Strategies for Simply-Typed Higher Order Unification via λs_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 λs_e-style of explicit substitution is proposed. The strategy is based on a λs_e-unification method recently developed by the authors. A *pre-cooking* translation for applying the λs_e-style of unification to HOU in the pure λ-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 λ-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 λs_e-style of explicit substitution [KR97]. It is claimed in [ARK00] that λs_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 λs_e-unification and a pre-cooking translation for applying it to HOU in the λ-calculus. It should be stressed that $\lambda\sigma$ and λs_e are two different styles of explicit substitution which are not isomorphic. This implies that reworking the HOU method in λs_e is not a translation of work already done in $\lambda\sigma$. Many rules and proofs of the λs_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 λs_e- and $\lambda\sigma$-calculi. In Section 3, we review our λs_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 λ-calculus. Then we conclude and discuss future work in Section 6.

2 Background

We assume familiarity with the notion of term algebra $\mathcal{T}(\mathcal{F},\mathcal{X})$ built on a (countable) set of variables $\mathcal{X}$ and a set of operators $\mathcal{F}$. Variables in $\mathcal{X}$ are denoted by upper case last letters of the Roman alphabet $X, Y, \ldots$. For a term $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $var(t)$ denotes the set of variables occurring in t. We assume familiarity with the λ-calculus as in [Bar84] and with the basic notions and notation of rewriting theory as in [BN98]. For a *reduction relation* R over a set A, $(A, \to_R)$, we denote with $\to_R^*$ the *reflexive and transitive closure* of $\to_R$. The subscript R is usually omitted. When $a \to^* b$ we say that there exists a *derivation* from a to b. Syntactical identity is denoted by $a = b$.

A **valuation** is a mapping from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F},\mathcal{X})$. The homeomorphic extension of a valuation, θ, from its domain $\mathcal{X}$ to the domain $\mathcal{T}(\mathcal{F},\mathcal{X})$ is called the **grafting** of θ. 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 θ is defined by $Dom(\theta) = \{X \mid X\theta \neq X, X \in \mathcal{X}\}$ and its **range** by $Ran(\theta) = \cup_{X \in Dom(\theta)} var(X\theta)$. The set of variables involved in θ is $var(\theta) = Dom(\theta) \cup Ran(\theta)$. A valuation and its corresponding grafting θ are explicitly denoted by $\theta = \{X/X\theta \mid X \in Dom(\theta)\}$. When necessary, explicit representations of graftings are differentiated from substitutions by a "g" subscript: $\{X/X\theta \mid X \in Dom(\theta)\}_g$.

* 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 λs_e-calculus, their typed versions and normal form characterizations are briefly included.

2.1 The λ-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 **λ-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 λ-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 λ-calculus with names uses *variable renaming* via α-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 α-conversions, β- and η-reduction rules are defined in $\Lambda(\mathcal{X})$ respectively by $(\lambda_x.a\ \ b) \to a\{x/b\}$ and $\lambda_x.(a\ \ x) \to a$, if $x \notin \mathcal{F}var(a)$, where $\mathcal{F}var(a)$ 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 λ-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, λ-calculus should be defined as the term algebra, $\Lambda(\mathcal{V}, \mathcal{X})$, over the set of operators $\{\lambda_x._ \mid x \in \mathcal{V}\} \cup \{(\_\ \_)\} \cup \mathcal{V}$ 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 β- and η-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, $\lambda_x.(\lambda_z.(x\ \ z)\ \ (x\ \ z))$ is translated into $\lambda.(\lambda.(\mathtt{2}\ \ \mathtt{1})\ \ (\mathtt{1}\ \ \mathtt{4}))$. Indices for free variables are appropriately selected to avoid relating them with abstractors.

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

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

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

Definition 22. *The application of the* **substitution** *with b of $n \in \mathbb{N} \setminus \{0\}$ 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.* $\{n/b\}\mathtt{m} = \begin{cases} \mathtt{m-1}, & \text{if } m > n \\ b, & \text{if } m = n \\ \mathtt{m}, & \text{if } m < n \end{cases}$ *if* $m \in \mathbb{N}$.

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 θ, is defined inductively as follows:*

a) $\theta(\mathtt{m}) = \mathtt{m}$ *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 θ^+ 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 η-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 "$x \notin \mathcal{F}var(a)$" 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 β-reduction is defined as $(\lambda.a\ \ b) \to \{1/b\}a$ and η-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* $\boldsymbol{\lambda\sigma}$ *presented in Table 1 where* TERMS $a ::= \mathtt{1} \mid X \mid (a\ b) \mid \lambda a \mid a[s]$ *and* SUBS $s ::= id \mid \uparrow \mid a.s \mid s \circ s$.

Table 1. $\boldsymbol{\lambda\sigma}$ Rewriting System of the $\lambda\sigma$-calculus

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

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

The rewriting system $\boldsymbol{\lambda\sigma}$ satisfies the following properties: it is locally confluent [ACCL91], confluent on substitution-closed terms (i.e., terms without substitution variables) [Río93] and not confluent on open terms (i.e., terms with term and substitution variables) [CHL96].

Proposition 25 ([Río93]). *Any $\lambda\sigma$-term in $\boldsymbol{\lambda\sigma}$-normal form is of one of the following forms: a) λa; b) $(a\ b_1 \dots b_n)$, where a is either $\mathtt{1}$, $\mathtt{1}[\uparrow^n]$, X or $X[s]$ being s a substitution term different from id in normal form; or c) $a_1 \dots a_p.\ \uparrow^n$, where $a_1, \dots, a_p$ are normal terms and $a_p \neq \mathtt{n}$.*

In $\Lambda(\mathcal{X})$ and $\Lambda_{dB}(\mathcal{X})$, 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 $X =_\sigma b[\uparrow]$. 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 $\mathcal{X}$ by $\mathcal{X}$-grafting and explicit lifting. Consequently a notion of $\mathcal{X}$-substitution in $\lambda\sigma$-calculus is unnecessary.

2.3 The λs_e-calculus

The λs_e-calculus avoids introducing two different sets of entities and insists on remaining close to the syntax of the λ-calculus. Next to λ and application, the λs_e-calculus introduces substitution (σ) and updating (φ) operators. In the λs_e-calculus, we let a, b, c, etc. range over the sets of terms Λs. A term containing neither substitution nor updating operators is called a *pure term*.

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

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

In order to clarify differences between the $\lambda\sigma$-calculus and the λs_e-calculus, we show the correspondence between their *Eta* rules; i.e., the correspondence between both conditions $b[\uparrow] = a$ and $\varphi^2_0 b = a$. Remember that in the $\lambda\sigma$-calculus we only use the de Bruijn index $\mathtt{1}$ and that the other indices are codified as $\mathtt{1}[\uparrow^n]$.

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

(σ-generation)	$(\lambda.a\ \ b) \longrightarrow a\,\sigma^1\,b$
(σ-λ-transition)	$(\lambda.a)\,\sigma^i b \longrightarrow \lambda.(a\,\sigma^{i+1}\,b)$
(σ-app-transition)	$(a_1\ \ a_2)\,\sigma^i b \longrightarrow ((a_1\,\sigma^i b)\ \ (a_2\,\sigma^i b))$
(σ-destruction)	$\mathtt{n}\,\sigma^i b \longrightarrow \begin{cases} \mathtt{n-1} & \text{if } n > i \\ \varphi_0^i\,b & \text{if } n = i \\ \mathtt{n} & \text{if } n < i \end{cases}$
(φ-λ-transition)	$\varphi_k^i(\lambda.a) \longrightarrow \lambda.(\varphi_{k+1}^i\,a)$
(φ-app-transition)	$\varphi_k^i(a_1\ \ a_2) \longrightarrow ((\varphi_k^i\,a_1)\ \ (\varphi_k^i\,a_2))$
(φ-destruction)	$\varphi_k^i\,\mathtt{n} \longrightarrow \begin{cases} \mathtt{n+i-1} & \text{if } n > k \\ \mathtt{n} & \text{if } n \leq k \end{cases}$
(Eta)	$\lambda.(a\ \ 1) \longrightarrow b \quad \text{if} \quad a =_{s_e} \varphi_0^2 b$
(σ-σ-transition)	$(a\,\sigma^i b)\,\sigma^j\,c \longrightarrow (a\,\sigma^{j+1}\,c)\ \sigma^i\,(b\,\sigma^{j-i+1}\,c)$ if $\quad i \leq j$
(σ-φ-transition 1)	$(\varphi_k^i\,a)\,\sigma^j\,b \longrightarrow \varphi_k^{i-1}\,a \quad$ if $\quad k < j < k+i$
(σ-φ-transition 2)	$(\varphi_k^i\,a)\,\sigma^j\,b \longrightarrow \varphi_k^i(a\,\sigma^{j-i+1}\,b) \quad$ if $\quad k+i \leq j$
(φ-σ-transition)	$\varphi_k^i(a\,\sigma^j\,b) \longrightarrow (\varphi_{k+1}^i\,a)\,\sigma^j\,(\varphi_{k+1-j}^i\,b) \quad$ if $\quad j \leq k+1$
(φ-φ-transition 1)	$\varphi_k^i\,(\varphi_l^j\,a) \longrightarrow \varphi_l^j\,(\varphi_{k+1-j}^i\,a) \quad$ if $\quad l+j \leq k$
(φ-φ-transition 2)	$\varphi_k^i\,(\varphi_l^j\,a) \longrightarrow \varphi_l^{j+i-1}\,a \quad$ if $\quad l \leq k < l+j$

