Comparing Calculi of Explicit Substitutions with Eta-reduction

Mauricio Ayala-Rincón\textsuperscript{1,3}

\textit{Departamento de Matemática}
\textit{Universidade de Brasília}
\textit{Brasília D.F., Brasil}

Flávio Leonardo Cavalcanti de Moura\textsuperscript{2,4}

\textit{Departamento de Matemática}
\textit{Universidade de Brasília}
\textit{Brasília D.F., Brasil}

Fairouz Kamareddine\textsuperscript{5}

\textit{Computing and Electrical Engineering}
\textit{Heriot-Watt University}
\textit{Edinburgh, Scotland}

\textbf{Abstract}

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

\textbf{Keywords:} Calculi of Explicit substitutions, lambda-calculi, Eta Reduction.

\textsuperscript{1} Corresponding author supported by CNPq brazilian research council
\textsuperscript{2} Author supported by CAPES brazilian foundation
\textsuperscript{3} Email: ayala@mat.unb.br
\textsuperscript{4} Email: flavio@mat.unb.br
\textsuperscript{5} Email: fairouz@cee.hw.ac.uk

\textcopyright 2002 Published by Elsevier Science B. V.
1 Introduction

Recent years have witnessed an explosion of work on expliciting substitutions [1,7,9,14,15,17,19] and on establishing its usefulness to computation: e.g., to automated deduction and theorem proving [24,25], to proof theory [31], to programming languages [8,20,23,26] and to higher order unification HOU [2,13]. This paper concentrates on three different styles of substitutions:

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

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

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

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

- Comparing the adequacy of their reduction process using the efficient simulation of $\beta$-reduction of [22].

- Extending the suspension calculus with eta-reduction resulting in $\lambda_{\text{susp}}$.

Eta-reduction for $\lambda\sigma$ was used in [13] to deal with HOU and was introduced in [2] for the same purpose in $\lambda s_e$.

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

2 Calculi à la $\lambda\sigma$, $\lambda s_e$ and $\lambda_{\text{susp}}$

2.1 The $\lambda\sigma$-calculus

The $\lambda\sigma$-calculus works on 2-sorted terms: (proper) terms, and substitutions.

Definition 2.1 The $\lambda\sigma$-calculus is defined as the calculus of the rewriting system $\lambda\sigma$ of Table 1 where
### Table 1
The λσ Rewriting System of the λσ-calculus with Eta-rule

| Rule | Description | Result |
|------|-------------|--------|
| (Beta) | \( \lambda a \ b \) \( \rightarrow \) \( a \ [b \cdot id] \) | |
| (Id) | \( a \ [id] \) \( \rightarrow \) \( a \) | |
| (VarCons) | \( \mathsf{1} \ [a \cdot s] \) \( \rightarrow \) \( a \) | |
| (App) | \( (a \ b) \ [s] \) \( \rightarrow \) \( (a \ [s]) \ (b \ [s]) \) | |
| (Abs) | \( (\lambda a) \ [s] \) \( \rightarrow \) \( \lambda a \ [1 \cdot (s \circ \uparrow)] \) | |
| (Clos) | \( (a \ [s]) \ [t] \) \( \rightarrow \) \( a \ [s \circ t] \) | |
| (IdL) | \( id \circ s \) \( \rightarrow \) \( s \) | |
| (IdR) | \( s \circ id \) \( \rightarrow \) \( s \) | |
| (ShiftCons) | \( \uparrow \circ (a \cdot s) \) \( \rightarrow \) \( s \) | |
| (Map) | \( (a \cdot s) \circ t \) \( \rightarrow \) \( a \ [t] \cdot (s \circ t) \) | |
| (Ass) | \( (s \circ t) \circ u \) \( \rightarrow \) \( s \circ (t \circ u) \) | |
| (VarShift) | \( 1 \uparrow \) \( \rightarrow \) \( id \) | |
| (SCons) | \( 1 \ [s] \cdot (\uparrow \circ s) \) \( \rightarrow \) \( s \) | |
| (Eta) | \( \lambda (a \ 1) \) \( \rightarrow \) \( b \) if \( a =_{\sigma} b[1] \) | |

### TERMS
\( a ::= 1 \mid X \mid (a \ a) \mid \lambda a \mid a[s] \) and

### SUBS
\( s ::= id \mid \uparrow \mid a \cdot s \mid s \circ s \)

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 \( 1[^{n-1}] \).

