Strategies for Simply-Typed Higher Order Unification via \(\lambda s_e\)-Style of Explicit Substitution

Mauricio Ayala-Rincón\(^1\) and Fairouz Kamareddine\(^2\)

1 Departamento de Matemática, Universidade de Brasília, 70910-900 Brasília D.F., Brasil
ayala@mat.unb.br, cee.hw.ac.uk
2 Department of Computing and Electrical Engineering, Heriot-Watt University, Riccarton, Edinburgh EH14 4AS, Scotland fairouz@cee.hw.ac.uk

Abstract. An effective strategy for implementing higher order unification (HOU) based on the \(\lambda s_e\)-style of explicit substitution is proposed. The strategy is based on a \(\lambda s\)-unification method recently developed by the authors. A pre-cooking translation for applying the \(\lambda s_e\)-style of unification to HOU in the pure \(\lambda\)-calculus is presented. Correctness and completeness of the proposed strategy and of the pre-cooking translation are shown and their applicability to HOU in the pure \(\lambda\)-calculus is illustrated.

1 Introduction

In [DHK00], a higher order unification (HOU) method was based on the \(\lambda \sigma\)-style of explicit substitution [ACCL91]. In [ARK00], HOU was studied in the \(\lambda s_e\)-style of explicit substitution [KR97]. It is claimed in [ARK00] that \(\lambda s_e\)-unification has the advantages of enabling quicker detection of redices and of having a clearer semantics. In this paper, we set out to provide an effective strategy for implementing \(\lambda s_e\)-unification and a pre-cooking translation for applying it to HOU in the \(\lambda\)-calculus. It should be stressed that \(\lambda \sigma\) and \(\lambda s_e\) are two different styles of explicit substitution which are not isomorphic. This implies that reworking the HOU method in \(\lambda s_e\) is not a translation of work already done in \(\lambda \sigma\). Many rules and proofs of the \(\lambda s_e\)-HOU differ from those of the \(\lambda \sigma\)-HOU. We outline some of these differences throughout the article.

In Section 2, we introduce the necessary notions, the relevance of explicit substitution in HOU and the \(\lambda s_e\)- and \(\lambda \sigma\)-calculi. In Section 3, we review our \(\lambda s_e\)-style based unification method (cf. [ARK00]). In Sections 4 and 5, we discuss our unification strategy and its applicability for HOU in the pure \(\lambda\)-calculus. Then we conclude and discuss future work in Section 6.

2 Background

We assume familiarity with the notion of term algebra \(\mathcal{T} (\mathcal{F}, \lambda \mathcal{X})\) built on a (countable) set of variables \(\lambda \mathcal{X}\) and a set of operators \(\mathcal{F}\). Variables in \(\lambda \mathcal{X}\) are denoted by upper case last letters of the Roman alphabet \(X, Y, \ldots\). For a term \(t \in \mathcal{T} (\mathcal{F}, \lambda \mathcal{X})\), \(\text{var} (t)\) denotes the set of variables occurring in \(t\). We assume familiarity with the \(\lambda\)-calculus as in [Bar84] and with the basic notions and notation of rewriting theory as in [BN98].

For a reduction relation \(\rightarrow\) over a set \(A\), \(A \rightarrow \)\(R\), we denote with \(\rightarrow^*\) the reflexive and transitive closure of \(\rightarrow\). The subscript \(R\) is usually omitted. When \(a \rightarrow^* b\) we say that there exists a derivation from \(a\) to \(b\). Syntactical identity is denoted by \(a = b\).

A valuation is a mapping from \(\lambda \mathcal{X}\) to \(\mathcal{T} (\mathcal{F}, \lambda \mathcal{X})\). The homeomorphic extension of a valuation, \(\theta\), from its domain \(\lambda \mathcal{X}\) to the domain \(\mathcal{T} (\mathcal{F}, \lambda \mathcal{X})\) is called the grafting of \(\theta\). This notion is usually called first order substitution and corresponds to simple substitution without renaming. As usual, valuations and their corresponding grafting valuations are denoted by the same Greek letter. The domain of a grafting \(\theta\) is defined by \(\text{Dom}(\theta) = \{X \mid X \theta \neq X, X \in \lambda \mathcal{X}\}\) and its range by \(\text{Ran}(\theta) = \bigcup_{X \in \text{Dom}(\theta)} \text{Var}(X \theta)\). The set of variables involved in \(\theta\) is \(\text{var}(\theta) = \text{Dom}(\theta) \cup \text{Ran}(\theta)\). A valuation and its corresponding grafting \(\theta\) are explicitly denoted by \(\theta = \{X / X \theta \mid X \in \text{Dom}(\theta)\}\). When necessary, explicit representations of graftings are differentiated from substitutions by a “\(g\)” subscript: \(\{X / X \theta \mid X \in \text{Dom}(\theta)\}_g\).

\(^*\) Partially supported by CAPES (BEX0384/99-2) Brazilian Foundation. Work carried out during one year visit at the ULTRA Group, CEE, Heriot-Watt University, Edinburgh, Scotland, and is partly supported by EPSRC grant numbers GR/L36963 and GR/L15685.
Needed properties of the $\lambda\sigma$- and $\lambda s_e$-calculus, their typed versions and normal form characterizations are briefly included.

### 2.1 The $\lambda$-calculus in de Bruijn notation

Let $\mathcal{V}$ be a (countable) set of variables (different from the ones in $\mathcal{X}$) 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 $a ::= x \mid (a \ a) \mid \lambda x.a$. Terms of the forms $\lambda x.a$ and $(a \ b)$ are called abstractions and applications, respectively. As it is well-known, first order substitution or grafting leads to problems in the $\lambda$-calculus. For example, applying the first order substitution $\{u/x\}$ to $\lambda x.(u \ x)\$ results in $\lambda x.(x \ x)$ which is wrong. Therefore, the $\lambda$-calculus with names uses variable renaming via $\alpha$-conversion so that $(\lambda x.(u \ x))\{u/x\}$, by renaming $x$ (say as $y$), results in the correct term $\lambda y.(x \ y)$. Taking care of appropriate $\alpha$-conversions, $\beta$ and $\eta$-reduction rules are defined in $\Lambda(\mathcal{X})$ respectively by $(\lambda x.a \ b) \to a[b/x]$, and $(\lambda x.a \ x) \to a$, if $x \not\in \mathcal{Fvar}(a)$, where $\mathcal{Fvar}(a)$ denotes the set of free variables occurring at $a$.

Unification in $\Lambda(\mathcal{V})$ differs from the first order notion, because bound variables in $\Lambda(\mathcal{V})$ are untouched by unification substitutions. Unification variables in the $\lambda$-calculus are free variables. Thus free variables occurring at terms of a unification problem can be partitioned into true unification variables and constants, that cannot be bound by the unifiers.

To differentiate between unification and constant variables, one could consider unification variables as meta-variables in a set $\mathcal{X}$. Thus, $\lambda$-calculus should be defined as the term algebra, $\Lambda(\mathcal{V}, \mathcal{X})$, over the set of operators $\{\lambda x. \mid x \in \mathcal{V}\} \cup \{\underbrace{\ldots}_{\mathcal{X}}\} \cup \mathcal{V}$ and the set of variables $\mathcal{X}$. In this setting, a notion of substitution could be adapted for meta-variables preserving the semantics of both $\beta$ and $\eta$-reduction. But the most appropriate notation for our purposes is the ones of de Bruijn indices [NgdV94] where bound variables are related to their corresponding abstractors by their relative height. For instance, $\lambda x. (\lambda z \ (x \ z))$ is translated into $\lambda \lambda (2 \ 1) (1 \ 4)$. Indices for free variables are appropriately selected to avoid relating them with abstractors.

The set $\Lambda_{DB}(\mathcal{X})$ of $\lambda$-terms in de Bruijn notation is defined inductively as $a ::= n \mid X \mid (a \ a) \mid \lambda a$ where $X \in \mathcal{X}$ and $n \in \mathbb{N} \ \backslash \ {0}$.

**Definition 21.** Let $a \in \Lambda_{DB}(\mathcal{X}), \ i \in \mathbb{N}$. The $i$-lift of $a$, $a^{+i}$, is defined as:

- $a) \ X^{+i} = X$, for $X \in \mathcal{X}$
- $b) \ (a_1 \ a_2)^{+i} = (a_1^{+i} \ a_2^{+i})$
- $c) \ (\lambda a)^{+i} = \lambda a_{1}^{+(i+1)}$
- $d) \ n^{+i} = \begin{cases} n + 1, & \text{if} \ n > i \\ n, & \text{if} \ n \leq i \end{cases}$

The lift of a term $a$, that is needed to define substitution, is its 0-lift, denoted briefly by $a^{+}$. We will denote by $a^{+(k)}$, the $i$ compositions of $k$-lift.

**Definition 22.** The application of the substitution with $b$ of $n \in \mathbb{N} \ \backslash \ {0}$ on a term $a$ in $\Lambda_{DB}(\mathcal{X})$, denoted $\{n/b\}a$, is defined inductively as:

1. $\{n/b\}X = X$, for $X \in \mathcal{X}$
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 23.** Let $\theta = \{X_1/a_1, \ldots, X_n/a_n\}$ be a valuation from the set of meta-variables $\mathcal{X}$ to $\Lambda_{DB}(\mathcal{X})$. The corresponding substitution, also denoted by $\theta$, is defined inductively as follows:

- $a) \ \theta(m) = m$ for $m \in \mathbb{N}$
- $b) \ \theta(X) = X\theta$, for $X \in \mathcal{X}$
- $c) \ \theta(a_1 \ a_2) = (\theta(a_1) \ \theta(a_2))$
- $d) \ \theta\lambda a_1 = \lambda \theta^{+}(a_1)$

where $\theta^{+}$ denotes the substitution corresponding to the valuation $\theta^{+} = \{X_1^{+}/a_1^{+}, \ldots, X_n^{+}/a_n^{+}\}$.