Example 27 Consider the term $\lambda.((2\ \ \lambda.(1\ \ 3))\ \ 1)$ in $\Lambda_{dB}(\mathcal{X})$. Observe that the *Eta* rule applies, since $\varphi_0^2 b = \varphi_0^2(1\ \ \lambda.(1\ \ 2)) \longrightarrow (\varphi_0^2 1\ \ \varphi_0^2\lambda.(1\ \ 2)) \longrightarrow (\varphi_0^2 1\ \ \lambda.\varphi_1^2(1\ \ 2)) \longrightarrow (\varphi_0^2 1\ \ \lambda.(\varphi_1^2 1\ \ \varphi_1^2 2)) \longrightarrow^* (2\ \ \lambda.(1\ \ 3)) = a$.

Analogously, in the $\lambda\sigma$-calculus we have: $(1\ \ \lambda.(1\ \ 2))[\uparrow] = (1\ \ \lambda.(1\ \ 1[\uparrow]))[\uparrow] \longrightarrow (1[\uparrow]\ \ \lambda.(1\ \ 1[\uparrow])[\uparrow]) \longrightarrow (1[\uparrow]\ \ \lambda.(1\ \ 1[\uparrow])[1.\ \uparrow^2]) \longrightarrow (1[\uparrow]\ \ \lambda.(1[1.\ \uparrow^2]\ \ 1[\uparrow][1.\ \uparrow^2])) \longrightarrow (1[\uparrow]\ \ \lambda.(1[1.\ \uparrow^2]\ \ 1[\uparrow \circ (1.\ \uparrow^2)])) \longrightarrow (1[\uparrow]\ \ \lambda.(1[1.\ \uparrow^2]\ \ 1[\uparrow^2])) \longrightarrow (1[\uparrow]\ \ \lambda.(1\ \ 1[\uparrow^2])) = (2\ \ \lambda.(1\ \ 3))$. •

The correspondence between both *Eta* rules is the case $k = 0$ of the following lemma.

Lemma 28 ([ARK00]). *Let $a \in \Lambda_{dB}$ and a' its corresponding codification in the language of the $\lambda\sigma$-calculus, where all indices $\mathtt{n} \in \mathbb{N}$ occurring at a are replaced with $1[\uparrow^{n-1}]$. Then, for all $k \geq 0$, the σ-normal form of $a'[1.1[\uparrow].\ldots.1[\uparrow^{k-1}].\uparrow^{k+1}]$ is the corresponding codification of the s-normal form of $\varphi_k^2 a$.*

The previous lemma can be straightforwardly extended for terms $a \in \Lambda_{dB}(\mathcal{X})$. In fact, observe that for a meta-variable $X \in \mathcal{X}$ at a position $i \in O(a)$, the corresponding subterms of the σ- and s-normal forms of $a[\uparrow]$ and $\varphi_0^2 a$ are of the form $X[1.1[\uparrow].\ldots.1[\uparrow^{k-1}].\uparrow^{k+1}]$ and $\varphi_k^2 X$, 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 λs_e-calculus over the signature of a first order sorted term algebra $\mathcal{T}_{\lambda s_e}(\mathcal{X})$ built on $\mathcal{X}$, the set of variables of sort TERM and its subsort NAT$\subset$TERM. The set of variables of sort TERM in a term $a \in \mathcal{T}_{\lambda s_e}(\mathcal{X})$ is denoted by $\mathcal{T}var(a)$.

Theorem 29 ([KR97]). *a) The s_e-calculus is weakly normalizing and confluent. b) The λs_e-calculus simulates β-reduction. c) The λs_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 λs_e-normal forms.

Corollary 210 (λs_e-normal forms). *$a \in \Lambda s_{op}$ is a λs_e-normal form iff:*

1. *$a \in \mathcal{X} \cup \mathbb{N}$;*
2. *$a = (b\ \ c)$, where b, c are λs_e-normal forms and b is not an abstraction of the form $\lambda.d$;*

3. $a = \lambda.b$, *where* b *is a* λs_e*-normal form excluding applications of the form* $(c\ \mathtt{1})$ *such that there exists* d *with* $\varphi_0^2 d =_{s_e} c$;
4. $a = b\sigma^j c$, *where* c *is a* λs_e*-normal form and* b *is an* λs_e*-normal form of one of the following forms:* *a)* X, *b)* $d\sigma^i e$, *with* $j < i$ *or* *c)* $\varphi_k^i d$, *with* $j \leq k$;
5. $a = \varphi_k^i b$, *where* b *is a* λs_e*-normal form of one of the following forms:* *a)* X, *b)* $c\sigma^j d$, *with* $j > k+1$ *or* *c)* $\varphi_l^j c$, *with* $k < l$.

2.4 Typed $\boldsymbol{\lambda}$-calculi

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

We recall that an environment Γ 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: TYPES $A ::= A \mid A \to B$ and ENVIRS $\Gamma ::= nil \mid A.\Gamma$. 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 λs_e-calculi is defined by adding to the rules in $\boldsymbol{\lambda\sigma}$ and in λs_e the necessary typing information. Thus, for the typed $\lambda\sigma$-calculus we have the typed rules *(Beta)*, *(Abs)* and *(Eta)* respectively as follows:

$$(\lambda_A.a\ \ b) \longrightarrow a\,[b \cdot id] \qquad (\lambda_A.a)[s] \longrightarrow \lambda_A.a\,[1 \cdot (s \circ \uparrow)] \qquad \lambda_A.(a\ \ 1) \longrightarrow b \text{ if } a =_\sigma b[\uparrow]$$

and for the typed λs_e-calculus:

(σ-generation)	$(\lambda_A.a\ \ b) \longrightarrow a\,\sigma^1\,b$	*(σ-λ-transition)*	$(\lambda_A.a)\,\sigma^i b \longrightarrow \lambda_A.(a\,\sigma^{i+1}\,b)$
(φ-λ-transition)	$\varphi_k^i(\lambda_A.a) \longrightarrow \lambda_A.(\varphi_{k+1}^i\,a)$	*(Eta)*	$\lambda_A.(a\ \ 1) \longrightarrow b$ if $a =_s \varphi_0^2 b$

We denote typability in $\Lambda_{dB}(\mathcal{X})$, the $\lambda\sigma$- and λs_e-calculi by $\vdash_{\Lambda_{dB}(\mathcal{X})}$, $\vdash_{\lambda\sigma}$ and $\vdash_{\lambda s_e}$ respectively.

Characterization of η-long normal forms in the typed $\lambda\sigma$- and λs_e-calculi is necessary to simplify the set of rules of the unification algorithms. Essentially, the use of η-long normal forms guarantees that meta-variables of a functional type $A \to B$ are instantiated with typed terms of the form $\lambda_A.a$.

Definition 211 (η-long normal form in $\lambda\sigma$). *Let* a *be a* $\lambda\sigma$*-normal form term of type* $A_1 \to \ldots \to A_n \to B$ *in the environment* Γ. *The* **η-long normal form (η-lnf)** *of* a, *written* a', *is defined by:*

1. *if* $a = \lambda_C.b$ *then* $a' = \lambda_C.b'$;
2. *if* $a = (\mathtt{k}\ b_1 \ldots b_p)$ *then* $a' = \lambda_{A_1} \ldots \lambda_{A_n}(\mathtt{k} + \mathtt{n}\, c_1 \ldots c_p\, \mathtt{n}' \ldots 1')$ *where* c_i *is the* η*-lnf of the normal form of* $b_i[\uparrow^n]$;
3. *if* $a = (X[s]\, b_1 \ldots b_p)$ *then* $a' = \lambda_{A_1} \ldots \lambda_{A_n}(X[s']\, c_1 \ldots c_p\, \mathtt{n}' \ldots 1')$ *where* c_i *is the* η*-lnf of* $b_i[\uparrow^n]$ *and if* $s = d_1 \ldots d_q.\ \uparrow^k$ *then* $s' = e_1 \ldots e_q.\ \uparrow^{k+n}$ *where* e_i *is the* η*-lnf of* $d_i[\uparrow^n]$.