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

The rewriting system \( \lambda \sigma \) is locally confluent [1], CR on substitution-closed terms (i.e., terms without substitution variables) [30] and not CR on open terms (i.e., terms with term and substitution variables) [11]. The possible forms of a \( \lambda \sigma \)-term in \( \lambda \sigma \)-normal form were given in [30] as: 1) \( \lambda a \), where \( a \) is a normal term; 2) \( a_1 \ldots a_p \cdot \uparrow^n \), where \( a_1, \ldots, a_p \) are normal terms and \( a_p \neq \mathsf{n} \) or 3) \( (a \ b_1 \ldots b_n) \), where \( a \) is either \( 1 \), \( 1[^n] \), \( X \) or \( X[s] \) for a substitution variable different from \( id \) in normal form.

In the \( \lambda \)-calculus with names or de Bruijn indices, the rule \( X \{y/a\} = X \), where \( y \) is a variable or a de Bruijn index, 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 \( X \).

2.2 The \( \lambda s_e \)-calculus

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

**Definition 2.2** Terms of the \( \lambda s_e \)-calculus, whose set of rules is presented in Table 2, are given by:

\[
\Lambda s_e := \mathbb{N} | X | \Lambda s_e \Lambda s_e | \lambda \Lambda s_e | \Lambda s_e \sigma^i \Lambda s_e | \varphi^i_k \Lambda s_e, \text{ where } j, i \geq 1, \ k \geq 0
\]

| \( \sigma \)-generation | \( \lambda a b \) \( \rightarrow a \sigma^1 b \) |
| --- | --- |
| \( \sigma \)-\( \lambda \)-transition | \( \lambda a \sigma^i b \) \( \rightarrow \lambda(a \sigma^{i+1} b) \) |
| \( \sigma \)-app-transition | \( (a_1 a_2) \sigma^i b \) \( \rightarrow ((a_1 \sigma^i b)(a_2 \sigma^i b)) \) |
| \( \sigma \)-destruction | \( n \sigma^i b \) \( \rightarrow \begin{cases} n-1 & \text{if } n > i \\ \varphi^i_0 b & \text{if } n = i \\ n & \text{if } n < i \end{cases} \) |
| \( \varphi \)-\( \lambda \)-transition | \( \varphi^i_k (\lambda a) \) \( \rightarrow \lambda(\varphi^i_{k+1} a) \) |
| \( \varphi \)-app-transition | \( \varphi^i_k (a_1 a_2) \) \( \rightarrow ((\varphi^i_k a_1)(\varphi^i_k a_2)) \) |
| \( \varphi \)-destruction | \( \varphi^i_k \mathbb{N} \rightarrow \begin{cases} n+i-1 & \text{if } n > k \\ n & \text{if } n \leq k \end{cases} \) |
| \( \text{Eta} \) | \( \lambda(a \ 1) \rightarrow b \) if \( \ a =_{s_e} \varphi^0_0 b \) |
| \( \sigma \)-\( \varphi \)-transition 1 | \( (\varphi^i_1 a) \sigma^j b \rightarrow \varphi^{i-1}_k a \) if \( k < j < k+i \) |
| \( \sigma \)-\( \varphi \)-transition 2 | \( (\varphi^i_1 a) \sigma^j b \rightarrow \varphi^i_k (a \sigma^{j+1} b) \) if \( k+i \leq j \) |
| \( \varphi \)-\( \sigma \)-transition | \( \varphi^i_k (a \sigma^j b) \rightarrow ((\varphi^i_{k+1} a) \sigma^j (\varphi^i_{k+1-j} b)) \) if \( j \leq k+1 \) |
| \( \varphi \)-\( \varphi \)-transition 1 | \( \varphi^i_k (\varphi^i_1 a) \rightarrow \varphi^{i}_l (\varphi^i_{k+1-j} a) \) if \( l+j \leq k \) |
| \( \varphi \)-\( \varphi \)-transition 2 | \( \varphi^i_k (\varphi^i_1 a) \rightarrow \varphi^{i+i-1}_l a \) if \( l \leq k < l+j \) |