In $\Lambda_{DB}(\mathcal{X})$, the left side of the $\eta$-reduction rule is written as $\lambda (a' \ 1)$, where $a'$ stands for the corresponding translation of $a$ into the language of $\Lambda_{DB}(\mathcal{X})$. The condition “$x \not\in \mathcal{Fvar}(a')$” means, in $\Lambda_{DB}(\mathcal{X})$, that there are neither occurrences in $a'$ of the index 1 at height zero nor of the index 2 at height one etc. This means, in general, that there exists a term $b$ such that $b^{+} = a$. Thus $\beta$-reduction is defined as $(\lambda a \ b) \to (\{1/b\}a)$ and $\eta$-reduction as $(\lambda a \ 1) \to b$ if $\exists b \ b^{+} = a$. 


2.2 The $\lambda\sigma$-calculus

**Definition 24.** The $\lambda\sigma$-calculus is defined as the calculus of the rewriting system $\lambda\sigma$ presented in Table 1 where terms $a ::= 1 \mid X \mid (a \, b) \mid \lambda a \, a[s]$ and subs $s ::= id \mid \uparrow \mid a.s \mid s \circ s$.

<table>
<thead>
<tr>
<th>Table 1. $\lambda\sigma$ Rewriting System of the $\lambda\sigma$-calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\text{Beta})$ $(\lambda a , b) \rightarrow a ,[b \cdot id]$</td>
</tr>
<tr>
<td>$(\text{VarCons})$ $1[a \cdot s] \rightarrow a$</td>
</tr>
<tr>
<td>$(\text{Abs})$ $(\lambda a ,[s] \rightarrow \lambda \alpha ,[1 \cdot (s \circ \uparrow)]$</td>
</tr>
<tr>
<td>$(\text{IdL})$ $id \circ s \rightarrow s$</td>
</tr>
<tr>
<td>$(\text{ShiftCons})$ $\uparrow \circ (a \cdot s) \rightarrow s$</td>
</tr>
<tr>
<td>$(\text{Ass})$ $(s \circ t) \circ u \rightarrow s \circ (t \circ u)$</td>
</tr>
<tr>
<td>$(\text{SCons})$ $1[s] \circ \uparrow \circ s \rightarrow s$</td>
</tr>
</tbody>
</table>

The equational theory associated to $\lambda\sigma$ defines a congruence denoted by $\equiv_{\lambda\sigma}$. The corresponding congruence obtained by dropping the Beta and Eta rules is denoted by $\equiv_{\sigma}$.

The rewriting system $\lambda\sigma$ satisfies the following properties: it is locally confluent [ACCL91], confluent on substitution-closed terms (i.e., terms without substitution variables) [Rio93] and not confluent on open terms (i.e., terms with term and substitution variables) [CHL96].

**Proposition 25 ([Rio93]).** Any $\lambda\sigma$-term in $\lambda\sigma$-normal form is of one of the following forms: a) $\lambda a \; b$) $(a \, b_1 \ldots b_n)$, where $a$ is either 1, 1[\uparrow^n], X or $X[s]$ being $s$ a substitution term different from id in normal form, or c) $a_1 \ldots a_n \,[\uparrow^n]$, where $a_1, \ldots, a_p$ are normal terms and $a_p \neq \text{n}$.

In $\lambda(X)$ and $\Lambda_{db}(X)$, the rule $X \{y/t\} = X$, where $y$ is an element of $\mathcal{V}$ or a de Bruijn index, respectively, is necessary because there is no way to suspend the substitution $\{y/t\}$ 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 $X$. Observe that the condition $a =_{\sigma} b[\uparrow]$ of the Eta rule is stronger than the condition $a = b^+$ as $X = X^+$, but there exists no term $b$ such that $X = b[\uparrow]$. 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 substitution of variables in $X$ by $X'$-grafting and explicit lifting. Consequently a notion of $X'$-substitution in $\lambda\sigma$-calculus is unnecessary.

2.3 The $\lambda s_e$-calculus

The $\lambda_s$-calculus avoids introducing two different sets of entities and insists on remaining close to the syntax of the $\lambda$-calculus. Next to $\lambda$ and application, the $\lambda_s$-calculus introduces substitution ($\sigma$) and updating ($\varphi$) operators. In the $\lambda s_e$-calculus, we let $a, b, c, \text{etc.}$ range over the sets of terms $\Lambda s$. A term containing neither substitution nor updating operators is called a pure term.

**Definition 26. ($\lambda s_e$-calculus).** The rules $\lambda s_e$ of the $\lambda s_e$-calculus are given in Table 2 and the terms are defined by $\Lambda s_{op} ::= X \mid \mathbb{N} \mid \Lambda s_{op} \Lambda s_{op} \mid \Lambda s_{op} \sigma^j \Lambda s_{op} \mid \varphi^i \Lambda s_{op}$, for $j \geq 1, \ k \geq 0$. The $\lambda s_e$-calculus is the reduction system $(\Lambda s_{op}, \rightarrow_{\lambda s_e})$ where $\rightarrow_{\lambda s_e}$ is the least compatible reduction on $\Lambda s_{op}$ generated by the set of rules $\lambda s_e$. The calculus of substitutions associated with the $\lambda s_e$-calculus is the rewriting system generated by the set of rules $s_e = \lambda s_e - \{\sigma\text{-generation, Eta}\}$ and we call it the $s_e$-calculus.

The equational theory associated with $\lambda s_e$ defines a congruence denoted by $\equiv_{\lambda s_e}$. The congruence obtained by dropping the $\sigma\text{-generation}$ and Eta rules is denoted by $\equiv_{s_e}$. When we restrict the reduction to these rules, we will use expressions such as $s_e\text{-reduction}, s_e\text{-normal form}$, etc, with the obvious meaning.

In order to clarify differences between the $\lambda\sigma$-calculus and the $\lambda s_e$-calculus, we show the correspondence between their Eta rules; i.e., the correspondence between both conditions $b[\uparrow] = a$ and $\varphi^j b = a$. Remember that in the $\lambda\sigma$-calculus we only use the de Bruijn index 1 and that the other indices are codified as 1[\uparrow^n].
Table 2. Rewriting System of the $\lambda s_e$-calculus with $\eta$-rule

\[
\begin{align*}
(\sigma \text{-generation}) & \quad (a \cdot b) \rightarrow a \cdot b \\
(\sigma \text{-\lambda-transition}) & \quad (\lambda.a) \sigma b \rightarrow \lambda.(a \sigma^{i+1} b) \\
(\sigma \text{-app-transition}) & \quad \sigma(a_1 \cdot a_2) \rightarrow ((a_1 \sigma b) \cdot (a_2 \sigma b)) \\
(\sigma \text{-destruction}) & \quad n \sigma b \rightarrow \\
& \quad \begin{cases} n - 1 & \text{if } n > i \\ 0 & \text{if } n = 0 \\ n & \text{if } n < i \end{cases} \\
(\varphi \text{-\lambda-transition}) & \quad \varphi_k^i (\lambda.a) \rightarrow \lambda. (\varphi_k^{i+1} a) \\
(\varphi \text{-app-transition}) & \quad \varphi_k^i (a_1 \cdot a_2) \rightarrow (\varphi_k^i a_1 \cdot (\varphi_k^{i+1} a_2)) \\
(\varphi \text{-destruction}) & \quad \varphi_k^i n \rightarrow \\
& \quad \begin{cases} n + 1 & \text{if } n > k \\ n & \text{if } n \leq k \end{cases} \\
(\text{Eta}) & \quad \lambda.(a \ 1) \rightarrow b \\
& \quad \text{if } a =^{n} \varphi_0 b \\
(\sigma \text{-\varphi-transition 1}) & \quad (a \cdot b) \sigma^i c \rightarrow (a \cdot \sigma^{i+1} c) \sigma^i (b \sigma^{i-1} c) \text{ if } i \leq j \\
(\sigma \text{-\varphi-transition 2}) & \quad (\varphi_k^i a) \sigma^i b \rightarrow \varphi_k^i (a \cdot \sigma^{i+1} b) \text{ if } k + i \leq j \\
(\sigma \text{-\varphi-transition 3}) & \quad \varphi_k^i (a) \cdot b \rightarrow (\varphi_k^{i+1} a) \sigma^i (\varphi_k^{i+1-j} b) \text{ if } j \leq k + 1 \\
(\varphi \text{-\varphi-transition 1}) & \quad \varphi_k^i (\varphi_k^j a) \rightarrow \varphi_k^l (\varphi_k^{l+1} a) \text{ if } l + j \leq k \\
(\varphi \text{-\varphi-transition 2}) & \quad \varphi_k^i (\varphi_k^j a) \rightarrow \varphi_k^l (\varphi_k^{l+1} a) \text{ if } l \leq k < l + j 
\end{align*}
\]

Example 27: Consider the term $\lambda.((\lambda.1 \cdot 3) \ 1)$ in $\Lambda_{AB}(X)$. Observe that the Eta rule applies, since $\varphi_0^0 b = \varphi_0^0 (\lambda.(1 \cdot 1)) \rightarrow (\varphi_0^0 \varphi_0^0 \lambda.(1 \cdot 2)) \rightarrow (\varphi_0^1 \lambda.\varphi_1^1 (1 \cdot 2)) \rightarrow (\varphi_0^1 \lambda.\varphi_1^1 (1 \cdot 2)) \rightarrow (\varphi_0^0 \lambda.(1 \cdot 1) = a$.

Analogously, in the $\lambda\sigma$-calculus we have: $(\lambda.(1 \cdot 1)[\upsilon]) = ((1 \cdot 1)[\upsilon]) \rightarrow (1 \cdot 1)[\upsilon] \text{ if } i \leq j.

The correspondence between both $\text{Eta}$ rules is the case $k = 0$ of the following lemma.

Lemma 28 ([ARK00]). Let $a \in \Lambda_{AB}$ and $a'$ its corresponding codification in the language of the $\lambda$-calculus, where all indices $n \in N$ occurring at $a$ are replaced with $1[\upsilon^n]$.

Then, for all $k \geq 0$, the $\sigma$-normal form of $a'[1[\upsilon] \ldots 1[\upsilon^k]]$ is the corresponding codification of the $s$-normal form of $\varphi_k^0 a$.

The previous lemma can be straightforwardly extended for terms $a \in \Lambda_{AB}(X)$. In fact, observe that for a meta-variable $X \in X$ at a position $i \in O(a)$, the corresponding subterms of the $\sigma$- and $s$-normal forms of $a[\upsilon]$ and $\varphi_k^0 a$ are of the form $X[1[\upsilon] \ldots 1[\upsilon^k]]$ and $\varphi_k^0 X$, respectively, supposing that the height of the occurrence of $X$ at position $i$ is $k$.