Definition 212 (η-long normal form in λs_e). *Let* a *be a* λs_e*-normal form term of type* $A_1 \to \ldots \to A_n \to B$ *in the environment* Γ. *The* **η-long normal form (η-lnf)** *of* a, *written* a', *is defined by:*

1. *if* $a = \lambda_C.b$ *then* $a' = \lambda_C.b'$;
2. *if* $a = (b_1 \ldots b_p)$ *then* $a' = \lambda_{A_1} \ldots \lambda_{A_n}(c_1 \ldots c_p\, \mathtt{n}' \ldots 1')$, *where* c_i *is the* η*-lnf of the normal form of* $\varphi_0^{n+1} b_i$;
3. *if* $a = b\sigma^i c$ *then* $a' = \lambda_{A_1} \ldots \lambda_{A_n}(d'\sigma^{i+n} e'\, \mathtt{n}' \ldots 1')$, *where* d', e' *are the* η*-lnfs of the normal forms of* $\varphi_0^{n+1} b$ *and* $\varphi_0^{n+1} c$, *respectively;*
4. *if* $a = \varphi_k^i b$ *then* $a' = \lambda_{A_1} \ldots \lambda_{A_n}(\varphi_k^i c'\, \mathtt{n}' \ldots 1')$, *where* c' *is the* η*-lnf of the normal form of* $\varphi_0^{n+1} b$.

The set of unification rules of both HOU methods are constructed by combining the different types of η-lnfs enumerated in Definitions 211 and 212 obtaining different types of equational problems. For the HOU setting based on the λs_e-style an additional characterization of λs_e-normal terms whose main operators are either σ or φ will be useful in order to combine directly η-lnfs of type 2 (See subsection 2.5) with the ones of type 3. and 4. This simplifies the comparison of both HOU approaches.

Definition 213 (Long normal form (lnf)). *Let* a *be either a* $\lambda\sigma$*-term or a* λs_e*-term. The* **long normal form** *of* a *is defined as the* η*-lnf of its* $\beta\eta$*-normal form.*

In both typed $\lambda\sigma$- and λs_e-calculi we have that two terms are $\beta\eta$-equivalent iff they have the same lnf.

2.5 λs_e-normal forms

We present a characterization of λs_e-normal terms whose main operators are either σ or φ (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 σ operator or arguments of the φ operator at λs_e-normal terms are neither applications, nor abstractions, nor de Bruijn indices. For instance, $\varphi_i^j(a\ \ b) \to (\varphi_k^i a\ \ \varphi_k^i b)$, $(a\ \ b)\sigma^i c \to (a\sigma^i c\ \ b\sigma^i c)$. 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 φ and σ whose left innermost argument is a meta-variable; for instance, $((\varphi_{i_3}^{j_3}((\varphi_{i_1}^{j_1}X)\sigma^{i_2}a))\sigma^{i_4}b)\sigma^{i_5}c$.

Definition 214. *Let t be a λs_e-normal term whose root operator is either σ or φ and let X be its left innermost meta-variable. Denote by $\psi_{i_k}^{j_k}$ the operator at the k^{th} position following the sequence of operators φ and σ, considering only left arguments of the σ operators, in the innermost outermost ordering. Additionally, if $\psi_{i_k}^{j_k}$ corresponds to an operator φ then j_k and i_k denote its super and subscripts, respectively and if $\psi_{i_k}^{j_k}$ corresponds to an operator σ then $j_k = 0$ and i_k denotes its superscript. Let a_k denote the corresponding right argument of the k^{th} operator if $\psi_{i_k}^{j_k} = \sigma^{i_k}$ and the empty argument if $\psi_{i_k}^{j_k} = \varphi_{i_k}^{j_k}$. The* **skeleton** *of t written $sk(t)$ is $\psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p)$.*

Example 215 Consider a λs_e-normal term t of the form $((\varphi_{i_3}^{j_3}((\varphi_{i_1}^{j_1}X)\sigma^{i_2}a))\sigma^{i_4}b)\sigma^{i_5}c$. Then the skeleton of t, $sk(t)$, is $\psi_{i_5}^0\psi_{i_4}^0\psi_{i_3}^{j_3}\psi_{i_2}^0\psi_{i_1}^{j_1}(X, a, b, c)$. •

Lemma 216. *Let t be a λs_e-normal term whose root operator is either σ or φ and let the skeleton of t, $sk(t) = \psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p)$. Successive subscripts i_k and i_{k+1} satisfy the following conditions:*

1. *$i_k > i_{k+1}$ if ψ_k and ψ_{k+1} are both σ operators or both φ operators;*
2. *$i_k \geq i_{k+1}$ if ψ_k and ψ_{k+1} are φ and σ operators, respectively;*
3. *$i_k > i_{k+1} + 1$ if ψ_k and ψ_{k+1} are σ and φ operators, respectively.*

Proof. By simple analysis of the arithmetic constraints at the λs_e rewrite rules. □

3 Unification in the λs_e-calculus

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

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

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

Definition 31. *A* **λs_e-unification problem** *P is a unification problem in the algebra $\mathcal{T}_{\lambda s_e}(\mathcal{X})$ modulo the equational theory of λs_e. An* **equation** *of such a problem is denoted $a =_{\lambda s_e}^{?} b$, where a and b are λs_e-terms of the same sort. An equation is called trivial when it is of the form $a =_{\lambda s_e}^{?} a$.*

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. λs_e-unification rules

(Dec-λ)	$P \wedge \lambda_A.a =^?_{\lambda s_e} \lambda_A.b \quad \rightarrow \quad P \wedge a =^?_{\lambda s_e} b$
(Dec-App)	$P \wedge (\mathtt{n}\ a_1 \dots a_p) =^?_{\lambda s_e} (\mathtt{n}\ b_1 \dots b_p) \quad \rightarrow \quad P \bigwedge_{i=1..p} a_i =^?_{\lambda s_e} b_i$
(App-Fail)	$P \wedge (\mathtt{n}\ a_1 \dots a_p) =^?_{\lambda s_e} (\mathtt{m}\ b_1 \dots b_q) \quad \rightarrow \quad \mathbb{F}$ if $\mathtt{n} \neq \mathtt{m}$
(Dec-σ)	$P \wedge a\sigma^i b =^?_{\lambda s_e} c\sigma^i d \quad \rightarrow \quad P \wedge a =^?_{\lambda s_e} c \wedge b =^?_{\lambda s_e} d$
(σ-Fail)	$P \wedge a\sigma^i b =^?_{\lambda s_e} c\sigma^j d \quad \rightarrow \quad \mathbb{F}$ if $i \neq j$ and $a\sigma^i b =^?_{\lambda s_e} c\sigma^j d$ is not *flex-flex*
(Dec-φ)	$P \wedge \varphi^i_k a =^?_{\lambda s_e} \varphi^i_k b \quad \rightarrow \quad P \wedge a =^?_{\lambda s_e} b$
(φ-Fail)	$P \wedge \varphi^i_k a =^?_{\lambda s_e} \varphi^j_l b \quad \rightarrow \quad \mathbb{F}$ if $i \neq j$ or $k \neq l$ and $\varphi^i_k a =^?_{\lambda s_e} \varphi^j_l b$ is not *flex-flex*
(Exp-λ)	$P \quad \rightarrow \quad \exists(Y : A.\Gamma \vdash B), P \wedge X =^?_{\lambda s_e} \lambda_A.Y$ if $(X : \Gamma \vdash A \rightarrow B) \in \mathcal{T}var(P), Y \notin \mathcal{T}var(P)$, and X is a unsolved variable
(Exp-App)	$P \wedge \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(X, a_1, \dots, a_p) =^?_{\lambda s_e} (\mathtt{m}\ b_1 \dots b_q) \quad \rightarrow \quad P \wedge \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(X, a_1, \dots, a_p) =^?_{\lambda s_e} (\mathtt{m}\ b_1 \dots b_q) \ \wedge \ \bigvee_{r \in R_p \cup R_i} \exists H_1, \dots, H_k, X =^?_{\lambda s_e} (\mathtt{r}\ H_1 \dots H_k)$ if $\psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(X, a_1, \dots, a_p)$ is the skeleton of a λs_e-normal term and X has an atomic type and is not solved where $H_1, \dots, H_k$ are variables of appropriate types, not occurring in P, with the environments $\Gamma_{H_i} = \Gamma_X$, R_p is the subset of $\{i_1, \dots, i_p\}$ of superscripts of the σ operator such that $(\mathtt{r}\ H_1 \dots H_k)$ has the right type, $R_i = \bigcup_{k=0}^{p}$ if $i_k \geq m+p-k-\sum_{l=k+1}^{p} j_l > i_{k+1}$ then $\{m+p-k-\sum_{l=k+1}^{p} j_l\}$ else $\emptyset$, where $i_0 = \infty, i_{p+1} = 0$
(Replace)	$P \wedge X =^?_{\lambda s_e} a \quad \rightarrow \quad \{X/a\}P \wedge X =^?_{\lambda s_e} a$ if $X \in \mathcal{T}var(P), X \notin \mathcal{T}var(a)$ and $a \in \mathcal{X} \Rightarrow a \in \mathcal{T}var(P)$
(Normalize)	$P \wedge a =^?_{\lambda s_e} b \quad \rightarrow \quad P \wedge a' =^?_{\lambda s_e} b'$ if a or b is not in lnf where a' is the lnf of a if a is not a solved variable and a otherwise. b' is defined from b identically

Definition 32. *The set in Table 3 defines the* λs_e**-unification rules** *for the typed* λs_e*-unification problems.*

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

Equations of the form $(\mathtt{n}\ a_1 \dots a_p) =^?_{\lambda s_e} (\mathtt{m}\ b_1 \dots b_q)$ are transformed by the rules *Dec-App* and *App-Fail* into the empty disjunction when $\mathtt{n} \neq \mathtt{m}$, as it has no solution, or into the conjunction $\bigwedge_{i=1..p} a_i =^?_{\lambda s_e} b_i$, when $\mathtt{n} = \mathtt{m}$. Remember that by terms of the form $(\mathtt{n}\ a_1 \dots a_p)$ we also mean those where $\mathtt{n}$ is omitted or $p = 0$. Analogously, the rules *Dec-σ* and *Dec-φ* decompose equations with leading operators σ and φ. But, the corresponding rules *σ-Fail* and *φ-Fail* should omit *flex-flex* equations as the Example 33 shows.