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

The \( \lambda s_e \)-calculus has been proved in [21] to be CR on open terms; to simulate \( \beta \)-reduction: let \( a, b \in \Lambda \), if \( a \to_\beta b \) then \( a \to^*_\lambda s_e b \); to be sound: let \( a, b \in \Lambda \), if \( a \to^*_\lambda s_e b \) then \( a \to^*_\beta b \); and its associated substitution calculus, that is the \( s_e \)-calculus, to be WN and CR. The characterization of the \( s_e \)-normal forms was given in [21,2] as: a term \( a \in \Lambda s_e \) is a \( s_e \)-nf if and only if one of the following holds:

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

\subsection{The Suspension Calculus}

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

- **Suspended Terms**
  
  \[ M, N ::= C | n | \lambda M | (M \ N) | [M, i, j, e_1] \]

- **Environments**
  
  \[ e_1, e_2 ::= \text{nil} | et :: e_1 | \{e_1, i, j, e_2\} \]

- **Environment Terms**
  
  \[ et ::= @i | (M, i) | \{et, i, j, e_1\} \]

where \( C \) denotes any constant and \( i, j \) are non negative integer numbers.

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

The suspension calculus owns a \textit{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 \( \sigma \)-\textit{generation} rules do) and two sets of rules used for handling the suspended terms. The first set, the \( r \) rules, for reading suspensions and the second one, the \( m \) rules, for merging suspensions. These rules are given in the Table 3.

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

**Definition 2.3** \([28]\) The length \( \text{len}(e) \) of an environment \( e \) is given by:

\[
\begin{align*}
\text{len}(\text{nil}) & := 0 \\
\text{len}(et :: e') & := \text{len}(e') + 1 \\
\text{len}(\{e_1, i, j, e_2\}) & := \text{len}(e_1) + (\text{len}(e_2) - i).
\end{align*}
\]