Similarly to the $\lambda\sigma$-calculus we can describe operators of the $\lambda s_e$-calculus over the signature of a first order sorted term algebra $\mathcal{T}_{\lambda s_e}(X)$ built on $X$, the set of variables of sort $\text{TERM}$ and its subsort $\text{NatTERM}$. The set of variables of sort $\text{TERM}$ in a term $a \in \mathcal{T}_{\lambda s_e}(X)$ is denoted by $\text{Var}(a)$.

Theorem 29 ([KR97]). a) The $s_e$-calculus is weakly normalizing and confluent. b) The $\lambda s_e$-calculus simulates $\beta$-reduction. c) The $\lambda s_e$-calculus is confluent on open terms.

As corollary of the characterization of the $s_e$-normal forms in [KR97] (Theorem 8) we obtain a characterization of $\lambda s_e$-normal forms.

Corollary 20 ($\lambda s_e$-normal forms). $a \in \Lambda_{op}$ is a $\lambda s_e$-normal form iff

1. $a \in X \cup \mathbb{N}$;
2. $a = (b \ c)$, where $b, c$ are $\lambda s_e$-normal forms and $b$ is not an abstraction of the form $\lambda.a$;
3. $a = \lambda b$, where $b$ is a $\lambda s_e$-normal form excluding applications of the form $(c \ 1)$ such that there exists $d$ with $\varphi^d_i = s_1 \ c$;
4. $a = \varphi^j i c$, where $c$ is a $\lambda s_e$-normal form and $b$ is an $\lambda s_e$-normal form of one of the following forms:
   a) $X$, b) $da^i c$, with $j < i$ or c) $\varphi^j i d$, with $j \leq k$;
5. $a = \varphi^j i b$, where $b$ is a $\lambda s_e$-normal form of one of the following forms:
   a) $X$, b) $ca^j d$, with $j > k + 1$ or c) $\varphi^j i c$, with $k < l$.

2.4 Typed $\lambda$-calculi

For the sake of clarity we include only the essential notation of typed $\lambda\sigma$- and $\lambda s_e$-calculi. Properties can be found in detail in [ARK00].

We recall that an environment $\Gamma$ in de Bruijn setting is simply a list of types and, in the case of the $\lambda\sigma$-calculus, substitutions receive environments as types. For all the systems we will consider, we take:

$\text{TYPES } A ::= A \mid A \rightarrow B$ and $\text{ENVS } \Gamma ::= \text{nil} \mid A. \Gamma$. The rewrite rules of the corresponding typed calculi are exactly the same except that rules involving abstractions are now typed. Reduction in the typed $\lambda\sigma$- and $\lambda s_e$-calculus is defined by adding to the rules in $\lambda\sigma$ and in $\lambda s_e$ the necessary typing information. Thus, for the typed $\lambda\sigma$-calculus we have the typed rules (Beta), (Abs) and (Eta) respectively as follows:

$$(\lambda(x). a \ b) \rightarrow a [b \cdot \text{id}] \quad (\lambda(x). a)[s] \rightarrow \lambda(x). (a \circ (s \circ \uparrow)) \quad \lambda(x). (a \ 1) \rightarrow b \text{ if } a =_\sigma b$$

and for the typed $\lambda s_e$-calculus:

$$(\sigma\text{-generation}) \quad (\lambda(x). a \ b) \rightarrow a \sigma^1 b \\ (\sigma\text{-transformation}) \quad (\lambda(x). a) \sigma^1 b \rightarrow \lambda(x). (a \sigma^{i+1} b)$$

We denote typability in $\Lambda_{\text{LB}}(X)$, the $\lambda\sigma$- and $\lambda s_e$-calculus by $\vdash_{\Lambda_{\text{LB}}(X)}$, $\vdash_{\lambda\sigma}$ and $\vdash_{\lambda s_e}$ respectively.

Characterization of $\eta$-long normal forms in the typed $\lambda\sigma$- and $\lambda s_e$-calculus is necessary to simplify the set of rules of the unification algorithms. Essentially, the use of $\eta$-long normal forms guarantees that meta-variables of a functional type $A \rightarrow B$ are instantiated with typed terms of the form $\lambda(x). a$.

**Definition 21** ($\eta$-long normal form in $\lambda\sigma$). Let $a$ be a $\lambda\sigma$-normal form term of type $A_1 \rightarrow \ldots \rightarrow A_n \rightarrow B$ in the environment $\Gamma$. The **$\eta$-long normal form** ($\eta$-nf) of $a$, written $a^\eta$, is defined by:

1. if $a = \lambda x. b$ then $a^\eta = \lambda x. b$;
2. if $a = (x \ b_1 \ldots b_p)$ then $a^\eta = \lambda x. \lambda A_1 \ldots \lambda A_n (k + n c_1 \ldots c_p \ n' \ldots 1')$ where $c_i$ is the $\eta$-nf of the normal form of $b_i[^m]$;
3. if $a = (X[s] \ b_1 \ldots b_p)$ then $a^\eta = \lambda A_1 \ldots \lambda A_n (X[s'] \ c_1 \ldots c_p \ n' \ldots 1')$ where $c_i$ is the $\eta$-nf of $b_i[^m]$ and if $s = d_1 \ldots d_i \ t^k$ then $s' = e_1 \ldots e_i \ t^{k+n}$ where $e_i$ is the $\eta$-nf of $d_i[^m]$.

**Definition 22** ($\eta$-long normal form in $\lambda s_e$). Let $a$ be a $\lambda s_e$-normal form term of type $A_1 \rightarrow \ldots \rightarrow A_n \rightarrow B$ in the environment $\Gamma$. The **$\eta$-long normal form** ($\eta$-nf) of $a$, written $a^\eta$, is defined by:

1. if $a = \lambda x. b$ then $a^\eta = \lambda x. b$;
2. if $a = (b_1 \ldots b_p)$ then $a^\eta = \lambda A_1 \ldots \lambda A_n (c_1 \ldots c_p \ n' \ldots 1')$, where $c_i$ is the $\eta$-nf of the normal form of $\varphi_0^{i+1} b_i$;
3. if $a = \sigma^i c$ then $a^\eta = \lambda A_1 \ldots \lambda A_n (d^i c + e' \ n' \ldots 1')$, where $d', e'$ are the $\eta$-nfs of the normal forms of $\varphi_0^{i+1} b$ and $\varphi_0^{i+1} c$, respectively;
4. if $a = \varphi_0^i b$ then $a^\eta = \lambda A_1 \ldots \lambda A_n (\varphi_0^i c' \ n' \ldots 1')$, where $c'$ is the $\eta$-nf of the normal form of $\varphi_0^{i+1} b$.

The set of unification rules of both HOU methods are constructed by combining the different types of $\eta$-nfs enumerated in Definitions 21 and 22 obtaining different types of equational problems. For the HOU setting based on the $\lambda s_e$-style an additional characterization of $\lambda s_e$-terms whose main operators are either $\sigma$ or $\varphi$ will be useful in order to combine directly $\eta$-nfs of type 2 (See subsection 2.5) with the ones of type 3 and 4. This simplifies the comparison of both HOU approaches.

**Definition 23** (Long normal form (lnf)). Let $a$ be either a $\lambda\sigma$-term or a $\lambda s_e$-term. The **long normal form** of $a$ is defined as the $\eta$-nf of its $\beta\eta$-normal form.

In both typed $\lambda\sigma$- and $\lambda s_e$-calculus, we have that two terms are $\beta\eta$-equivalent if they have the same lnf.
2.5 \(\lambda s_e\)-normal forms

We present a characterization of \(\lambda s_e\)-normal terms whose main operators are either \(\sigma\) or \(\varphi\) (i.e. of type 3. and 4. in Corollary 210). This is essential in order to simplify our presentation of the unification rules and of the flex-flex equations.

Observe that left arguments of the \(\sigma\) operator or arguments of the \(\varphi\) operator at \(\lambda s_e\)-normal terms are neither applications, nor abstractions, nor de Bruijn indices. For instance, \(\varphi_1(a\ b) \rightarrow (\varphi_1\ a\ \varphi_1\ b)\), \((a\ b)\sigma c \rightarrow ((a\sigma c)\ b\sigma c)\). Hence, the sole possibility is to have as a left argument a meta-variable. Thus one has to consider terms with alternating sequences of operators \(\varphi\) and \(\sigma\) whose left innermost argument is a meta-variable; for instance, \(((\varphi_1^2\ ((\varphi_1^i\ X)\sigma i^2 a))\sigma^i b)\sigma^k c\).

**Definition 214.** Let \(t\) be a \(\lambda s_e\)-normal term whose root operator is either \(\sigma\) or \(\varphi\) and let \(X\) be its left innermost meta-variable. Denote by \(\psi_{i_1}^j\) the operator at the \(k^{th}\) position following the sequence of operators \(\varphi\) and \(\sigma\), considering only left arguments of \(\sigma\) operators, in the innermost leftmost ordering. Additionally, if \(\psi_{i_k}^j\) corresponds to an operator \(\varphi\) then \(j = 0\) and \(i_k\) denotes its superscript, respectively and if \(\psi_{i_k}^j\) corresponds to an operator \(\sigma\) then \(j = 0\) and \(i_k\) denotes its subscript. Let \(k_{x}\) denote the corresponding right argument of the \(k^{th}\) operator if \(\psi_{i_k}^j\ = \sigma^i x\) and the empty argument if \(\psi_{i_k}^j\ = \varphi_{i_k}^j\). The skeleton of \(t\) written \(sk(t)\) is \(\psi_{i_1}^j \ldots \psi_{i_p}^j (X, a_1, \ldots, a_p)\).

**Example 215** Consider a \(\lambda s_e\)-normal term \(t\) of the form \(((\varphi_1^2\ ((\varphi_1^i\ X)\sigma i^2 a))\sigma^i b)\sigma^k c\). Then the skeleton of \(t\), \(sk(t)\), is \(\psi_{i_1}^j \psi_{i_2}^j \psi_{i_3}^j \psi_{i_4}^j X, a_1, \ldots, a_p\).

