1) \rightarrow^{\text{pse-nf Eta}_\xi} N \)
if $N = \xi$-pse-nf\(\text{gen}_{\xi}\((\lambda M) \diamond\), root\)) and $\diamond$ does not occur in $N$, is called the implementation by $\xi$-pseudo normalization of the $\eta$-reduction, denoted by pse-nfEta$_\xi$.

From the argumentations before the previous definition, one can conclude that the implementation of $\eta$-reduction by $\lambda$s-$\xi$-pseudo-normalization is cleaner and more efficient than the corresponding implementations of $\eta$-reduction for $\lambda\sigma$ and $\lambda$susp.

**Lemma 7.2 (pse-nfEta$_\text{susp}$ and pse-nfEta$_\sigma$ implementations of the $\eta$-reduction are unclean)**

The implementations of $\eta$-reduction by susp- and $\sigma$-pseudo normalization are unclean.

**Proof.** Observing the pseudo-normalization rules for these two calculi we can see that, for $\lambda\sigma$, the rules named Clos, IdL and IdR must be implemented without conditional as the others, i.e., these rules do not propagate the $\diamond$ symbol. The justification for this can be found in the third paragraph of Section 7.

An analogous argument is used in the case of $\lambda$susp. \qed

**Lemma 7.3 (pse-nfEta$_s_e$ implementation of the $\eta$-reduction is clean)**

The implementation of $\eta$-reduction by $s_e$-pseudo normalization is clean.

**Proof.** By direct inspection of the pseudo-normalization rules of the $\lambda s_e$-calculus (Table 15). Note that all applied rules just propagate the $\diamond$ symbol. \qed

The following three propositions show the completeness of the implementations of the $\text{Eta}$ rules based on these pseudo-normalizations, denoted by $\text{Eta}_\xi$ for $\xi \in \{\sigma, s_e, \text{susp}\}$, restricted for pure lambda terms.

**Lemma 7.4** Let $M \in \Lambda_{DB}$. The $\sigma$-pseudo-nf of

\[
M[1\,1[\uparrow]\ldots 1[\uparrow^{k-2}].\diamond[\uparrow^{k-1}].\uparrow^{k-1}]
\]

gives a term that preserves all occurrences of terms in $M$ corresponding to variables less than $k$ unchanged, replaces all occurrences corresponding to the the $k^{th}$ variable with $\diamond[\uparrow^{k-1}]$ and decrements by one all occurrences corresponding to variables greater than $k$.

**Proof.** We use the word variable for occurrences of $1[\uparrow^{k-1}]$. By induction on the structure of $M$:

- $M = n$. If $n < k$ then $1[\uparrow^{n-1}][1\,1[\uparrow]\ldots 1[\uparrow^{k-2}].\diamond[\uparrow^{k-1}].\uparrow^{k-1}] \rightarrow_{\text{Clos}}$

37
Let $M \in \Lambda_{ID}$. If $\lambda(M \mathbf{1}) \rightarrow_\eta N$ then $\lambda(M \mathbf{1}) \rightarrow_\text{pse-nfEt\alpha} N$.

PROOF. Here we are interpreting the de Bruijn index $k$ in the language of $\lambda\sigma$ as usual by $\mathbf{1}[\uparrow^{k-1}]$. The proof is by induction on the structure of $M$.

- $M = \mathbf{n}$. If $n \neq 1$ then on the one side, $\lambda(\mathbf{n} \mathbf{1}) \rightarrow_\eta \mathbf{n} - \mathbf{1}$. On the other side we have to that $\mathbf{n} [\diamond.id] = \mathbf{1}[\uparrow^{n-1}][\diamond.id] \sigma$-pseudo-normalizes to $\mathbf{n} - \mathbf{1}$. In fact, $\mathbf{1}[\uparrow^{n-1}][\diamond.id] \rightarrow_\text{Abs} \mathbf{1}[\uparrow^{n-1} \circ (\uparrow \diamond.id)] \rightarrow_\text{Map} \mathbf{1}[\uparrow^{n-2} \circ (\uparrow \diamond.id)] \rightarrow_\text{ShiftClos} \mathbf{1}[\uparrow^{n-2} \circ (\diamond.id)] \rightarrow_\text{IdR} \mathbf{1}[\uparrow^{n-2}] = \mathbf{n} - \mathbf{1}$.

- $M = \mathbf{A B}$. For $A$ and $B$ without occurrences of the free de Bruijn index $\mathbf{1}$, by the condition for the application of the $\eta$-reduction to $(A B)$, we have that $\lambda(A \mathbf{1}) \rightarrow_\eta A'$ and $\lambda(B \mathbf{1}) \rightarrow_\eta B'$, where $A'$ and $B'$ are obtained from $A$ and $B$ by decrementing all the free variables by one. Also, $(A B)[\diamond.id] \rightarrow_\text{App} A[\diamond.id] B[\diamond.id]$. By the induction hypothesis the $\sigma$-pseudo-nf of $A[\diamond.id]$ and $B[\diamond.id]$ corresponds respectively to $A'$ and $B'$.