The index \( \text{ind}(et) \) of an environment term \( et \), and the \( l \)-th index \( \text{ind}_l(e) \) of the environment \( e \) and \( l \in \mathbb{N} \), are simultaneously defined by induction on the structure of expressions:
\begin{table}
\centering
\begin{tabular}{|l|}
\hline
\textbf{Rewriting rules of the suspension calculus} \\
\hline
\hline
\textbf{\((\beta_3)\)} & \((\lambda t_1 \ t_2) \rightarrow [t_1, 1, 0, (t_2, 0) :: nil]\) \\
\textbf{\((r_1)\)} & \([c, ol, nl, e] \rightarrow c\), where \(c\) is a constant \\
\textbf{\((r_2)\)} & \([i, 0, nl, nil] \rightarrow i+nl\) \\
\textbf{\((r_3)\)} & \([l, ol, nl, @ l :: e] \rightarrow nl - l\) \\
\textbf{\((r_4)\)} & \([l, ol, nl, (t, l) :: e] \rightarrow [t, 0, (nl - l), nil]\) \\
\textbf{\((r_5)\)} & \([l, ol, nl, et :: e] \rightarrow [l - 1, (ol - 1), nl, e], \text{ for } i > 1\) \\
\textbf{\((r_6)\)} & \([l(t_1 \ t_2), ol, nl, e] \rightarrow ([l(t_1, ol, nl, e)], [l(t_2, ol, nl, e)])\) \\
\textbf{\((r_7)\)} & \([l \ t, ol, nl, e] \rightarrow \lambda [t, (ol + 1), (nl + 1), @ nl :: e]\) \\
\textbf{\((m_1)\)} & \([l[t, ol_1, nl_1, e_1], ol_2, nl_2, e_2] \rightarrow [t, ol', nl', \{e_1, nl_1, ol_2, e_2\}], \text{ where}\) \\
& \((nl' = nl_2 + (nl_1 - ol_2))\) \\
\textbf{\((m_2)\)} & \(\{nil, nl, 0, nil\} \rightarrow nil\) \\
\textbf{\((m_3)\)} & \(\{nil, nl, ol, et :: e\} \rightarrow \{nil, (nl - 1), (ol - 1), e\}, \text{ for } nl, ol \geq 1\) \\
\textbf{\((m_4)\)} & \(\{nil, 0, ol, e\} \rightarrow e\) \\
\textbf{\((m_5)\)} & \(\{et :: e_1, nl, ol, e_2\} \rightarrow \{et, nl, ol, e_2\ :: \{e_1, nl, ol, e_2\}\}\) \\
\textbf{\((m_6)\)} & \(\{et, nl, 0, nil\} \rightarrow et\) \\
\textbf{\((m_7)\)} & \(\{@m, nl, ol, @ l :: e\} \rightarrow @(l + (nl - ol)), \text{ for } nl = m + 1\) \\
\textbf{\((m_8)\)} & \(\{@m, nl, ol, (t, l) :: e\} \rightarrow (t, (l + (nl - ol))), \text{ for } nl = m + 1\) \\
\textbf{\((m_9)\)} & \(\{t, nl, ol, et :: e\} \rightarrow ([t, ol, l', et :: e], m), \text{ where}\) \\
& \((l' = ind(et) \text{ and } m = l' + (nl - ol))\) \\
\textbf{\((m_{10})\)} & \(\{et, nl, ol, et' :: e\} \rightarrow \{et, (nl - 1), (ol - 1), e\}, \text{ for } nl \neq ind(et)\) \\
\hline
\end{tabular}
\caption{}
\end{table}

\[
\text{ind}(@m) = m + 1 \quad \text{ind}(t', m) = m
\]

\[
\text{ind}(\{et', j, k, c\}) = \begin{cases} 
\text{ind}_m(e) + (j + k) & \text{if } len(e) > j + \text{ind}(et') = m \\
\text{ind}(et') & \text{otherwise}
\end{cases}
\]

\[
\text{ind}(\{et :: e'\}) = \text{ind}(et) \text{ and } \text{ind}_{i+1}(et :: e') = \text{ind}(e')
\]

\[
\text{ind}(\{e_1, j, k, e_2\}) = \begin{cases} 
\text{ind}_m(e_2) + (j + k) & \text{if } l < \text{len}(e_1) \text{ and } \text{len}(e_2) > m = j + \text{ind}(e_1) \\
\text{ind}(e_1) & \text{if } l < \text{len}(e_1) \text{ and } \text{len}(e_2) \leq m = j + \text{ind}(e_1) \\
\text{ind}_{l-1+l}(e_2) & \text{if } l \geq l_1 = \text{len}(e_1)
\end{cases}
\]

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

\textbf{Definition 2.4} [[28]] An expression of the suspension calculus is said to be \textit{well-formed} if the following conditions hold over all its subexpressions \(s\):

- if \(s \in [t, ol, nl, e]\) then \(\text{len}(e) = ol\) and \(\text{ind}(e) \leq nl\)
- if \(s \in et :: e\) then \(\text{ind}(e) \leq \text{ind}(et)\)
\[ \text{if } s = \langle et, j, k, e \rangle \text{ then } \text{len}(e) = k \text{ and } \text{ind}(et) \leq j \]

\[ \text{if } s = \{ e_1, j, k, e_2 \} \text{ then } \text{len}(e_2) = k \text{ and } \text{ind}(e_1) \leq j. \]

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

The suspension calculus has been proved to simulate \( \beta \)-reduction and its associated substitution calculus \( \text{susp} \) to be CR (over closed and open terms) and SN [28]. In [26] Nadathur conjectures that the suspension calculus preserves strong normalization too. The following lemma characterizes the \( \triangleright_{\text{rm}} \)-normal forms.