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

Then $(\lambda.(\lambda.(X\ \mathtt{2})\ \mathtt{1})\ \ Y) \rightarrow^* ((X\sigma^2 Y)\sigma^1(\varphi^1_0 Y)\ \ \varphi^1_0 Y)$ and $(\lambda.(Z\ \mathtt{1})\ \ U) \rightarrow^* (Z\sigma^1 U\ \ \varphi^1_0 U)$. Thus applying the rule *Normalize* to the original equation we obtain $((X\sigma^2 Y)\sigma^1(\varphi^1_0 Y)\ \ \varphi^1_0 Y) =^?_{\lambda s_e} (Z\sigma^1 U\ \ \varphi^1_0 U)$ which can be decomposed into $(X\sigma^2 Y)\sigma^1(\varphi^1_0 Y) =^?_{\lambda s_e} Z\sigma^1 U \quad \wedge \quad \varphi^1_0 Y =^?_{\lambda s_e} \varphi^1_0 U$ and subsequently into $(X\sigma^2 Y) =^?_{\lambda s_e} Z \quad \wedge \quad \varphi^1_0 Y =^?_{\lambda s_e} U \quad \wedge \quad Y =^?_{\lambda s_e} U$.

Since $\forall n \in \mathbb{N}$, $\varphi^1_0 \mathtt{n} \rightarrow \mathtt{n}$, the equation $\varphi^1_0 Y =^?_{\lambda s_e} U$ 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 $\{X/\mathtt{n}\}$; thus if $n > 2$, $\{Z/\mathtt{n}-\mathtt{1}\}$ else if $n = 2$, $\{Z/\varphi^2_0 Y\}$ else $\{Z/\mathtt{1}\}$.

Analogously, by normalization and decomposition with the $\lambda\sigma$-unification rules we have:
$(\lambda.(\lambda.(X\ \ 2)\ \ 1)\ \ Y) =^?_{\lambda\sigma} (\lambda.(Z\ \ 1)\ \ U) \to_{Normalize} (X[Y.Y.id]\ \ Y) =^?_{\lambda\sigma} (Z[U.id]\ \ U)$, which can be decomposed into $X[Y.Y.id] =^?_{\lambda\sigma} Z[U.id] \wedge Y =^?_{\lambda\sigma} U$. A further step of replacement gives the corresponding *flex-flex* equation of the $\lambda\sigma$-calculus $X[Y.Y.id] =^?_{\lambda\sigma} Z[Y.id]$. •

In $\lambda\sigma$-HOU, the rule *Exp-App* advances towards solutions to equations of the form $X[a_1 \dots a_p.\ \uparrow^n] =^?_{\lambda s_e} (\mathtt{m}\ b_1 \dots b_q)$ where X is an unsolved variable of an atomic type. The λs_e-unification rule *Exp-App* has the analogous role for λs_e-unification problems. Use of λs_e-normal forms in *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 before applying *Exp-App*, λs_e-unification problems are normalized.

Example 34 From the unification problem $\lambda.(\lambda.(Y\ \ 1)\ \ \lambda.(X\ \ 1)) =^? \lambda.(\lambda.V\ \ \lambda.W)$ we reach the equations: $(Y[\lambda.(X\ \ 1).id]\ \ \lambda.(X\ \ 1)) =^?_{\lambda\sigma} V[\lambda.W.id]$ and $(Y\sigma^1\lambda.(X\ \ 1)\ \ \lambda.(\varphi^1_1\ \ 1)) =^?_{\lambda s_e} V\sigma^1\lambda.W$. After applying the corresponding *Exp-App* rules, with $V =^?_{\lambda\sigma} (V_1\ \ V_2)$ and $V =^?_{\lambda s_e} (V_1\ \ V_2)$, additional equations appear: $\lambda.(X\ \ 1) =^?_{\lambda\sigma} V_2[\lambda.(X\ \ 1).id]$ and $\lambda.(\varphi^1_1 X\ \ 1) =^?_{\lambda s_e} V_2\sigma^1\lambda.(X\ \ 1)$. Solutions result by selecting the case $V_2 =^?_{\lambda\sigma} \mathtt{1}$ or correspondingly $V_2 =^?_{\lambda s_e} \mathtt{1}$. •

Definition 35. *A unification system P is a λs_e-***solved form** *if it is a conjunction of non trivial equations of the following forms:*

(Solved)	$X =^?_{\lambda s_e} a$, *where the variable X does not occur anywhere else in P and a is in lnf. Such an equation and variable are said to be* **solved** *in P.*
(Flex-Flex)	*non solved equations between long normal terms whose root operator is σ or φ which can be represented as equations between their skeleton:* $\psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(X, a_1, \dots, a_p) =^?_{\lambda s_e} \psi^{l_q}_{k_q} \dots \psi^{l_1}_{k_1}(Y, b_1, \dots, b_q)$.

Remark 36 Consider a λs_e-normal term t whose root operator is either σ or φ and with skeleton of the form $sk(t) = \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(X, a_1, \dots, a_p)$. Then by binding X with $\mathtt{n}$, $n > i_1$, one obtains the normal form $t \to^* \mathtt{n} + \sum_{k=1}^{p} \mathtt{j}_k - \mathtt{p}$. This is a direct consequence of lemma 216. •

The rest of this section lists relevant properties of the λs_e-unification rules. For proofs, see [ARK00].

Lemma 37. *Any λs_e-solved form has λs_e-unifiers.*

Lemma 38 (Well-typedness). *Deduction by the λs_e-unification rules of a well typed equation gives rise only to well typed equations, $\mathbb{T}$ and $\mathbb{F}$.*

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

Definition 310. *Let P and P' be λs_e-unification problems, let "rule" denote the name of a λs_e-unification rule and "$\to^{rule}$" its corresponding deduction relation over unification problems. By* **correctness** *of rule we understand: $P \to^{rule} P'$ implies $\mathcal{U}_{\lambda s_e}(P') \subseteq \mathcal{U}_{\lambda s_e}(P)$. By* **completeness** *of rule we understand: $P \to^{rule} P'$ implies $\mathcal{U}_{\lambda s_e}(P) \subseteq \mathcal{U}_{\lambda s_e}(P')$*

Theorem 311 (Correctness and Completeness). *The λs_e-unification rules are correct and complete.*

Table 4. Unification replace strategy

Normalize **or** *Dec-*λ **or** *Dec-App* **or** *App-Fail* **or** *Dec-*σ **or** σ*-Fail* **or**
Dec-φ **or** φ*-Fail* **or** *Exp-*λ; *Replace* **or** *Exp-App* ; *Replace*

4 A unification strategy

λs_e-unification rules should be applied following some strategy that avoids non termination of the unification process. Observe, in particular, that the rule *Exp-*λ 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-*λ 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 λs_e-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-*λ; *Replace*) and (*Exp-App*; *Replace*) are denoted by *Exp-*$\boldsymbol{\lambda}$*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 $P = \langle Q, R\rangle$. 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 θ resulting from the unification algorithm. For a solved system R consisting of only solved equations, $Subst(R)$ denotes the canonical grafting associated to R. For example if $R = (X =^?_{\lambda s_e} a)$ then $Subst(R) = \{X/a\}$.

Take in the rest of this section a λs_e-normalized grafting solution θ of a unification problem P.