- $M = (\lambda A)$. $A$ does not own occurrences of terms corresponding to the free de Bruijn index $\mathbf{2}$. Then $\lambda((\lambda A) \mathbf{1}) \rightarrow_\eta \lambda A''$, where $A''$ is obtained from $A$ by decrementing all its free variables except $\mathbf{1}$ by one. Thus applying Lemma 7.4 to the term $M[\diamond.id]$, we obtain the desired result. □

Lemma 7.6 Let $M \in \Lambda_{ID}$. Then the $s_\epsilon$-pseudo-nf of $M \sigma^i \diamond$ gives a term obtained from $M$ by preserving all free de Bruijn indices less than $i$ unchanged, replacing the occurrences of the $i^{th}$ free de Bruijn index with $\varphi_0^i \diamond$ and decrementing all the free occurrences of de Bruijn indices greater than $i$ by one.

PROOF. Induction on the structure of $M$.

- $M = \mathbf{n}$. If $n < i$ then $\mathbf{n} \sigma^i \diamond \rightarrow_\text{sigma-dest} \mathbf{n}$. If $n = i$ then $\mathbf{n} \sigma^i \diamond \rightarrow_\text{sigma-dest} \varphi_0^i \diamond$. If $n > i$ then $\mathbf{n} \sigma^i \diamond \rightarrow_\text{sigma-dest} \varphi_0^{i-1} \diamond$.

...
\[ n > i \text{ then } n \sigma^i \circ \to_{\sigma-\text{dest}} n - 1. \]

- \( M = (A \ B) \). \((A \ B)\sigma^i \circ \to_{\sigma-\text{app}} (A\sigma^i \circ) (B\sigma^i \circ)\). And by induction hypothesis we can conclude.
- \( M = (\lambda A) \). \((\lambda A)\sigma^i \circ \to_{\sigma-\lambda} \lambda A\sigma^{i+1} \circ \). And by induction hypothesis we can conclude. \( \square \)

**Proposition 7.7 (Completeness of pse-nfEta)**

Let \( M \in \Lambda_{dB} \). If \( \lambda(M \ 1) \to_{\eta} N \) then \( \lambda(M \ 1) \to_{pse-nfEtaN} N \).

**Proof.** Induction on the structure of \( M \).

- \( M = n \). If \( n > 1 \) then \( n \sigma^i \circ \to_{\sigma-\text{dest}} n - 1 \).
- \( M = (A \ B) \). For \( A \) and \( B \) without free occurrences of the de Bruijn index 1, we have that \( \lambda(A \ 1) \to_{\eta} A' \) and \( \lambda(B \ 1) \to_{\eta} B' \), where \( A' \) and \( B' \) are obtained from \( A \) and \( B \) by decrementing all their free occurrences of de Bruijn indices by one. Also, \((A \ B)\sigma^i \circ \to_{\sigma-\text{app}} (A\sigma^i \circ) (B\sigma^i \circ)\), and by the induction hypothesis we have that \( (A\sigma^i \circ) \to_{EtaN} A' \) and \( (B\sigma^i \circ) \to_{EtaN} B' \).
- \( M = (\lambda A) \). For \( A \) without free occurrences of the de Bruijn index 2, \( \lambda((\lambda A) \ 1) \to_{\eta} \lambda A'' \), where \( A'' \) is obtained from \( A \) by decrementing all its free de Bruijn indices except 1 by one. Also, \((\lambda A)\sigma^i \circ \to_{\sigma-\lambda} \lambda A\sigma^{2} \circ \). Now by Lemma 7.6 we get the desired result. \( \square \)

**Lemma 7.8** Let \( A \) and \( B \) be well-formed \( \lambda_{\text{susp}} \)-terms and \( k > 0 \). Then the rm-normalization of the well-formed term

\[ [A, k, k - 1, @k - 2 : \ldots : @0 :: (B, l) :: \text{nil}] \]

gives a term by decrementing by one all free de Bruijn indices greater than \( k \) occurring at \( A \), replacing the \( k \)-th free variable of \( A \) with \( B \) (actualized according to the context of the term) and keeps unchanged all other free occurrences of de Bruijn indices.

**Proof.** Similar to the proof of Lemma 3.15. \( \square \)

**Proposition 7.9 (Completeness of pse-nfEtaN)**

Let \( M \in \Lambda_{dB} \). If \( \lambda(M \ 1) \to_{\eta} N \) then \( \lambda(M \ 1) \to_{pse-nfEtaN} N \).