**Lemma 2.5 ([28])** A well-formed expression of the suspension calculus \( x \) is in its \( \triangleright_{\text{rm}} \)-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 \( \triangleright_{\text{rm}} \)-nf term;
3) \( x \) is the environment \( \text{nil} \) or \( et :: e \) for \( et \) and \( e \) resp. an environment term and an environment in \( \triangleright_{\text{rm}} \)-nf.

**3 The suspension calculus enlarged with the \( \eta \)-reduction: the \( \lambda_{\text{susp}} \)-calculus**

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

\[(\text{Eta}) \quad (\lambda (t_1 \downarrow)) \longrightarrow t_2, \quad \text{if } t_1 \triangleright_{\text{rm}} [t_2, 0, 1, \text{nil}]\]

Intuitively \( \text{Eta} \) may be interpreted as: when it is possible to apply the \( \eta \)-reduction to the redex \( \lambda(t_1 \downarrow) \) 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 \( \downarrow \) in \( t_1 \).

Proposition 3.2 proves the correctness of \( \text{Eta} \) according to this interpretation. We follow [10] and [3] for \( \lambda \sigma \) and \( \lambda s_e \) respectively, and implement the \( \text{Eta} \) rule of the \( \lambda_{\text{susp}} \)-calculus by introducing a dummy symbol \( \diamond \), as:

\( \lambda(M \downarrow) \rightarrow_{\text{Eta}} N \text{ if } N = \triangleright_{\text{rm}} \text{-nf}(\llbracket M, 1, 0, (\diamond, 0) :: \text{nil} \rrbracket) \) and \( \diamond \) does not occur in \( N \).

The correctness of this implementation is explained because an \( \eta \)-reduction \( \lambda(M \downarrow) \rightarrow_{\eta} N \) gives us a term \( N \), that is obtained from \( M \) by decrements of all free occurrences of de Bruijn indices, as previously mentioned, and that corresponds exactly to the \( \triangleright_{\text{rm}} \)-normalization of the term \( ((\lambda M) \diamond) \rightarrow_{\beta,} [M, 1, 0, (\diamond, 0) :: \text{nil}] \), whenever \( \diamond \) does not appear in this normalized term.

**Lemma 3.1** Let \( A \) be a well-formed term of the suspension calculus. Then the \( \text{susp} \)-normalization of the term \( \llbracket A, k, k + 1, @k :: @k - 1 :: \ldots :: @1 :: \text{nil} \rrbracket \)
Proposition 3.2 (Soundness of the Eta rule) Every application of the Eta rule of \( \lambda_{\text{susp}} \) to the redex \( \lambda(t_1 \mathbf{1}) \) gives effectively the term \( t_2 \) obtained from \( t_1 \) by decrementing all its de Bruijn free indices by one.

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

Lemma 3.3 (\( \text{susp} \) plus Eta is SN) The rewriting system associated to \( \text{susp} \) and the Eta rule is noetherian.

Lemma 3.4 (Local-confluence of \( \text{susp} \) plus Eta) The rewriting system of the substitution calculus \( \text{susp} \) plus the Eta rule is locally-confluent.

Finally, since the rewriting system associated to \( \text{susp} \) enlarged with the Eta rule is locally-confluent and noetherian, we can apply the Newman diamond lemma for concluding its confluence.

Theorem 3.5 (Confluence of \( \text{susp} \) plus Eta) The calculus \( \text{susp} \) jointly with the Eta rule, is confluent.

4 Comparing the adequacy of the calculi

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

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

Definition 4.1 [[22]] (Adequacy) Let \( \xi_1, \xi_2 \in \{ \sigma, s_e, \text{susp} \} \). The \( \lambda \xi_1 \)-calculus is more adequate (in simulating one step of \( \beta \)-reduction) than the \( \lambda \xi_2 \)-calculus, denoted \( \lambda \xi_1 \prec \lambda \xi_2 \), if

- for every \( \beta \)-reduction \( a \rightarrow_\beta b \) 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 \( \lambda s_e \) are non comparable presented in [22] apply for the incomparability of \( \lambda\sigma \) and \( \lambda s_e \) since \( \lambda s_e \) is an extension of \( \lambda s_e \) for open terms.