**Lemma 216.** Let \(t\) be a \(\lambda s_e\)-normal term whose root operator is either \(\sigma\) or \(\varphi\) and let the skeleton of \(t\), \(sk(t)\) is \(\psi_{i_1}^j \ldots \psi_{i_p}^j (X, a_1, \ldots, a_p)\). Successive subscripts \(i_k\) and \(i_{k+1}\) satisfy the following conditions:

1. \(i_k > i_{k+1}\) if \(\psi_{i_k}\) and \(\psi_{i_{k+1}}\) are both \(\sigma\) operators or both \(\varphi\) operators;
2. \(i_k > i_{k+1}\) if \(\psi_{i_k}\) and \(\psi_{i_{k+1}}\) are \(\varphi\) and \(\sigma\) operators, respectively;
3. \(i_k > i_{k+1} + 1\) if \(\psi_{i_k}\) and \(\psi_{i_{k+1}}\) are \(\sigma\) and \(\varphi\) operators, respectively.

**Proof.** By simple analysis of the arithmetic constraints at the \(\lambda s_e\) rewrite rules.

3 Unification in the \(\lambda s_e\)-calculus

In this section we briefly present unification in the \(\lambda s_e\)-style of explicit substitution, as is given in [ARK00]. Normal form characterization of \(\lambda s_e\)-terms jointly with WN and CR properties are the essential requirements to develop a unification method for the \(\lambda s_e\)-calculus, which can be applied for HOU in the \(\lambda\)-calculus.

Let \(T(F, X)\) be a term algebra over a set of function symbols \(F\) and a countable set of variables \(X\) and let \(A\) be an \(F\)-algebra. A **unification problem** over \(T(F, X)\) is a first order formula without universal quantifier or negation, whose atoms are of the form \(F\), \(T\) or \(s = t\). Unification problems are written as disjunctions of existentially quantified conjunctions of atomic unifiable equations: \(D = \bigvee_{j \in J} \exists w. \bigwedge_{i \in I} s_i = \gamma^A t_i\). When \(|J| = 1\), the unification problem is called a **unification system**. Variables in the set \(w\) of a unification system \(\exists w. \bigwedge_{i \in I} s_i = \gamma^A t_i\) are bound and all other variables are free. \(T\) and \(F\) stand for the empty conjunction and disjunction, respectively. The empty disjunction corresponds to an unsatisfiable problem.

A unifier of a unification system \(\exists w. \bigwedge_{i \in I} s_i = \gamma^A t_i\) is a grafting such that \(A \models \exists w. \bigwedge_{i \in I} s_i \sigma_{\gamma^A t_i} = t_i\sigma_{\gamma^A t_i}\) where \(\sigma_{\gamma^A t_i}\) denotes the restriction of the grafting \(\sigma\) to the domain \(X \setminus w\). A unifier of \(\bigvee_{j \in J} \exists w. \bigwedge_{i \in I} s_i = \gamma^A t_i\) is a grafting \(\sigma\) that unifies at least one of the unification systems. The set of unifiers of a unification problem, \(D\), or system, \(P\), is denoted by \(\mathcal{U}_A(D)\) or \(\mathcal{U}_A(P)\), respectively.

**Definition 31.** A \(\lambda s_e\)-unification problem \(P\) is a unification problem in the algebra \(T_{\lambda s_e}(X)\) modulo the equations theory of \(\lambda s_e\). An equation of such a problem is denoted \(\alpha = \beta\), where \(\alpha\) and \(\beta\) are \(\lambda s_e\)-terms of the same sort. An equation is called trivial when it is of the form \(\alpha = \beta\).

We present a set of rewrite rule schemata used to simplify unification problems. The objective of applying the rules is to obtain a description of the set of unifiers. Basic decomposition rules for unification should be applied modulo the usual boolean simplification rules as given in [DHK00].
Table 3. $\lambda_s$-unification rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(Dec)$</td>
<td>$P \land \lambda_s a \equiv_{\lambda_s} \lambda_s b \rightarrow P \land a \equiv_{\lambda_s} b$</td>
</tr>
<tr>
<td>$(Dec-App)$</td>
<td>$P \land (n_1 \ldots a_1 \ldots a_p) \equiv_{\lambda_s} (n_2 \ldots b_1 \ldots b_p) \rightarrow P \land a_1 \ldots a_p \equiv_{\lambda_s} b_1 \ldots b_p$</td>
</tr>
<tr>
<td>$(App-Fail)$</td>
<td>$P \land (n_1 \ldots a_1 \ldots a_p) \equiv_{\lambda_s} (m_1 \ldots b_1 \ldots b_p) \rightarrow \varnothing$</td>
</tr>
<tr>
<td>$(Dec-\sigma)$</td>
<td>$P \land a \sigma^b \equiv_{\lambda_s} \sigma^a d \rightarrow P \land a \equiv_{\lambda_s} c \land b \equiv_{\lambda_s} d$</td>
</tr>
<tr>
<td>$(\sigma-Fail)$</td>
<td>$P \land a \sigma^b \equiv_{\lambda_s} \sigma^a d \rightarrow \varnothing$</td>
</tr>
<tr>
<td>$(Dec-\varphi)$</td>
<td>$P \land \varphi_1 a \equiv_{\lambda_s} \varphi_1 b \rightarrow P \land a \equiv_{\lambda_s} b$</td>
</tr>
<tr>
<td>$(\varphi-Fail)$</td>
<td>$P \land \varphi_1 a \equiv_{\lambda_s} \varphi_1 b \rightarrow \varnothing$</td>
</tr>
<tr>
<td>$(Exp)$</td>
<td>$P \rightarrow \exists (Y : \Gamma \vdash B). P \land X =_{\lambda_s} \lambda_s Y$</td>
</tr>
<tr>
<td>$(Exp-App)$</td>
<td>$P \land \psi_1 \ldots \psi_n (X, a_1, \ldots, a_p) \equiv_{\lambda_s} (m_1 \ldots b_1 \ldots b_p) \land \bigwedge_{i \in \Gamma_1 \cup \Gamma_2} \exists H_1, \ldots, H_n, X =_{\lambda_s} \langle \lambda_s H_1 \ldots H_n \rangle$</td>
</tr>
<tr>
<td>$(Replace)$</td>
<td>$P \land X \equiv_{\lambda_s} a \rightarrow {X/a} P \land X \equiv_{\lambda_s} a$</td>
</tr>
<tr>
<td>$(Normalization)$</td>
<td>$P \land a \equiv_{\lambda_s} b \rightarrow P \land a' \equiv_{\lambda_s} b'$</td>
</tr>
</tbody>
</table>

Definition 32. The set in Table 3 defines the $\lambda_s$-unification rules for the typed $\lambda_s$-unification problems.

Since $\lambda_s$ is CR and WN, the search can be restricted to $\eta$-long normal solutions that are graftings binding functional variables into $\eta$-long normal terms of the form $\lambda a$ and atomic variables into $\eta$-long normal terms of the form $(k \ b_1 \ldots b_p)$ or $\sigma^a b$ or $\varphi_1 a$, where in the first case $k$ can be omitted and $p$ be zero. The use of the $\eta$-rule is important to reduce the number of cases (or unification rules) to be considered when defining the unification algorithm, but as for the $\lambda$-calculus, the $\eta$-rule can be dropped [DHK00]. As for the $\lambda$-style of unification, Normalize and Dec-$\lambda$ use the fact that $\lambda_s$ is CR and WN to normalize equations of the form $\lambda a \equiv_{\lambda_s} \lambda b$ into $a' \equiv_{\lambda_s} b'$ and the rule Replace propagates the grafting $\{X/a\}$ corresponding to equations $X \equiv_{\lambda_s} a$. Exp-$\lambda$ generates the grafting $\{X/Y\}$ for a variable $X$ of type $A \rightarrow B$, where $Y$ is a new variable of type $B$.

Equations of the form $(n \ a_1 \ldots a_p) \equiv_{\lambda_s} (m \ b_1 \ldots b_p)$ are transformed by the rules Dec-$App$ and App-Fail into the empty disjunction when $n \neq m$, as it has no solution, or into the conjunction $\land_{i=1 \ldots p} a_i \equiv_{\lambda_s} b_i$, when $n = m$. Remember that by terms of the form $(n \ a_1 \ldots a_p)$ we also mean those where $n$ is omitted or $p = 0$. Analogously, the rules Dec-$\sigma$ and Dec-$\varphi$ decompose equations with leading operators $\sigma$ and $\varphi$. But, the corresponding rules $\sigma-Fail$ and $\varphi-Fail$ should omit flex-flex equations as the Example 33 shows.

Example 33 Let $\lambda(\lambda(X \ 2 \ 1) \ Y) \equiv_{\lambda_s} (\lambda(Z \ 1) \ U)$ be a unification problem, where $X, Y, Z$ and $U$ are meta-variables of the same atomic type, say $A$.

Then $(\lambda(X \ 2 \ 1) \ Y) \rightarrow^* (\lambda(X^2 \ Y) \sigma^1 (\varphi_0^1 Y) \ 
\varphi_0^1 Y)$ and $(\lambda(Z \ 1) \ U) \rightarrow^* (Z^\varphi_1 U \ \varphi_0^1 U)$. Thus applying the rule Normalize to the original equation we obtain $(\lambda(X^2 \ Y) \sigma^1 (\varphi_0^1 Y) \ 
\varphi_0^1 Y) \equiv_{\lambda_s} (Z^\varphi_1 U \ \varphi_0^1 U)$ which can be decomposed into $(\lambda(X^2 \ Y) \sigma^1 (\varphi_0^1 Y) \equiv_{\lambda_s} Z \ \varphi_0^1 U)$ and $\varphi_0^1 Y \equiv_{\lambda_s} \varphi_0^1 U$ and subsequently into $(\lambda(X^2 \ Y) \equiv_{\lambda_s} Z \ \varphi_0^1 Y \equiv_{\lambda_s} U \ \wedge \ Y \equiv_{\lambda_s} U)$.