Definition 41. *For a system of equations $P = \langle Q, R\rangle$ and a λs_e-normalized grafting θ, which is a solution of P, we define the* **UnifStrat** *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. $\langle Q, R, \theta\rangle \to^{Normalize} \langle Q', R', \theta\rangle$, *where Q' and R' are the normalized forms of Q and R as defined in the λs_e-unification.*
2. $\langle Q \wedge \lambda_A.a =^?_{\lambda s_e} \lambda_A.b, R, \theta\rangle \to^{\mathbf{Dec-\lambda}} \langle Q', R', \theta\rangle$, *where $Q' = Q \wedge a =^?_{\lambda s_e} b$ and $R' = R$ when $a =^?_{\lambda s_e} b$ is not solved with respect to $Q \wedge R$ or $Q' = Q$ and $R' = R \wedge a =^?_{\lambda s_e} b$ when $a =^?_{\lambda s_e} b$ is solved.*
3. $\langle Q \wedge (\mathtt{n}\ a_1 \dots a_p) =^?_{\lambda s_e} (\mathtt{n}\ b_1 \dots b_p), R, \theta\rangle \to^{\mathbf{Dec-App}} \langle Q', R', \theta\rangle$, *where Q' consists in Q and the unsolved equations (with respect to $Q \wedge R$) in $\bigwedge_{i=1..p} a_i =^?_{\lambda s_e} b_i$ and R' consists in R and the solved equations in $\bigwedge_{i=1..p} a_i =^?_{\lambda s_e} b_i$.*
4. $\langle Q, R, \theta\rangle \to^{\mathbf{Dec-\sigma}} \langle Q', R', \theta\rangle$, *where Q' consists in Q and the unsolved equations (with respect to $Q \wedge R$) in $a =^?_{\lambda s_e} c \wedge b =^?_{\lambda s_e} d$ and R' consists in R and the solved equations in $a =^?_{\lambda s_e} c \wedge b =^?_{\lambda s_e} d$.*
5. $\langle Q, R, \theta\rangle \to^{\mathbf{Dec-\varphi}} \langle Q', R', \theta\rangle$, *where if $a =^?_{\lambda s_e} b$ is unsolved with respect to $Q \wedge R$, $Q' = Q \wedge a =^?_{\lambda s_e} b$ and $R' = R$ else $Q' = Q$ and $R' = R \wedge a =^?_{\lambda s_e} b$.*
6. $\langle Q \wedge X =^?_{\lambda s_e} b, R, \theta\rangle \to^{\mathbf{Replace}} \langle \{X/b\}Q, R \wedge X =^?_{\lambda s_e} b, \theta\rangle$
7. $\langle Q, R, \theta\rangle \to^{\mathbf{Exp-\lambda R}} \langle \{X/\lambda_A.Y\}Q, R \wedge X =^?_{\lambda s_e} \lambda_A.Y, \theta \setminus \{X/\lambda_A.a\} \cup \{Y/a\}$, *when Exp-*$\lambda$ *applies on $Q \wedge R$.*
8. $\langle Q \wedge \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(X, a_1, \dots, a_p) =^?_{\lambda s_e} (\mathtt{m}\ b_1 \dots b_q), R, \theta\rangle \to^{\mathbf{Exp-AppR}}$
$$\langle \{X/(\mathtt{r}\ H_1 \dots H_k)\}(Q \wedge \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(X, a_1, \dots, a_p) =^?_{\lambda s_e} (\mathtt{m}\ b_1 \dots b_q)),$$
$$R \wedge X =^?_{\lambda s_e} (\mathtt{r}\ H_1 \dots H_k), \theta \setminus \{X/(\mathtt{r}\ c_1 \dots c_k) \cup \{H_i/c_i\}\rangle$$
for one of the $r \in R_p \cup R_i$, when Exp-App applies on $Q \wedge R$.

Lemma 42 (*UnifStrat* is well defined). *The transformations Exp-*λ*R and Exp-AppR are well defined.*

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

First, observe that since *Exp-*λ preserves the solutions, θ is also a λs_e-solution of the equation $X =^?_{\lambda s_e} \lambda_A.Y$. This together with the assumption that θ is λs_e-normalized, implies that the instantiation of X by θ is of the form $\{X/\lambda_A.a\}$. Hence the transformation *Exp-*λ*R* is well defined.

Second, since *Exp-App* preserves the solutions, θ is λs_e-solution of the equation $X =^?_{\lambda s_e} (\mathtt{r}\ H_1 \dots H_k)$. Thus $\theta(X) = (\mathtt{r}\ c_1 \dots c_k)$, for some c_i, $i = 1, \dots, k$. Consequently, the transformation *Exp-AppR* is well defined too. □

Lemma 43 (Finiteness of *UnifStrat*). *For a system $\langle Q, R\rangle$ having for solution a λs_e-normalized grafting θ, there is no infinite derivation issued from $\langle Q, R, \theta\rangle$, using the UnifStrat transformations.*

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

First, we prove that there are no infinite sequences of transformation applications involving the transformations *Normalize*, *Replace*, *Dec-*λ, *Dec-App*, *Dec-*σ and *Dec-*φ. We define a complexity measure, τ, of a system $P = \langle Q, R\rangle$ by: $\tau(P) = \langle |var(Q)|, \{\kappa_i, max(|a_i|, |b_i|)\}_{a_i =^?_{\lambda s_e} b_i \in Q}\rangle$ where κ_i is the length of the shortest λs_e-normalizing derivation of $a_i =^?_{\lambda s_e} b_i$. 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 κ_i. *Replace* decreases the number of unsolved variables. *Dec-*λ, *Dec-App*, *Dec-*σ and *Dec-*φ never increase κ_i (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 *Exp-*λ *R* and *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-*λ, *Dec-App*, *Dec-*σ and *Dec-*φ 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 *Exp-*λ *R* and *Exp-AppR* always decrease the size of the current grafting θ. Consequently the application of *UnifStrat* is terminating. □

Lemma 44 (Preservation of solutions). *If θ is a λs_e-solution of system Q and if $\langle Q, R, \theta\rangle \to^r \langle Q', R', \theta'\rangle$ then θ' is a λs_e-solution of Q' and $\theta \circ Subst(R) =^{var(Q,R)}_{\lambda s_e} \theta' \circ Subst(R')$.*

Proof. For all the rules except *Exp-*λ *R* and *Exp-AppR*, we have $\theta' = \theta$. Additionally, since the rules *Exp-*λ and *Exp-App* preserve solutions, we have that θ' is a solution of Q'. Observe that the equality modulo λs_e is introduced by possible normalization steps.

As the proofs for *Exp-*λ*R* and *Exp-AppR* are similar, we only do one. By the definition of *Exp-*λ*R*, θ' is a λs_e-solution of Q'. Let Z be a variable in $var(Q, R)$ then either $Z = X$ or $Z \neq X$. In the first case, θ satisfies $\theta(X) = (\theta \circ Subst(R))(X) = \theta(X) = \lambda_A.a$, and $(\theta' \circ Subst(R'))(X) = \theta'(\lambda_A.Y) = \lambda_A.a$. In the second case, both graftings give the same image for Z. □

Lemma 45 (Construction of solutions). *Let $\langle Q_0, R_0, \theta_0\rangle \to \langle Q_1, R_1, \theta_1\rangle \to \cdots \to \langle Q_n, R_n, \theta_n\rangle$ be a finite derivation applying transformations of UnifStrat starting from the problem $P_0 = \langle Q_0, R_0\rangle$ and the λs_e-normalized solution θ_0. Then, $\theta_0 =^{var(P_n)}_{\lambda s_e} \theta_n \circ Subst(R_n)$, where θ_n is a solution of the solved form Q_n.*

Proof. Observe that $\theta_0 = \theta_0 \circ Subst(R_0)$. 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: $\theta_0 \circ Subst(R_0) =^{var(P_0)}_{\lambda s_e} \theta_1 \circ Subst(R_1) =^{var(P_1)}_{\lambda s_e} \cdots =^{var(P_{n-1})}_{\lambda s_e} \theta_n \circ Subst(R_n)$. By Lemma 39, $Q_n \wedge R_n$ should be a solved form. Moreover, $var(P_0) \supseteq var(P_1) \supseteq \cdots \supseteq var(P_n)$ since the set of variables of the unification problems could only decrease due to the *Normalize* rule. Then we have $\theta_0 =^{var(P_n)}_{\lambda s_e} \theta_n \circ Subst(R_n)$. □

Theorem 46 (Completeness of *UnifStrat*). *The λs_e-unification rules describe a correct and complete λs_e-unification procedure in the sense that, given a λs_e-unification problem P:*

if the λs_e-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 λs_e-unifiable and a solution to P is the solution constructed in Lemma 37 for a solved constitutive system,
if P has a unifier θ then the strategy UnifStrat 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. □

5 HOU in the pure λ-calculus

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

Observe firstly that unifying two terms a and b in the λ-calculus consists in finding a *substitution* θ such that $\theta(a) =_{\beta\eta} \theta(b)$. 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 $\lambda.X =^?_{\beta\eta} \lambda.2$ (where $=_{\beta\eta}$ denotes the congruence generated by the β- and η-rules on $\Lambda_{dB}(\mathcal{X})$) is not a term $t = \theta X$ such that $\lambda.t =^?_{\beta\eta} \lambda.2$ but a term $t = \theta X$ such that $\theta(\lambda.X) = \lambda.\theta^+(X) = \lambda.2$ as $(\lambda.X)\{X/t\} = \lambda.X\{X/t^+\} = \lambda.t^+$ and not $\lambda.t$. This observation can be extended to any unifier and by translating appropriately λ-terms $a, b \in \Lambda_{dB}(\mathcal{X})$, the HOU problem $a =^?_{\beta\eta} b$ can be reduced to equational unification. [DHK00] presents a translation called *pre-cooking* from $\Lambda_{dB}(\mathcal{X})$ terms into the $\lambda\sigma$-calculus such that searching for solutions of the corresponding $\lambda\sigma$-unification problem corresponds to searching for solutions of the HOU problem $a =^?_{\beta\eta} b$. In the following examples, we illustrate informally the analogous situation in the λs_e-calculus.