Proposition 4.2 The \( \lambda\sigma \)- and the \( \lambda s_e \)-calculi are non comparable.
Lemma 4.3 Every λσ-derivation of \(((\lambda\lambda 2) \mathbf{1})\) to its λσ-nf has length greater than or equal to 6.

Lemma 4.4 Every λ\text{susp}-derivation of \((\lambda\lambda (2 2)) \mathbf{1}^n\) to its λ\text{susp}-nf has length \(4n + 5\).

Lemma 4.5 (\cite{22}) There exists a derivation of \((\lambda\lambda (2 2)) \mathbf{1}^n\) to its λσ-nf whose length is \(n + 9\).

Proposition 4.6 The λσ- and λ\text{susp}-calculi are non comparable.

To prove that λs\text{e} is more adequate than λ\text{susp} we need to estimate the lengths of derivations.

Definition 4.7 Let \(A, B, C \in \Lambda\) and \(k \geq 0\). We define the functions \(M : \Lambda \rightarrow \mathbb{N}\) and \(Q_k : \Lambda \times \Lambda \rightarrow \mathbb{N}\) by:

- \(M(\mathbf{n})=1\)
- \(M(\lambda A) = M(A) + 1\)
- \(M(AB) = M(A) + M(B) + 1\)
- \(Q_k(\mathbf{n}, B) = n + M(B)\) if \(n = k\)
- \(Q_k(\mathbf{n}, B) = n + M(B)\) if \(n < k\)
- \(Q_k(\mathbf{n}, B) = n + M(B)\) if \(n > k\)
- \(Q_k(A, C) = Q_k(A, C)\)
- \(Q_k(A, B) = Q_k(A, B) + 1\)

Lemma 4.8 Let \(A \in \Lambda\). Then all s\text{e}-derivations of \(\varphi^i_k A\) to its s\text{e}-nf have length \(M(A)\).

Lemma 4.9 Let \(A \in \Lambda\). Then all susp-derivations of

\([A, i, i, @i - 1 :: \ldots :: @0 :: nil]\)

to its susp-nf have length greater than or equal to \(M(A)\).

Lemma 4.10 Let \(B \in \Lambda\) and \(i, j \geq 0\). The derivation of the susp-term

\([B, i, j, @j - 1 :: e]\)

to its susp-nf has length greater than or equal to \(M(B)\).

Proposition 4.11 Every susp-derivation of

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

where \(A, B \in \Lambda\) and \(k \geq 0\) to its susp-nf has length greater than or equal to \(Q_k(A, B)\).

Proposition 4.12 Let \(A, B \in \Lambda\) and \(k \geq 1\). s\text{e}-derivations of \(A\sigma^k B\) to its s\text{e}-nf have length \(\leq Q_k(A, B)\).

Theorem 4.13 (λs\text{e} \textless λ\text{susp}) The λs\text{e} is more adequate than the λ\text{susp}-calculus.

As mentioned in the proof above, we prove a stronger result than simple better adequacy of λs\text{e} as in \cite{22}. In fact, we prove that the length of all λs\text{e}-simulations are shorter than the length of any λ\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 $(\lambda(\lambda^n_1))_j$, where $n \geq 0$:

$$(\lambda(\lambda^n_1))_j \rightarrow_{\sigma\text{-gen}} (\lambda^n_1)_{\sigma^j 1} \rightarrow_{\sigma\rightarrow_{\lambda\text{-tra}}} \lambda^n(\lambda^1_1)_{n+1} =: t_1$$

$$(\lambda(\lambda^n_1))_j \rightarrow_{\beta} [\lambda^n_1, 1, 0, (\lambda^1_1, 0) :: \text{nil}] \rightarrow_r \lambda^n_1 n + 1, n @ n - 1 :: :: @ 0 : (\lambda^1_1, 0) :: \text{nil}] =: t_2$$