Since $\forall n \in \mathbb{N}, \varphi_0^1 n \rightarrow n$, the equation $\varphi_0^1 Y \equiv_{\lambda_s} U$ always has solutions, and solutions of the last two equations are graftings of the form $\{Y/V, U/V\}$. Additionally, observe that the first equation has a variety of solutions: take $\{X/n\}$; thus if $n > 2$, $\{Z/n-1\}$ else if $n = 2$, $\{Z/\varphi_0^1 Y\}$ else $\{Z/1\}$.
Analogously, by normalization and decomposition with the $\lambda_\sigma$-unification rules we have:
\[
(\lambda (X \ 2) \ 1) \ Y =^2_{\lambda_\sigma} (\lambda (Z \ 1) \ U) \rightarrow_{\text{Normalize}} (X[Y;Y.id] \ Y) =^2_{\lambda_\sigma} (Z[U.id] \ U),
\]
which can be decomposed into $X[Y;Y.id] =^2_{\lambda_\sigma} Z[U.id] \wedge Y =^2_{\lambda_\sigma} U$. A further step of replacement gives the corresponding \textit{flex-flex} equation of the $\lambda_\sigma$-calculus $X[Y;Y.id] =^2_{\lambda_\sigma} Z[Y.id]$.

In $\lambda_\sigma$-HOU, the rule $\textit{Exp-App}$ advances towards solutions to equations of the form $X[a_1 \ldots a_p, x^n] =^2_{\lambda_\sigma} (m b_1 \ldots b_j)$ where $X$ is an unsolved variable of an atomic type. The $\lambda_{s_\sigma}$-unification rule $\textit{Exp-App}$ has the analogous role for $\lambda_{s_\sigma}$-unification problems. Use of $\lambda_{s_\sigma}$-normal forms in $\textit{Exp-App}$ is not essential. This is done with the sole objective of simplifying the case analysis presented in the definition of the rule and its completeness proof. In fact, this can be dropped and subsequently incorporated as an efficient unification strategy, where after applying $\textit{Exp-App}$, $\lambda_{s_\sigma}$-unification problems are normalized.

**Example 34** From the unification problem $\lambda (\lambda (Y \ 1) \lambda (X \ 1)) =^2_{\lambda_\sigma} \lambda (\lambda (V \ 1) \ W)$ we reach the equations:
\[
(Y[\lambda (X \ 1);id] \lambda (X \ 1)) =^2_{\lambda_\sigma} \lambda (\lambda (V;id) \ W) \text{ and } (Y[V;\lambda (X \ 1) \lambda (\varphi_1 \ 1)]) =^2_{\lambda_{s_\sigma}} \lambda (V;\lambda (X \ 1)).
\]
After applying the corresponding $\textit{Exp-App}$ rules, with $V =^2_{\lambda_\sigma} (V_1 \ V_2)$ and $V =^2_{\lambda_{s_\sigma}} (V_1 \ V_2)$, additional equations appear:
\[
\lambda (X \ 1) =^2_{\lambda_\sigma} V_2[\lambda (X \ 1);id] \text{ and } \lambda (\varphi_1 \ 1) =^2_{\lambda_{s_\sigma}} V_2[\lambda (X \ 1);id] \lambda (\lambda (V;id) \ W).
\]
Solutions result by selecting the case $V_2 =^2_{\lambda_\sigma} V_1$ or correspondingly $V_2 =^2_{\lambda_{s_\sigma}} V_1$.

**Definition 35.** A unification system $P$ is a $\lambda_{s_\sigma}$-\textit{solvable form} if it is a conjunction of non trivial equations of the following forms:

- (Solved) $X =^2_{\lambda_{s_\sigma}} \ a$, where the variable $X$ does not occur anywhere else in $P$ and $a$ is in $\textit{inf}$. Such an equation and variable are said to be \textit{solved} in $P$.
- (Flex-Flex) \textit{non solved equations between long normal terms whose root operator is $\sigma$ or $\varphi$} which can be represented as equations between their skeleton:
\[
\psi_{i_p} \ldots \psi_{i_1} (X, a_1, \ldots, a_p) =^2_{\lambda_{s_\sigma}} \psi_{b_1} \ldots \psi_{b_j} (Y, b_1, \ldots, b_j).
\]

**Remark 36** Consider a $\lambda_{s_\sigma}$-normal term $t$ whose root operator is either $\sigma$ or $\varphi$ and with skeleton of the form $sk(t) = \psi_{i_p} \ldots \psi_{i_1} (X, a_1, \ldots, a_p)$. Then by binding $X$ with $n, n > i_1$, one obtains the normal form $t \rightarrow^* n - p + \sum_{i=1}^p j_k$. This is a direct consequence of lemma 216.

The rest of this section lists relevant properties of the $\lambda_{s_\sigma}$-unification rules. For proofs, see [ARK00].

**Lemma 37.** Any $\lambda_{s_\sigma}$-solvable form has $\lambda_{s_\sigma}$-unifiers.

**Lemma 38 (Well-typedness).** Deduction by the $\lambda_{s_\sigma}$-unification rules of a well typed equation gives rise only to well typed equations, $\textit{TT}$ and $\textit{EF}$.

**Lemma 39 (Equivalence of solvedness and normalization).** Solved problems are normalized for the $\lambda_{s_\sigma}$-unification rules and, conversely, if a system is a conjunction of equations that cannot be reduced by the $\lambda_{s_\sigma}$-unification rules then it is solved.

**Definition 40.** Let $P$ and $P'$ be $\lambda_{s_\sigma}$-unification problems, let “rule” denote the name of a $\lambda_{s_\sigma}$-unification rule and “$\rightarrow^\text{rule}$” its corresponding deduction relation over unification problems. By \textit{correctness} of rule we understand: $P \rightarrow^{\text{rule}} P'$ implies $U_{\lambda_{s_\sigma}}(P') \subseteq U_{\lambda_{s_\sigma}}(P)$. By \textit{completeness} of rule we understand: $P \rightarrow^{\text{rule}} P'$ implies $U_{\lambda_{s_\sigma}}(P) \subseteq U_{\lambda_{s_\sigma}}(P')$.

**Theorem 311 (Correctness and Completeness).** The $\lambda_{s_\sigma}$-unification rules are correct and complete.
4 A unification strategy

\(\lambda_\alpha\)-unification rules should be applied following some strategy that avoids non termination of the unification process. Observe, in particular, that the rule \(Exp\-\lambda\) can be applied infinitely many times on one variable of a system if no replacement is done. Analogously to [DHK00] we define a unification strategy that after each application of either \(Exp\-\lambda\) or \(Exp\-app\) applies the rule \(Replace\). Rules should, of course, be applied in a fair manner, which means that in one disjunction of systems, none of the constitutive systems is left forever without applying unification rules on it.

Our so called unification replace strategy, consists of a fair application of the \(\lambda_\alpha\)-unification rules as presented in Table 4 (A; B means A before B, and A or B means choose either A or B).

Successive applications of \((Exp\-\lambda; \text{Replace})\) and \((Exp\-App; \text{Replace})\) are denoted by \(Exp\-\lambda R\) and \(Exp\-AppR\) respectively.

A unification problem \(P\), is divided into the non solved, say \(Q\), and solved, say \(R\), equations. We use the notation \(P = (Q, R)\). Completeness of the unification replace strategy is proved by showing that all the above groups of rules decrease a complexity measure based on the grafting \(\theta\) resulting from the unification algorithm. For a solved system \(R\) consisting of only solved equations, \(\text{Subst}(R)\) denotes the canonical grafting associated to \(R\). For example if \(R = (X = \lambda \alpha a)\) then \(\text{Subst}(R) = \{X/a\}\).

Take in the rest of this section a \(\lambda_\alpha\)-normalized grafting solution \(\theta\) of a unification problem \(P\).

**Definition 41.** For a system of equations \(P = (Q, R)\) and a \(\lambda_\alpha\)-normalized grafting \(\theta\), which is a solution of \(P\), we define the UniStrat transformations \(\langle Q, R, \theta \rangle \to r \langle Q', R', \theta' \rangle\), where \(r\) is a group of rules of the unification replace strategy, as follows:

1. \(\langle Q, R, \theta \rangle \to \text{Normalize} \langle Q', R', \theta' \rangle\), where \(Q'\) and \(R'\) are the normalized forms of \(Q\) and \(R\) as defined in the \(\lambda_\alpha\)-unification.
2. \(\langle Q, R, \theta \rangle \to \text{Dec-\lambda} \langle Q', R', \theta' \rangle\), where \(Q' = Q \wedge a = \lambda_\alpha b\) and \(R' = R\) when \(a = \lambda_\alpha b\) is not solved with respect to \(Q \wedge R\) or \(Q' = Q\) and \(R' = R \wedge a = \lambda_\alpha b\) when \(a = \lambda_\alpha b\) is solved.
3. \(\langle Q, R, \theta \rangle \to \text{Dec-\-App} \langle Q', R', \theta' \rangle\), where \(Q'\) consists in \(Q\) and the unsolved equations (with respect to \(Q \wedge R\) in \(\bigwedge_{i=1 \ldots p} a_i = \lambda_\alpha b_i\)) and \(R'\) consists in \(R\) and the solved equations in \(a = \lambda_\alpha b\).
4. \(\langle Q, R, \theta \rangle \to \text{Dec-\varphi} \langle Q', R', \theta' \rangle\), where \(Q'\) consists in \(Q\) and the unsolved equations (with respect to \(Q \wedge R\) in \(a = \lambda_\alpha b\)) and \(R' = R\) else \(Q' = Q\) and \(R' = R \wedge a = \lambda_\alpha b\).
5. \(\langle Q, R, \theta \rangle \to \text{Exp-\lambda R} \langle \langle X/\lambda A \theta \rangle, Q, R \wedge X = \lambda_\alpha Y \theta \rangle \in \{X/\lambda A a\} \cup \{Y/a\}\), when \(Exp\-\lambda\) applies on \(Q \wedge R\).
6. \(\langle Q, R, \theta \rangle \to \text{Exp-AppR} \langle \{X/(\tau H_1 \ldots H_k)\}, (X, a_1, \ldots, a_p) = \lambda_\alpha (m b_1 \ldots b_q)\), \(R, \theta \rangle \to \text{Exp-AppR} \langle \{X/(\tau H_1 \ldots H_k)\}, (X, a_1, \ldots, a_p) = \lambda_\alpha (m b_1 \ldots b_q)\), \(R \wedge X = \lambda_\alpha (\tau H_1 \ldots H_k) \theta \cup \{X/(\tau c_1 \ldots c_k) \cup \{H_i/c_i\}\})\),

for one of the \(r \in R_p \cup R_t\), when \(Exp\-App\) applies on \(Q \wedge R\).