Example 51 Consider the higher order unification problem $\lambda.(X\ \ 2) =^?_{\beta\eta} \lambda.2$, where 2 and X are of type A and $A \to A$, respectively. Observe that applying a substitution solution θ to the $\Lambda_{dB}(\mathcal{X})$-term $\lambda.(X\ \ 2)$ gives $\theta(\lambda.(X\ \ 2)) = \lambda.(\theta^+(X)\ \ 2)$. Then in the λs_e-calculus we are searching for a grafting θ' such that $\theta'(\lambda.(\varphi_0^2(X)\ \ 2)) =_{\lambda s_e} \lambda.2$. Correspondingly, in the $\lambda\sigma$-calculus, $\lambda.(X\ \ 2)$ is translated or pre-cooked into $\lambda.(X[\uparrow]\ \ 2)$. Observe that this correspondence results from lemma 28. Then we should search for unifiers for the problem $\lambda.(\varphi_0^2(X)\ \ 2) =^?_{\lambda s_e} \lambda.2$.

Now we apply λs_e-unification rules to the problem $\lambda.(\varphi_0^2(X)\ \ 2) =^?_{\lambda s_e} \lambda.2$. By applying *Dec-*$\lambda$ and *Exp-*λ we get $(\varphi_0^2(X)\ \ 2) =^?_{\lambda s_e} 2$ and subsequently $\exists Y(\varphi_0^2(X)\ \ 2) =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y$. Then by applying *Replace* and *Normalize* we obtain $\exists Y(\varphi_0^2(\lambda.Y)\ \ 2) =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y$ and $\exists Y(\varphi_1^2 Y)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y$. Now, by applying rule *Exp-app* we obtain
$(\exists Y(\varphi_1^2 Y)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y) \wedge (Y =^?_{\lambda s_e} 1 \vee Y =^?_{\lambda s_e} 2)$ which by *Replace* gives
$((\varphi_1^2 1)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.1) \vee ((\varphi_1^2 2)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.2)$ and, finally, by *Normalize*
$(2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.1) \vee (2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.2)$. In this way substitution solutions $\{X/\lambda.1\}$ and $\{X/\lambda.2\}$ are found.

To complete the analysis note that by Definitions 22, 23 and β-reduction in $\Lambda_{dB}(\mathcal{X})$ we have:
$\{X/\lambda.1\}(\lambda.(X\ \ 2)) = \lambda.(\{X/(\lambda.1)^+\}(X)\ \ 2) = \lambda.(\lambda.1^{+1}\ \ 2) = \lambda.(\lambda.1\ \ 2) =_\beta \lambda.2$ and
$\{X/\lambda.2\}(\lambda.(X\ \ 2)) = \lambda.(\{X/(\lambda.2)^+\}(X)\ \ 2) = \lambda.(\lambda.2^{+1}\ \ 2) = \lambda.(\lambda.3\ \ 2) =_\beta \lambda.2$.
Observe that the last application of β-reduction is as follows: $(\lambda.3\ \ 2) =_\beta \{1/2\}(3) = 2$. •

In general, before the unification process, a λ-term a should be translated into the λs_e-term a' resulting by simultaneously replacing each occurrence of a meta-variable X at position i in a with $\varphi_0^{k+1}X$, where k is the number of abstractors 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 $\mathtt{n}$ into $1[\uparrow^{n-1}]$ and all occurrences of meta-variables X into $X[\uparrow^k]$, 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 $F(f(a)) =^? f(F(a))$. In $\Lambda_{dB}(\mathcal{X})$ it can be seen as $(X\ \ (2\ \ 1)) =^?_{\beta\eta} (2\ \ (X\ \ 1))$, where both X and 2 are of type $A \to A$ and 1 is of type A. Since there are no abstractors at the terms of the equational problem, the equation remains unchanged: $(X\ \ (2\ \ 1)) =^?_{\lambda s_e} (2\ \ (X\ \ 1))$.

For simplicity we omit existential quantifiers. After one application of *Exp-*λ and another of *Replace* we get $(\lambda.Y\ \ (2\ \ 1)) =^?_{\lambda s_e} (2\ \ (\lambda.Y\ \ 1)) \wedge X =^?_{\lambda s_e} \lambda.Y$ where Y is of type A. Applying *Normalize* we obtain $Y\sigma^1(2\ \ 1) =^?_{\lambda s_e} (2\ \ Y\sigma^1 1) \wedge X =^?_{\lambda s_e} \lambda.Y$ And by one application of *Exp-App* we get $Y\sigma^1(2\ \ 1) =^?_{\lambda s_e} (2\ \ Y\sigma^1 1) \wedge X =^?_{\lambda s_e} \lambda.Y \wedge (Y =^?_{\lambda s_e} 1 \vee Y =^?_{\lambda s_e} (3\ \ H_1))$.

Note that other possible cases do not produce solved forms. By *Replace* and *Normalize* we get:
$((2\ \ 1) =^?_{\lambda s_e} (2\ \ 1) \wedge X =^?_{\lambda s_e} \lambda.1) \vee ((2\ \ H_1\sigma^1(2\ \ 1)) =^?_{\lambda s_e} (2\ \ (2\ \ H_1\sigma^1 1)) \wedge X =^?_{\lambda s_e} \lambda.(3\ \ H_1))$, from which we have the first solved system corresponding to the identity solution: $\{X/\lambda.1\}$.

Subsequently, other solutions can be obtained from the equational system

$$(2\ \ H_1\sigma^1(2\ \ 1)) =^?_{\lambda s_e} (2\ \ (2\ \ H_1\sigma^1 1)) \wedge X =^?_{\lambda s_e} \lambda.(3\ \ H_1)$$

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

$$H_1\sigma^1(2\ \ 1) =^?_{\lambda s_e} (2\ \ H_1\sigma^1 1) \wedge X =^?_{\lambda s_e} \lambda.(3\ \ H_1) \wedge (H_1 =^?_{\lambda s_e} 1 \vee H_1 =^?_{\lambda s_e} (3\ \ H_2))$$

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

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

Now we can define our pre-cooking translation.

Definition 53 (Pre-cooking). *Let $a \in \Lambda_{dB}(\mathcal{X})$ such that $\Gamma \vdash_{\Lambda_{dB}(\mathcal{X})} a : T$. To every variable X of type A occurring at a we associate the same type and context Γ in the λs_e-calculus. The pre-cooking of a from $\Lambda_{dB}(\mathcal{X})$ to the λs_e-calculus is defined by $a_{pc} = PC(a, 0)$ 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. $PC(\mathtt{k}, n) = \mathtt{k}$
4. $PC(X, n) = \text{if } n = 0 \text{ then } X \text{ else } \varphi_0^{n+1} X$

Lemma 54. *If $\Gamma \vdash_{\Lambda_{dB}(\mathcal{X})} a : T$, then $\Gamma \vdash_{\lambda s_e} a_{pc} : T$.*

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

Initially, observe that cases $a = \mathtt{k}$ and $a = (a_1\ \ a_2)$ are simple. Afterwards, suppose that $a = \lambda_B.b$. Then $T = B \to C$ and $B, A_1 \ldots A_n, \Gamma \vdash_{\Lambda_{dB}(\mathcal{X})} b : C$. Thus $B, A_1 \ldots A_n, \Gamma \vdash_{\lambda s_e} PC(b, n+1) : C$ and $A_1 \ldots A_n, \Gamma \vdash_{\lambda s_e} PC(\lambda_B.b, n) = \lambda_B.PC(b, n+1) : B \to C$. Finally, for $a = X$ by definition of $\Gamma \vdash_{\Lambda_{dB}(\mathcal{X})} X : T$, $\Gamma \vdash_{\lambda s_e} X : T$ and $A_1 \ldots A_n, \Gamma \vdash_{\lambda s_e} \varphi_0^{n+1}(X) : T$. □

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

Proposition 55 (Semantics of pre-cooking). *Let $a, b_1, \ldots, b_p$ be terms of $\Lambda_{dB}(\mathcal{X})$. We have:*
$(a\{X_1/b_1, \ldots, X_p/b_p\})_{pc} = a_{pc}\{X_1/b_{1_{pc}}, \ldots, X_p/b_{p_{pc}}\}_g$