**Proof.** By induction on the structure of \( M \).

- \( M = n \). If \( n > 1 \) then \([n, 1, 0, (\Diamond, 0) :: \text{nil}] \to_{r_5} [n - 1, 0, 0, \text{nil}] \to_{r_5} n - 1 \).
- \( M = (A \ B) \). Similar to Lemma 7.8 using that \([A \ B], 1, 0, (\Diamond, 0) :: \text{nil}] \to_{r_6} [A, 1, 0, (\Diamond, 0) :: \text{nil}] \to_{r_6} [B, 1, 0, (\Diamond, 0) :: \text{nil}] \to_{EtaN} A' \) and \([B, 1, 0, (\Diamond, 0) :: \text{nil}] \to_{EtaN} B' \).
- \( M = (\lambda A) \). For \( A \) without free occurrences of the de Bruijn index 2, \( \lambda((\lambda A) \ 1) \to_{\eta} \lambda A'' \), where \( A'' \) is obtained from \( A \) decrementing by one
all its free de Bruijn indices except 1. Now use $[(\lambda A), 1, 0, (\varnothing, 0)::nil] \rightarrow_{r_7} \lambda[A, 2, 1, @0::(\varnothing, 0)::nil]$ and Lemma 7.8.

\[ \square \]

8 Future Work and Conclusion

[15,3] 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 [37,34] with an adequate Eta rule for $\eta$-reduction and showed that this extended suspension calculus, named $\lambda_{susp}$, enjoys confluence and termination of the associated substitution calculus $susp$ (with Eta).

Additionally, we used the notion of adequacy of [26] for comparing these three calculi when simulating one step $\beta$-reduction. We concluded that $\lambda\sigma$ and $\lambda_\xi$ are mutually non comparable for $\xi \in \{s_e, susp\}$ but that $\lambda s_e$ is more adequate than $\lambda_{susp}$ in simulating one step beta-reduction. 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. Recently Liang and Nadathur pointed out the importance of having the possibility to combine steps of beta-reduction in practical implementations, which resumes to the ability of the calculus to compose substitutions [31,35]. This results in natural applications for $\lambda\sigma$ and the suspension calculus in contrast to the $\lambda s_e$. Consequently, it will be of great importance to study possible adaptations of the $\lambda s_e$ which enable this property. In particular, this would be interesting if the work carried out for $\lambda s_e$ on HOU, can be mapped into the $\lambda t$ [26] which is a calculus à la $\lambda s_e$ but which updates à la $\lambda\sigma$. That is, $\lambda t$ does partial updating, like $\lambda\sigma$ and the suspension calculus, whereas, $\lambda s_e$ does global updating. We leave this for future work.

Moreover, we established the correspondence of these Eta rules of the three calculi. This correspondence means that the operational effects of applying these Eta rules over pure $\lambda$-terms in the three calculi are identical. For the three calculi in question enlarged with adequate eta rules we showed how to implement these eta rules. For the $\lambda s_e$ we build a clean implementation of the eta rule, that is, avoiding the application of other rules of the substitution calculi than the ones strictly involved in the verification of the $\eta$-redices. And we proved that it is not possible to follow the same approach for the $\lambda\sigma$ and $\lambda_{susp}$. We proved that these implementations are complete in the sense that any $\eta$-reduction for dealing with pure $\lambda$-terms in de Bruijn notation can be simulated by these Eta implementations. For the three treated calculi, the main advantage of our clean eta implementation approach is that it is closer than previous implementations to the operational semantics of the usual $\eta$-reduction of the $\lambda$-calculus. Additionally, we have pointed out that for $\lambda_{susp}$
as well as for the $\lambda\sigma$-calculus, in these $\textit{Eta}$ implementations, the application of rules not strictly involved with the $\eta$-reduction is necessary, but that this is not the case for $\lambda_s\varepsilon$. We have also showed that for the former two calculi, conditional rewriting rules whose premises are decided in linear time in the size of the terms in normalization are necessary while for $\lambda_s\varepsilon$ this is done via non conditional rules whose applicability is decided by simple matching of their left-hand sides. Our $\textit{Eta}$ implementation is being incorporated into an ELAN prototype for simply-typed higher order unification via $\lambda_s\varepsilon$.

An immediate work to be done is to study two open questions: whether the $s_c$-calculus has strong normalization (SN) [27], and whether $\lambda_{\text{SUSP}}$ preserves SN. Interesting points arise in this context since: $\lambda_s\varepsilon$ is more adequate in the simulation of one step $\beta$-reduction than $\lambda_{\text{SUSP}}$; $\lambda_s\varepsilon$ does not preserves SN [18]; and the substitution calculus of $\lambda_{\text{SUSP}}$ has SN.

References