**Lemma 42 (UniStrat is well defined).** The transformations \(Exp\-\lambda R\) and \(Exp\-AppR\) are well defined.

**Proof.** We show that the transformations applied on the grafting part \(\theta\) of \(\langle Q, R, \theta \rangle\) make sense.

First, observe that since \(Exp\-\lambda\) preserves the solutions, \(\theta\) is also a \(\lambda_\alpha\)-solution of the equation \(X = \lambda_\alpha Y\). This together with the assumption that \(\theta\) is \(\lambda_\alpha\)-normalized, implies that the instantiation of \(X\) by \(\theta\) is of the form \(\{X/\lambda A a\}\). Hence the transformation \(Exp\-\lambda R\) is well defined.

Second, since \(Exp\-App\) preserves the solutions, \(\theta\) is \(\lambda_\alpha\)-solution of the equation \(X = \lambda_\alpha (\tau H_1 \ldots H_k)\). Thus \(\theta(X) = (\tau c_1 \ldots c_k)\), for some \(c_i, i = 1, \ldots, k\). Consequently, the transformation \(Exp\-AppR\) is well defined too.
Lemma 43 (Finiteness of UniStrat). For a system \( \langle Q, R \rangle \) having for solution a \( \lambda_{s} \)-normalized grafting \( \theta \), there is no infinite derivation issued from \( \langle Q, R, \theta \rangle \), using the UniStrat transformations.

Proof. We define the size of a grafting as the sum of the size of the terms in its range: \( |\theta| = \sum_{t \in \text{Ran}(\theta)} |t| \).

First, we prove that there are no infinite sequences of transformation applications involving the transformations Normalize, Replace, Dec-\( \lambda \), Dec-App, Dec-\( \sigma \) and Dec-\( \varphi \). We define a complexity measure, \( \tau \), of a system \( P = \langle Q, R \rangle \) by: \( \tau(P) = \langle |\text{var}(Q)|, \{ \text{max}(\text{max}(\text{var}(Q))[k]), |k| \} \rangle \), where \( k \) is the length of the shortest \( \lambda_{s} \)-normalized derivation of \( a \). Complexities are compared lexicographically using the ordering on naturals for the first component and the multiset ordering for the second component itself ordered by the lexicographic ordering on naturals (for ground notions on multiset ordering see [BN98]).

Now observe that for each possible application of these transformations on a system \( P \) its complexity decreases. Normalize may decrease the number of variables but always decreases the size of one of the \( k \). Replace decreases the number of unsolved variables. Dec-\( \lambda \), Dec-App, Dec-\( \sigma \) and Dec-\( \varphi \) never increase \( k \) (since the normalization derivation of a subterm is always equal or smaller than the derivation of its context term) and they decrease the size of the equation to which they are applied.

Second, in order to involve in the whole argumentation transformations \( \text{Exp-} \lambda \text{R} \) and \( \text{Exp-AppR} \) we define a new complexity measure involving the size of the grafting: \( \rho(P) = \langle |\theta|, \tau(P) \rangle \).

Since the transformations Normalize, Replace, Dec-\( \lambda \), Dec-App, Dec-\( \sigma \) and Dec-\( \varphi \) do not change the grafting, previous argumentation holds for the resulting lexicographical ordering on these complexity using the ordering on naturals for the first component. Moreover, transformations \( \text{Exp-} \lambda \text{R} \) and \( \text{Exp-AppR} \) always decrease the size of the current grafting \( \theta \). Consequently the application of UniStrat is terminating. \( \square \)

Lemma 44 (Preservation of solutions). If \( \theta \) is a \( \lambda_{s} \)-solution of system \( Q \) and if \( \langle Q, R, \theta \rangle \rightarrow^{r} \langle Q', R', \theta' \rangle \) then \( \theta' \) is a \( \lambda_{s} \)-solution of \( Q' \) and \( \theta \circ \text{Subst}(R) = \text{var}(Q, R) \theta' \circ \text{Subst}(R') \).

Proof. For all the rules except \( \text{Exp-} \lambda \text{R} \) and \( \text{Exp-AppR} \), we have \( \theta' = \theta \). Additionally, since the rules \( \text{Exp-} \lambda \text{R} \) and \( \text{Exp-AppR} \) preserve solutions, we have that \( \theta' \) is a solution of \( Q' \). Observe that the equality modulo \( \lambda_{s} \) is introduced by possible normalization steps.

As the proofs for \( \text{Exp-} \lambda \text{R} \) and \( \text{Exp-AppR} \) are similar, we only do one. By the definition of \( \text{Exp-} \lambda \text{R} \), \( \theta \) is a \( \lambda_{s} \)-solution of \( Q \). Let \( Z \) be a variable in \( \text{var}(Q, R) \) then either \( Z = X \) or \( Z \neq X \). In the first case, \( \theta \) satisfies \( \theta(X) = (\theta \circ \text{Subst}(R))(X) = \theta(X) = \lambda_{A} a \), and \( (\theta \circ \text{Subst}(R'))(X) = \theta(\lambda_{A} Y) = \lambda_{A} a \). In the second case, both graftings give the same image for \( Z \). \( \square \)

Lemma 45 (Construction of solutions). Let \( \langle Q_0, R_0, \theta_0 \rangle \rightarrow \langle Q_1, R_1, \theta_1 \rangle \rightarrow \cdots \rightarrow \langle Q_n, R_n, \theta_n \rangle \) be a finite derivation applying transformations of UniStrat starting from the problem \( P_0 = \langle Q_0, R_0 \rangle \) and the \( \lambda_{s} \)-normalized solution \( \theta_0 \). Then, \( \theta_0 = \text{var}(P_0) \theta_n \circ \text{Subst}(R_n) \), where \( \theta_n \) is a solution of the solved form \( Q_n \).

Proof. Observe that \( \theta_0 = \theta_0 \circ \text{Subst}(R_0) \). From Lemmas 43 and 44, the derivation originated from \( \langle Q_0, R_0, \theta_0 \rangle \) is finite, say of length \( n \), and we have: \( \theta_0 \circ \text{Subst}(R_0) = \text{var}(P_0) \theta_1 \circ \text{Subst}(R_1) = \text{var}(P_1) \cdots = \text{var}(P_n-1) \theta_n \circ \text{Subst}(R_n) \). By Lemma 39, \( Q_n \land R_n \) should be a solved form. Moreover, \( \text{var}(R_0) \supseteq \text{var}(R_1) \supseteq \cdots \supseteq \text{var}(P_n) \) since the set of variables of the unification problems could only decrease due to the Normalize rule. Then we have \( \theta_0 = \text{var}(P_n) \theta_n \circ \text{Subst}(R_n) \). \( \square \)

Theorem 46 (Completeness of UniStrat). The \( \lambda_{s} \)-unification rules describe a correct and complete \( \lambda_{s} \)-unification procedure in the sense that, given a \( \lambda_{s} \)-unification problem \( P \):

if the \( \lambda_{s} \)-unification rules lead in a finite number of steps to a disjunction of systems having one of its one constitutive system solved, then the problem \( P \) is \( \lambda_{s} \)-unifiable and a solution to \( P \) is the solution constructed in Lemma 37 for a solved constitutive system,

if \( P \) has a unifier \( \theta \) then the strategy UniStrat leads in a finite number of steps to a disjunction of systems such that one constitutive system is solved and, like \( P \), has a unifier.

Proof. Straightforward, using Lemma 43 and Theorem 311. \( \square \)
5 HOU in the pure λ-calculus

We present in an informal way two examples on how to apply our λσ-unification method in order to solve HOU problems in the pure λ-calculus. We compare our work to the application of λσ-HOU.

Observe firstly that unifying two terms a and b in the λ-calculus consists in finding a substitution θ such that θ(a) = βη b. But in the λ-calculus the notion of substitution is different from the first order one or grafting, as was shown in Section 2. Thus using the notation of substitution in Definitions 22 and 23, a unifier in the λ-calculus of the problem λ X = βη λ 2 (where = βη denotes the congruence generated by the β- and η-rules of λDB(X)) is not a term t = θX such that λ t = βη λ 2 but a term t = θX such that λ t = θλ X = λ X[t/λ] = λ X X t/λ = λ t/ and not λ t. This observation can be extended to any unifier and by translating appropriately λ-terms a, b ∈ λDB(X), the HOU problem a = βη b can be reduced to equalational unification. [DHK00] presents a translation called pre-cooking from λDB(X) terms into the λσ-calculus such that searching for solutions of the corresponding λσ-unification problem corresponds to searching for solutions of the HOU problem a = βη b. In the following examples, we illustrate informally the analogous situation in the λσ-calculus.

Example 51 Consider the higher order unification problem λ (X 2) = βη λ 2, where 2 and X are of type A and A → A, respectively. Observe that applying a substitution solution θ to the λDB(X)-term λ (X 2) gives θ (λ (X 2)) = λ θ (X 2) when the λσ-calculus we are searching for a grafting θ such that θ (λ (ϕ(2) X 2)) = λσ θ (Y 2). Correspondingly, in the λσ-calculus, λ (X 2) is translated or pre-cooked into λ (X[θ]) 2. Observe that this correspondence follows from lemma 28. Then we should search for unifiers for the problem λ (ϕ(2) X 2) = βη λ 2, where 2 and X are of type A and A → A, respectively. By applying Dec-λ and Exp-λ we get (ϕ(2) X 2) = βη λ 2 and subsequently ΎX (ϕ(2) X 2) = βη λσ λ Y. Then by applying Replace and Normalize we obtain ΎX (ϕ(2) Y 2) = βη λσ λ Y and ΎX (ϕ(2) Y 2) = βη λσ λ Y. Now, by applying rule Exp-app we obtain

(ā (ϕ(2) Y 2)) = βη λσ λ Y ∧ (Y = βη λσ λ Y ∧ X = βη λσ λ Y) which by Replace gives

(ā (ϕ(2) Y 2)) = βη λσ λ Y ∧ X = βη λσ λ Y) and finally, by Normalize

(ā (ϕ(2) Y 2)) = βη λσ λ Y ∧ (Y = βη λσ λ Y ∧ X = βη λσ λ Y) in this way substitution solutions X/λ 1 and X/λ 2 are found.