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

$$t_1 \rightarrow_{\sigma\rightarrow_{\text{dest}}} \begin{cases} 
\lambda^n_1, & \text{if } i < n + 1 \\
\lambda^n_1 - 1, & \text{if } i > n + 1 \\
\lambda^n(\lambda^1_1)_{n+1} \rightarrow_{\phi\rightarrow_{\text{dest}}} \lambda^n_1 + n, & \text{if } i = n + 1
\end{cases}$$

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

$$t_2 \begin{cases} 
\rightarrow_{r_5}^{i-1} \lambda^n_1 [1, n - i + 2, n, @ n - i :: :: @ 0 : (\lambda^1_1, 0) :: \text{nil}] \rightarrow r_3 \lambda^n_1, & \text{if } i < n + 1 \\
\rightarrow_{r_5}^{n-1} \lambda^n_1 [1, n - 1, 0, n, \text{nil}] \rightarrow r_3 \lambda^n_1 - 1, & \text{if } i > n + 1 \\
\rightarrow_{r_5}^{i-1} \lambda^n_1 [1, 1, n, (\lambda^1_1, 0) :: \text{nil}] \rightarrow r_3 \lambda^n_1 [1, 0, n, \text{nil}] \rightarrow r_2 \lambda^n_1 + n, & \text{if } i = n + 1
\end{cases}$$

These simple considerations lead us to believe that the main difference of the two calculus (at least in the simulation of $\beta$-reduction) is given by the manipulation of indices: although $\lambda_{\text{susp}}$ includes all de Bruijn indices, it does not profit from the existence of the built-in arithmetic for indices. These observations may be relevant for the treatment of the open question of preservation or not of strong normalization by $\lambda_{\text{susp}}$ as conjectured positively in [26], since the $\lambda_{se}$ has been proved to answer this question negatively in [15,16].

5 Future Work and Conclusion

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

Additionally, we used the notion of adequacy of [22] for comparing these three calculi when simulating one step of $\beta$-reduction. We concluded that $\lambda\sigma$ and $\xi$ are mutually non comparable for $\xi \in \{se, \text{susp}\}$ but that $\lambda_{se}$ is more adequate than $\lambda_{\text{susp}}$.

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

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

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

[3] M. Ayala-Rincón and F. Kamareddine. On Applying the $\lambda_s$-Style of Unification for Simply-Typed Higher Order Unification in the Pure lambda Calculus. In R.J.G.B. de Queiroz and M. Ayala-Rincón, editors, \textit{Pre-Proceedings Eighth Workshop on Logic, Language, Information and Computation - WoLLIC 2001}, pages 41–54, 2001.

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

[5] F. Baader and T. Nipkow. \textit{Term Rewriting and All That}. Cambridge University Press, 1998.

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

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

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

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

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

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

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

[13] G. Dowek, T. Hardin, and C. Kirchner. Higher-order Unification via Explicit Substitutions. \textit{Information and Computation}, 157(1/2):183–235, 2000.
[14] M. C. F. Ferreira, D. Kesner, and L. Puel. $\lambda$-Calculi with Explicit Substitutions and Composition which Preserve $\beta$-Strong Normalisation. In *Algebraic and Logic Programming, ALP 96, LNCS* 1139, 284–298. Springer, 1996.

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

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

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

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

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

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

[21] F. Kamareddine and A. Ríos. Extending a $\lambda$-calculus with Explicit Substitution which Preserves Strong Normalisation into a Confluent Calculus on Open Terms. *Journal of Functional Programming*, 7:395–420, 1997.

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

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

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

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

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

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

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

[29] R. Nederpelt, J. H. Geuvers and R. de Vrijer. *Selected Papers on Automath*. North-Holland, Amsterdam, 1994.
[30] A. Ríos. *Contribution à l'étude des λ-calculs avec substitutions explicites*. PhD thesis, Université de Paris 7, 1993.

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