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

- $\underline{\boldsymbol{a = \lambda.b}}$. $PC((\lambda.b)\{X_j/b_j^{+^i}\}, i) = PC(\lambda.(b\{X_j/b_j^{+^{i+1}}\}), i) = \lambda.(PC(b\{X_j/b_j^{+^{i+1}}\}, i+1))$. By induction hypothesis, the previous expression is equal to $\lambda.(PC(b, i+1)\{X_j/b_{j_{pc}}\}_g) = \lambda.(PC(b, i+1))\{X_j/b_{j_{pc}}\}_g = PC(\lambda.b, i)\{X_j/b_{j_{pc}}\}_g$.
- $\underline{\boldsymbol{a = (a_1\ \ a_2)}}$. Observe that $PC((a_1\ \ a_2)\{X_j/b_j^{+^i}\}, i) = PC((a_1\{X_j/b_j^{+^i}\}\ \ a_2\{X_j/b_j^{+^i}\}), i)$ and this is equal to $(PC(a_1\{X_j/b_j^{+^i}\}, i)\ \ PC(a_2\{X_j/b_j^{+^i}\}, i))$. By applying the induction hypothesis the last expression is equal to $(PC(a_1, i)\{X_j/b_{j_{pc}}\}_g\ \ PC(a_2, i)\{X_j/b_{j_{pc}}\}_g)$. To conclude, the last expression is equal to $(PC(a_1, i)\ \ PC(a_2, i))\{X_j/b_{j_{pc}}\}_g = PC((a_1\ \ a_2), i)\{X_j/b_{j_{pc}}\}_g$.

• $\underline{\boldsymbol{a = \mathtt{n}}}$. $PC(\mathtt{n}\{X_j/b_j^{+^i}\}, i) = PC(\mathtt{n}, i) = \mathtt{n}\{X_j/b_{j_{pc}}\}_g = PC(\mathtt{n}, i)\{X_j/b_{j_{pc}}\}_g$.
• $\underline{\boldsymbol{a = X}}$. 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 $PC(b_j^{+^i}, i) = PC(X_j, i)\{X_j/b_{j_{pc}}\}_g = \varphi_0^{i+1} b_{j_{pc}}$. We will prove the more general fact that $PC(b^{(+k)^i}, i+k) = \varphi_k^{i+1} PC(b, k)$. This is done by induction on the structure of b as follows:
– $\underline{\boldsymbol{b = \lambda.c}}$. $PC((\lambda.c)^{(+k)^i}, i+k) = PC(\lambda.c^{(+(k+1))^i}, i+k)$ since $(\lambda.c)^{(+k)^i} = \lambda.c^{(+(k+1))^i}$. The last expression is equal to $\lambda.PC(c^{(+(k+1))^i}, i+k+1)$ which is equal to $\lambda.\varphi_{k+1}^{i+1} PC(c, k+1)$ by the induction hypothesis. The last expression is equal to $\varphi_k^{i+1}\lambda.PC(c, k+1) = \varphi_k^{i+1} PC(\lambda.c, k)$.
– $\underline{\boldsymbol{b = (b_1 \;\; b_2)}}$. Initially, we have the following: $PC((b_1 \;\; b_2)^{(+k)^i}, i+k) = PC((b_1^{(+k)^i} \;\; b_2^{(+k)^i}), i+k) = (PC(b_1^{(+k)^i}, i+k) \;\; PC(b_2^{(+k)^i}, i+k))$. Then by applying the induction hypothesis this is equal to $(\varphi_k^{i+1} PC(b_1, k) \;\; \varphi_k^{i+1} PC(b_2, k)) = \varphi_k^{i+1} PC((b_1 \;\; b_2), k)$.
– $\underline{\boldsymbol{b = \mathtt{n}}}$. Case $n > k$, $PC(\mathtt{n}^{(+k)^i}, i+k) = PC(\mathtt{n+i}, i+k) = \mathtt{n+i}$ and $\varphi_k^{i+1} PC(\mathtt{n}, k) = \varphi_k^{i+1}\mathtt{n} = \mathtt{n+i}$. Case $n \leq k$, $PC(\mathtt{n}^{(+k)^i}, i+k) = PC(\mathtt{n}, i+k) = \mathtt{n}$ and $\varphi_k^{i+1} PC(\mathtt{n}, k) = \varphi_k^{i+1}\mathtt{n} = \mathtt{n}$.
– $\underline{\boldsymbol{b = X}}$. $PC(X^{(+k)^i}, i+k) = PC(X, i+k) = \varphi_0^{i+k+1} X$ and $\varphi_k^{i+1} PC(X, k) = \varphi_k^{i+1}\varphi_0^{k+1} X = \varphi_0^{i+k+1} X$. □

In contrast with the corresponding proof in [DHK00], where substitution objects are necessary for proving the critical case of $a = X$ (i.e., substitutions of the form $[1...k.\uparrow^{i+k}]$) our proof uses pure term objects by selecting the appropriate super and subscripts for the φ operator (i.e., φ_k^{i+1}).

The following proposition presents necessary facts for relating the existence of solutions for unification problems in the pure λ-calculus and in the λs_e-calculus.

Proposition 56. *Let a and b be terms in $\Lambda_{dB}(\mathcal{X})$. Then*

1. $a \to_\beta b$ implies $a_{pc} \to^*_{\lambda s_e} b_{pc}$. *2. If a is $\beta\eta$-normal then a_{pc} is λs_e-normal.*
3. $a \to_\eta b$ implies $a_{pc} \to_{eta} b_{pc}$. *4. $a =_{\beta\eta} b$ if and only if $a_{pc} =_{\lambda s_e} b_{pc}$*

Proof. First. We will prove the more general fact that $(\lambda^{k+1}.a \;\; b) \to_\beta (\lambda^k.a)\{1/b\}$ implies $((\lambda^{k+1}.a) \;\; b)_{pc} \to^*_{\lambda s_e} ((\lambda^k.a)\{1/b\})_{pc}$. This is done by induction on a for all k. The case $k = 0$ corresponds to our case of interest. Initially, notice that $((\lambda^{k+1}.a) \;\; b)_{pc} = ((\lambda^{k+1}.PC(a, k+1)) \;\; b_{pc}) \to_{\lambda s_e} \lambda^k.(PC(a, k+1)\sigma^{k+1} b_{pc})$ and that $(\lambda^k.a)\{1/b\} = \lambda^k.(a\{k+1/b^{+^k}\})$. Thus $((\lambda^k.a)\{1/b\})_{pc} = \lambda^k.PC(a\{k+1/b^{+^k}\}, k)$.
• $\underline{\boldsymbol{a = \mathtt{n}}}$. The interesting case occurs when $n = k+1$. In this case, $\lambda^k.(\mathtt{k+1}\{k+1/b^{+^k}\}) = \lambda^k.b^{+^k}$. Additionally, we have $((\lambda^{k+1}.\mathtt{n}) \;\; b)_{pc} = (\lambda^{k+1}.\mathtt{n} \;\; b_{pc}) \to^*_{\lambda s_e} \lambda^k.(\mathtt{n}\sigma^{k+1} b_{pc}) = \lambda^k.\varphi_0^{k+1} b_{pc}$ and $(\lambda^k.b^{+^k})_{pc} = \lambda^k.PC(b^{+^k}, k)$. Then we have to prove that $\lambda^k.\varphi_0^{k+1} b_{pc} \to^*_{\lambda s_e} \lambda^k.PC(b^{+^k}, k)$. We prove the more general fact that $\lambda^{k+i}.\varphi_i^{k+1} PC(b, i) \to^*_{\lambda s_e} \lambda^{k+i}.PC(b^{(+i)^k}, k+i)$. This is done by structural induction on b for all $k > 0$ and $i \geq 0$.
– $\underline{\boldsymbol{b = \mathtt{m}}}$. On one side, $\lambda^{k+i}.\varphi_i^{k+1} PC(\mathtt{m}, i) = \lambda^{k+i}.\varphi_i^{k+1}\mathtt{m} = \begin{cases} \lambda^{k+i}.\mathtt{m+k}, & \text{if } n > i; \\ \lambda^{k+i}.\mathtt{m}, & \text{if } m \leq i. \end{cases}$
On the other side, by definition of i-lift, $\lambda^{k+i}.PC(\mathtt{m}^{(+i)^k}, k+i) = \begin{cases} \lambda^{k+i}.\mathtt{m+k}, & \text{if } n > i; \\ \lambda^{k+i}.\mathtt{m}, & \text{if } m \leq i. \end{cases}$
– $\underline{\boldsymbol{b = X}}$. We have two cases: either $i = 0$ or $i > 0$. In the first case we have $\lambda^k.\varphi_0^{k+1} PC(X, 0) = \lambda^k.\varphi_0^{k+1} X$ and $\lambda^k.PC(X^{+^k}, k) = \lambda^k.PC(X, k) = \lambda^k.\varphi_0^{k+1} X$. If $i > 0$, $\lambda^{k+i}.\varphi_i^{k+1} PC(X, i) = \lambda^{k+i}.\varphi_i^{k+1}\varphi_0^{i+1} X \to_{\lambda s_e} \lambda^{k+i}.\varphi_0^{k+i+1} X$ and $\lambda^{k+i}.PC(X, k+i) = \lambda^{k+i}.\varphi_0^{k+i+1} X$.
– $\underline{\boldsymbol{b = \lambda.c}}$. We have that $\lambda^{k+i}.\varphi_i^{k+1} PC(\lambda.c, i) = \lambda^{k+i}.\varphi_i^{k+1}\lambda.PC(c, i+1) \to_{\lambda s_e} \lambda^{k+i}.\lambda.(\varphi_{i+1}^{k+1} PC(c, i+1))$ $= \lambda^{k+i+1}.\varphi_{i+1}^{k+1} PC(c, i+1)$ that λs_e-derives into $\lambda^{k+i+1}.PC(b^{(+i+1)^k}, k+i+1)$ by induction hypothesis.
– $\underline{\boldsymbol{b = (b_1 \;\; b_2)}}$. Using the hypotheses that $\lambda^{k+i}.\varphi_i^{k+1} PC(b_j, i) \to^*_{\lambda s_e} \lambda^{k+i}.PC(b_j^{(+i)^k}, k+i)$, for $j = 1, 2$, we obtain the desired conclusion.
• $\underline{\boldsymbol{a = X}}$. Observe that $\lambda^k.(PC(X, k+1)\sigma^{k+1} b_{pc}) = \lambda^k.(\varphi_0^{k+2} X)\sigma^{k+1} b_{pc}) \to_{\lambda s_e} \lambda^k.\varphi_0^{k+1} X$ and that $\lambda^k.PC(X\{k+1/b^{+^k}\}, k) = \lambda^k.PC(X, k) = \lambda^k.\varphi_0^{k+1} X$.
• $\underline{\boldsymbol{a = \lambda.c}}$. Straightforwardly by applying the induction hypothesis on the more simple term c for $k+1$: $((\lambda^{k+2}.c) \;\; b)_{pc} \to^*_{\lambda s_e} ((\lambda^{k+1}.c)\{1/b\})_{pc}$.
• $\underline{\boldsymbol{a = (a_1 \;\; a_2)}}$. By induction.