To complete the analysis note that by Definitions 22, 23 and β-reduction in λDB(X) we have:

{X/λ 1\} {λ (X 2)} = λ (ϕ(2) X 2) = λ (λ (X 2) 1) = βλ 2 and

{X/λ 2\} {λ (X 2)} = λ (ϕ(2) X 2) = λ (λ (X 2) 1) = βλ 2

Observe that the last application of β-reduction is as follows: (λ 3 2) = β{1/2}(3) = 2.

In general, before the unification process, a λ-term a should be translated into the λσ-term a′ resulting by simultaneously replacing each occurrence of a meta-variable X at position i in a with ϕ(2)i X, where k is the number of abstractions between the root position of a, ε, and position i. If k = 0 then the occurrence of X remains unchanged. Essentially, what the pre-cooking translation defined in [DHK00] does is to transcribe all occurrences of de Bruijn indices n into [1..n−1] and all occurrences of meta-variables X into X[X], where k is determined as above. Notice that the two pre-cooking translations can be implemented non-recursively in an efficient way.

Example 52 Consider the HOU problem F (f (a)) = βη f (F (a)). In λDB(X) it can be seen as (X 2 1) = βη (2 1). Since there are no abstractions at the terms of the equational problem, the equation remains unchanged: (X 2 1) = βη (X 2 1).

For simplicity we omit existential quantifiers. After one application of Exp-λ and another of Replace we get

(λ Y 1) 2 = βη (λ Y 1) ∧ X = βη λ Y ∧ X = βη λ Y

And by one application of Exp-App we get

(2 Y 1) 2 = βη (2 Y 1) ∧ X = βη (2 Y 1) ∧ X = βη λ Y ∧ Y = βη λ Y

Note that other possible cases do not produce solved forms. By Replace and Normalize we get:

{(2 1) = βη (2 1) ∧ X = βη λ Y \} ∧ ((2 H 1)(2 1) = βη (2 H 1)(2 1) ∧ X = βη λ (3 H 1))

From which we get the first solved system corresponding to the identity solution: X/λ 1.
Subsequently, other solutions can be obtained from the equational system

$$(2 \ H_1 \sigma^1(2 \ 1) = \lambda_{s_0} \ (2 \ (2 \ H_1 \sigma^1(1))) \ \wedge \ X = \lambda_{s_0} \ \lambda \ (3 \ H_1))$$

In fact, by Dec-App and Exp-App we obtain

$$H_1 \sigma^1(2 \ 1) = \lambda_{s_0} \ ((2 \ H_1 \sigma^1(1)) \ \wedge \ X = \lambda_{s_0} \ \lambda \ (3 \ H_1)) \ \wedge \ (H_1 = \lambda_{s_0} \ 1 \ \vee \ H_1 = \lambda_{s_0} \ (3 \ H_2))$$

Other possible cases do not produce solved forms. By Replace and Normalize we obtain $((2 \ 1) = \lambda_{s_0} \ ((2 \ (2 \ H_2 \sigma^1(1)) \ \wedge \ X = \lambda_{s_0} \ \lambda \ (3 \ H_2)))$, from where we have the second solved system corresponding to the grafting solution: $\{X/\lambda \ (3 \ 1)\}$. This corresponds to the solution $F = f_1$ in fact, by replacing $X$ with $\lambda \ (3 \ 1)$ in the original unification problem we obtain $\lambda \ (3 \ 1) \ (2 \ 1) = \lambda_{s_0} \ 2 \ (\lambda \ (3 \ 1) \ 1))$, from where it is clear that de Bruijn indices 3 and 2 correspond to the same operator. Additionally, note that $(\lambda \ (3 \ 1) \ (2 \ 1)) \rightarrow (2 \ (2 \ 1))$ and $(2 \ (\lambda \ (3 \ 1) \ 1)) \rightarrow (2 \ (2 \ 1))$.

Hence, applying Dec-App, Exp-App, Replace and Normalize to the equational system $((2 \ H_2 \sigma^1(1)) \ \wedge \ X = \lambda_{s_0} \ \lambda \ (3 \ H_2))$ we obtain the third solved system giving the grafting solution $\{X/\lambda \ (3 \ (3 \ 1))\}$ corresponding to the solution $F = f f f$. The unification process continues infinitely producing solved systems corresponding to the grafting solutions $\{X/\lambda \ (3 \ (3 \ (3 \ 1)))\}$ (i.e. $F = f f f f$, $\{X/\lambda \ (3 \ (3 \ (3 \ (3 \ 1))))\}$ (i.e. $F = f f f f f$), etc.

Now we can define our pre-cooking translation.

**Definition 53 (pre-cooking).** Let $a \in A_{dB}(X)$ such that $\Gamma \vdash a_{dB}(X) \ a : T$. To every variable $X$ of type $A$ occurring at $a$ we associate the same type and context $\Gamma$ in the $\lambda_{s_0}$-calculus. The pre-cooking of $a$ from $A_{dB}(X)$ to the $\lambda_{s_0}$-calculus is defined by $a_{pre} = PC(a, 0)$ where $PC(a, n)$ is defined by:

1. $PC(\lambda_B, a, n) = \lambda_B. PC(a, n + 1)$
2. $PC((a \ b), n) = (PC(a, n) \ PC(b, n))$
3. $PC(k, n) = k$
4. $PC(X, n) = in \ n = 0 \ then \ X \ else \ \lambda_{s_0} \ \varphi_{0}^{n+1} \ X$

**Lemma 54.** If $\Gamma \vdash a_{dB}(X) \ a : T$, then $\Gamma \vdash a_{pre} : T$.

**Proof.** We prove the more general result: if $A_1, \ldots, A_n, \Gamma \vdash a_{dB}(X) \ a : T$ and if to every variable occurring at $a$, the same type and context $\Gamma$ is associated, then $A_1, \ldots, A_n, \Gamma \vdash PCA_{dB}(a, n) : T$. This is done by induction on the structure of terms, for all $n$.

Initially, observe that cases $a = k$ and $a = (a_1 \ a_2)$ are simple. Afterwards, suppose that $a = \lambda_B.\, b$. Then $T = B \rightarrow C$ and $B, A_1, \ldots, A_n, \Gamma \vdash b : C$. Thus $B, A_1, \ldots, A_n, \Gamma \vdash PC(b, n + 1) : C$ and $A_1, \ldots, A_n, \Gamma \vdash PCA_{dB}(\lambda_B.\, b, n) = \lambda_B. PC(b, n + 1) : B \rightarrow C$. Finally, for $a = X$ by definition of $\Gamma \vdash a_{dB}(X) \ X : T, \Gamma \vdash a_{pre} : X : T$ and $A_1, \ldots, A_n, \Gamma \vdash \varphi_{0}^{n+1} \ (X) : T$.

Now pre-cooking is justified by the following proposition that relates substitution in $A_{dB}(X)$ and grafting in $\lambda_{s_0}$.

**Proposition 55 (Semantics of pre-cooking).** Let $a, b_1, \ldots, b_p$ be terms of $A_{dB}(X)$. We have:

$$(a[X_1/b_1, \ldots, X_p/b_p])_{pre} = a_{pre}[X_1/b_1_{pre}, \ldots, X_p/b_p_{pre}]_g$$

**Proof.** The more general fact $PC(a[X_1/b_1^{i+1}, \ldots, X_p/b_p^{i+1}]), i) = PC(a, i)[X_1/b_1_{pre}, \ldots, X_p/b_p_{pre}]_g$ is what we will prove. Observe that the case $i = 0$ corresponds to the proposition: $(a[X_1/b_1])_{pre} = PC(a[\lambda_X.\, b_1^{i+1}], 0) = PC(a, 0)[X_1/b_1_{pre}]_g = a_{pre}[X_1/b_1_{pre}]_g$. The proof is done by induction on the structure of terms for all $i$.

- $a = \lambda_b \ PC((\lambda_b.\, b[X_1/b_1^{i+1}]), i) = \lambda_b. PC((\lambda_b.\, b[X_1/b_1^{i+1}]), i) = \lambda_b. PC((\lambda_b.\, b[X_1/b_1^{i+1}]), i + 1)$. By induction hypothesis, the previous expression is equal to $\lambda_b. PC(b[i+1]) \ (X_1/b_1_{pre})_g = \lambda_b. PC(b[i+1]) \ (X_1/b_1_{pre})_g = PC(\lambda_b, i)[X_1/b_1_{pre}]_g$.

- $a = (a_1 \ a_2)$. Observe that $PC((a_1 \ a_2)[X_1/b_1^{i+1}], i) = PC((a_1 \ a_2)[X_1/b_1^{i+1}], i)$ and this is equal to $(PC(a_1 \ a_2)[X_1/b_1^{i+1}], i) = PC(a_1 \ a_2)[X_1/b_1^{i+1}, i)$. By applying the induction hypothesis the last expression is equal to $PC((a_1 \ a_2)[X_1/b_1^{i+1}], i) = PC(a_1 \ a_2)[X_1/b_1^{i+1}, i)$. To conclude, the last expression is equal to $PC((a_1 \ a_2)[X_1/b_1_{pre}])_g = PC((a_1 \ a_2)[X_1/b_1_{pre}])_g$.
\( a = n \), \( PC(n \{ X_j / \beta \}) \), \( i \) \( = PC(n, i) = n \{ X_j / \beta \} \Rightarrow PC(n, i) \{ X_j / \beta \} \).

\( a = X \). We have two cases: either \( X = X_j \), for some \( 1 \leq j \leq p \), or \( X \neq X_j \), for all \( 1 \leq j \leq p \). The interesting case is the first one. Suppose that \( X = X_j \), for some \( 1 \leq j \leq p \). Then we should prove that

\[
PC(\beta_j^i, i) = PC(X_j, i) \{ X_j / \beta_j \} = \varphi_0^i \beta_j.
\]

We will prove the more general fact that

\[
PC(\beta_j^i, i + k) = \varphi_k^{i+1} PC(\beta_j, k).
\]

This is done by induction on the structure of \( b \) as follows:

\(- \quad b = \lambda c. PC((\beta_j^i, i + k) = PC(\lambda c. \beta_j^i, i + k) \) (i.e., \( c \in \{ (1..k, \beta_j^i) \}) our proof uses pure term objects by selecting the appropriate super and subscripts for the \( \varphi \) operator (i.e., \( \varphi_k \)).

The following proposition presents necessary facts for relating the existence of solutions for unification problems in the pure lambda-calculus and in the \( \lambda \)-calculus.

**Proposition 56.** Let \( a \) and \( b \) be terms in \( A_{H4}(X) \). Then

1. \( a \rightarrow \beta b \) implies \( a_{pc} \rightarrow_\lambda^* b_{pc} \).
2. If \( a \) is \( \beta \eta \)-normal then \( a_{pc} \) is \( \lambda_\varepsilon \)-normal.

**Proof.** First, we will prove the more general fact that trivia (\( \lambda \{a\} \{1/b\} \)) implies (\( \lambda \{a\} \{1/b\} \)). This is done by induction on \( a \) for all \( k \). The case \( k = 0 \) corresponds to our case of interest. Initially, notice that (\( \lambda \{a\} \{1/b\} \)) (\( \lambda \{a\} \{1/b\} \)) (\( \lambda \{a\} \{1/b\} \)).

\( a = n \). Case \( n > k \), \( PC(n \{a\} \{i+k\}) = PC(n+i, k) = n+i \) and \( \varphi_k^{i+1} PC(n, k) = \varphi_k^{i+1} n = n+i \). Conversely, \( a = X \). Then we have that \( \lambda \{a\} \{1/b\} \). We prove the more general fact that

\[
\lambda \{a\} \{1/b\} \Rightarrow \lambda \{a\} \{1/b\} \).
\]

\( a = n \). On one side, \( \lambda \{a\} \{1/b\} \) (\( \lambda \{a\} \{1/b\} \)) (\( \lambda \{a\} \{1/b\} \)). This is done by structural induction on \( b \) for all \( k > 0 \) and \( i \geq 0 \).

\( b = n \). We have two cases: either \( i = 0 \) or \( i > 0 \). In the first case we have \( \lambda \{a\} \{1/b\} \) (\( \lambda \{a\} \{1/b\} \)) (\( \lambda \{a\} \{1/b\} \)) (\( \lambda \{a\} \{1/b\} \)) (\( \lambda \{a\} \{1/b\} \)). We prove that \( \lambda \{a\} \{1/b\} \) (\( \lambda \{a\} \{1/b\} \)) (\( \lambda \{a\} \{1/b\} \)).
We prove the more general fact that if \( a \) is \( \beta\eta\)-normal then for all \( k, PC(a, k) = \lambda s_e\)-normal. This is done by structural induction on the structure of \( a \) for all \( k \), as follows.

- **\( a = n \).** Obvious.
- **\( a = X \).** \( PC(X, k) = \varphi^k + X \) that is \( \lambda s_e\)-normal.
- **\( a = \lambda b \).** \( PC(\lambda b, k) = \lambda PC(b, k + 1) \) and since \( b \) should be \( \beta\eta\)-normal, \( PC(b, k + 1) \) is \( \lambda s_e\)-normal. Then \( \lambda PC(b, k + 1) \) is \( \lambda s_e\)-normal too.
- **\( a = (b \ c) \).** Since \( b \) and \( c \) are \( \beta\eta\)-normals then \((PC(a, k) = \lambda s_e\)-normal).

**Third.** We prove the more general fact that if \( \lambda (\lambda (X a) 1) \rightarrow d \) then \( (\lambda (\lambda (X a) 1))_p \rightarrow \eta \ d \). Observe that \( d \) should be of the form \( \lambda^k c \) and such that \((\lambda^k c)^+ = \lambda^k c^{\downarrow +} = \lambda^k a \). Then \( c^{\downarrow +} = a \). Notice that our case of interest is the corresponding to \( k = 0 \). The proof is by induction on the structure of \( a \) for all \( k \).

- **\( a = n \).** Firstly, notice that \( \lambda (\lambda (X a) 1) \rightarrow^\eta \lambda^X a \). Then \( \lambda^X a = \lambda^X (d^{\downarrow +}) \) and \( \lambda a = d^{\downarrow +} \). Then \( PC(\lambda (\lambda (X a) 1), 0) = \lambda (\lambda (PC(X, a + 1)) = \lambda (d^{\downarrow +}, c^{\downarrow +}), \eta, \phi_0, c^{\downarrow +}, c^{\downarrow +}, c^{\downarrow +}) \). It should hold that \( c^{\downarrow +} = a \) for \( i = 1, 2 \). Then \( \lambda (\lambda (a_1 a_2) 1) \rightarrow n \lambda^X (c_1 c_2) \) if \( \lambda^X (a_1 a_2) = \lambda^X (c_1 c_2) \). By the induction, \( \lambda (\lambda X (a_1, a_2) 1) \rightarrow \eta \lambda^X (c_1 c_2) \).

**Fourth.** On one side, that \( a = \beta\eta b \) implies \( a_p = a_{\lambda s_e} b_p \) is proved by induction on the length of the proof of \( a = \beta\eta b \) using the previous first and second items.

On the other side, suppose that \( a_p = a_{\lambda s_e} b_p \) and let \( a' \) and \( b' \) normal forms of \( a \) and \( b \), respectively. By previous items, terms \( a_p \) and \( b_p \) reduce to \( a'_p \) and \( b'_p \) respectively. Consequently, \( a_{\lambda s_e} b_{\lambda s_e} = a_{\lambda s_e} b_{\lambda s_e} \).

Again, our proof differs from the corresponding in [DHK00] in that we avoid the use of complicated substitution objects because we profit from the semantics of the \( \varphi \) operator of the \( \lambda s_e \)-calculus.

Finally, we relate solutions and their existence in the pure \( \lambda \)-calculus and for the corresponding pre-cooked terms in the \( \lambda s_e \)-calculus.

**Proposition 57 (Correspondence between solutions).** Let \( a \) and \( b \) be terms in \( \Lambda_\lambda (X) \). Then there exist terms \( N_1, \ldots, N_p \) in \( \Lambda_\lambda (X) \) such that \( a \{X_i \mid N_i \} \) if and only if \( \lambda s_e \)-terms \( M_1, \ldots, M_p \) such that \( a_{\lambda s_e} b_{\lambda s_e} \{X_i \mid M_i \} \).

**Proof.** On the one side, suppose that \( \{X_i \mid N_i \} \) is a solution of the unification problem \( a = \beta\eta b \). Then \( a \{X_i \mid N_i \} = \beta\eta b \{X_i \mid N_i \} \). By the fourth item of the Proposition 56, \( (a \{X_i \mid N_i \})_p = a_{\lambda s_e} b_{\lambda s_e} \{X_i \mid M_i \} \).

On the other side, suppose that \( a_{\lambda s_e} b_{\lambda s_e} \{X_i \mid M_i \} \). We select terms \( N_i, i = 1, \ldots, p \), in the range of the pre-cooking translation such that \( N_i = \lambda s_e M_i \) and let \( M_i \) be terms in \( \Lambda_\lambda (X) \) such that \( M_i = N_i \). Then \( a_{\lambda s_e} b_{\lambda s_e} \{X_i \mid M_i \} \).

**6 Conclusions.**

Following the \( \lambda \eta \)-unification approach introduced in [DHK00], we have developed an effective strategy for implementing the \( \lambda s_e \)-unification rules presented in [ARK00]. Additionally, we presented a pre-cooking translation that transcribes pure \( \lambda \)-terms in de Bruijn notation into \( \lambda s_e \)-terms, for which the search of grafting solutions corresponds to substitution solutions in the pure \( \lambda \)-calculus.
Note that correctness and completeness proofs for the \(\lambda\sigma\) and the \(\lambda s\)-unification strategies don't differ because these strategies are based on an appropriate ordering of the application of the unification rules which is in a certain way independent of the calculi. Of course, the strategies differ on the unification transformations, because these are built on different unification rules, which is the subject of [ARK00].

However, our proofs for the \(\lambda s\)-HOU differ from the ones for the \(\lambda\sigma\)-HOU mainly because of the differences between the two calculi. Moreover, proofs of correctness of the semantics and preservation of solutions for our pre-cooking translation are very different from the ones for the \(\lambda\sigma\)-HOU, since our definition of pre-cooking translation depends directly on the syntactic properties and semantics of the \(\lambda s\)-calculus.

More concretely, our pre-cooking translation transcribes a term \(a\) by replacing each occurrence of a metavariable \(X\) with \(\varphi^{-1}(X)\) where \(\varphi\) is the number of abstractors between the position of the occurrence of \(X\) and the root position. Additionally, the pre-cooking translation in [DHK00] transcribes each occurrence of a de Bruijn index \(n\) in \(\varphi\) into \(\varphi^n\). Conformity of the two pre-cooking translations is therefore evident. But our proofs differ from the corresponding ones in [DHK00] in that we don't need the use of complex substitution objects because of the appropriate semantics and flexibility of the \(\varphi\) operator in the \(\lambda s\)-calculus. This can be observed in the proof of the correct semantics of the pre-cooking translation (Proposition 53) and the proof of Proposition 56 which relates the existence of unification solutions in the \(\lambda\) and the \(\lambda s\)-calculus. In these proofs, only a correct selection of the scripts for the operator \(\varphi\) was necessary, avoiding the manipulation of substitution objects as in the \(\lambda\sigma\)-HOU approach.

Of course, much work remains to be done in order to obtain a complete HOU theoretical framework which could be implemented. In particular, it is necessary to present a back translation that enables the reconstruction of solved forms of unification problems in the \(\lambda s\)-calculus into a description of solutions of the corresponding HOU problems in the pure \(\lambda\)-calculus.

Additionally, a formal distinction, from the practical point of view, between the \(\lambda s\)-calculus (and our procedure) and the suspension calculus developed by Nadathur and Wilson in [NW98,NW99] (and used in the implementation of the higher order logical programming language AProlog) should be elaborated. This is meaningful, since the \(\lambda s\)-calculus and the calculus of [NW98,NW99] have correlated nice properties. For instance the laziness in the substitution needed in implementations of \(\beta\)-reduction, that arises naturally in the \(\lambda s\)-calculus, is provided as the informal but empirical concept of suspension of substitutions by Nadathur and Wilson rewrite rules. Establishing these precise distinctions and correlations is important for estimating the appropriateness of the \(\lambda s\)-HOU approach in that practical framework.

References