Second. We prove the more general fact that if a is $\beta\eta$-normal then for all k, $PC(a,k)$ is λs_e-normal. This is done by structural induction on the structure of a for all k, as follows.

- $\underline{\boldsymbol{a = \mathtt{n}}}$. Obvious.
- $\underline{\boldsymbol{a = X}}$. $PC(X,k) = \varphi_0^{k+1}X$ that is λs_e-normal.
- $\underline{\boldsymbol{a = \lambda.b}}$. $PC(\lambda.b,k) = \lambda.PC(b,k+1)$ and since b should be $\beta\eta$-normal, $PC(b,k+1)$ is λs_e-normal. Then $\lambda.PC(b,k+1)$ is λs_e-normal too.
- $\underline{\boldsymbol{a = (b\ \ c)}}$. Since b and c are $\beta\eta$-normals then $(PC(a,k)\ \ PC(b,k))$ is λs_e-normal.

Third. We prove the more general fact that if $\lambda.(\lambda^k.a\ \ 1) \to_\eta d$ then $(\lambda.(\lambda^k.a\ \ 1))_{pc} \to_{eta} d_{pc}$. Observe that d should be of the form $\lambda^k.c$ and such that $(\lambda^k.c)^+ = \lambda^k.c^{+k} = \lambda^k.a$. Then $c^{+k} = a$. 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.

- $\underline{\boldsymbol{a = \mathtt{n}}}$. Firstly, notice that $\lambda.(\lambda^k.\mathtt{n}\ \ 1) \to_\eta \lambda^k.\mathtt{m}$ whenever $\lambda^k.\mathtt{n} = (\lambda^k.\mathtt{m})^+ = \lambda^k.(\mathtt{m}^{+k}) =$ if $m \leq k$ then $\lambda^k.\mathtt{m}$ else $\lambda^k.\mathtt{m}+1$. Then $m = n$ if $n \leq k$ and $m = n-1$ if $n-1 > k$. Observe that $n \neq k+1$. Thus, $PC(\lambda.(\lambda^k.\mathtt{n}\ \ 1),0) = \lambda.(\lambda^k.PC(\mathtt{n},k+1)\ \ 1) = \lambda.(\lambda^k.\mathtt{n}\ \ 1) \to_{eta}$ if $n \leq k$ then $\lambda^k.\mathtt{n}$ else if $n-1 > k$ then $\lambda^k.\mathtt{n}-1$, since $\varphi_0^2\lambda^k.\mathtt{m} = \lambda^k.\varphi_k^2\mathtt{m} =$ if $m \leq k$ then $\lambda^k.\mathtt{m}$ else if $m > k$ then $\lambda^k.\mathtt{m}+1$.
- $\underline{\boldsymbol{a = X}}$. Firstly, notice that $\lambda.(\lambda^k.X\ \ 1) \to_\eta \lambda^k.X$ since $(\lambda^k.X)^+ = \lambda^k.X^{+k} = \lambda^k.X$. Afterwards, observe that $PC(\lambda.(\lambda^k.X\ \ 1),0) = \lambda.(\lambda^k.PC(X,k+1)\ \ PC(1,1)) = \lambda.(\lambda^k.\varphi_0^{k+2}X\ \ 1) \to_{eta} \lambda^k.\varphi_0^{k+1}X$, since $\varphi_0^2(\lambda^k.\varphi_0^{k+1}X) =_{s_e} \lambda^k.\varphi_k^2\varphi_0^{k+1}X =_{s_e} \lambda^k.\varphi_0^{k+2}X$. Finally, we have that $PC(\lambda^k.X,0) = \lambda^k.PC(X,k) = \lambda^k.\varphi_0^{k+1}X$.
- $\underline{\boldsymbol{a = \lambda.b}}$. Suppose, $\lambda.(\lambda^k.\lambda.a\ \ 1) \to_\eta \lambda^k.d$. Then $\lambda^k.\lambda.a = (\lambda^k.d)^+ = \lambda^k.d^{+k}$ and $\lambda.a = d^{+k}$. Then $PC(\lambda.(\lambda^k.\lambda.a\ \ 1),0) = \lambda.(\lambda^k.PC(\lambda.a,k+1)\ \ 1) \to_{eta} \lambda^k.PC(d,k) = PC(\lambda^{k+1}.c,0)$, for some c.
- $\underline{\boldsymbol{a = (b\ \ c)}}$. Notice that $\lambda.(\lambda^k.(a_1\ \ a_2)\ \ 1) \to_\eta \lambda^k.(c_1\ \ c_2)$ if $\lambda^k.(a_1\ \ a_2) = (\lambda^k.(c_1\ \ c_2))^+ = \lambda^k.(c_1^{+k}\ \ c_2^{+k})$. It should hold that $c_i^{+k} = a_i$ for $i = 1,2$. Then $\lambda.(\lambda^k.a_i\ \ 1) \to_\eta \lambda^k.c_i$, for $i = 1,2$. Now, $PC(\lambda.(\lambda^k.(a_1\ \ a_2)\ \ 1),0) = \lambda.(\lambda^k.(PC(a_1,k+1)\ \ PC(a_2,k+1))\ \ 1)$ and by the induction, $\lambda.(\lambda^k.(PC(a_1,k+1)\ \ PC(a_2,k+1))\ \ 1) \to_{eta} \lambda^k.(PC(c_1,k)\ \ PC(c_2,k)) = PC(\lambda^k.(c_1\ \ c_2),0)$.

Fourth. On one side, that $a =_{\beta\eta} b$ implies $a_{pc} =_{\lambda s_e} b_{pc}$ 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 $a_{pc} =_{\lambda s_e} b_{pc}$ and select a' and b' normal forms of a and b, respectively. By previous items, terms a_{pc} and b_{pc} reduce to a'_{pc} and b'_{pc}, respectively. Consequently, $a'_{pc} =_{\lambda s_e} b'_{pc}$ and $a'_{pc} = b'_{pc}$ since these terms should be λs_e-normal. To conclude, by the fact that the pre-cooking translation is injective on $\Lambda_{dB}(\mathcal{X})$, $a' = b'$. Then we obtain that $a =_{\beta\eta} b$. □

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 φ operator of the λs_e-calculus.

Finally, we relate solutions and their existence in the pure λ-calculus and for the corresponding pre-cooked terms in the λs_e-calculus.

Proposition 57 (Correspondence between solutions). *Let a and b be terms in $\Lambda_{dB}(\mathcal{X})$. Then there exist terms $N_1,\ldots,N_p$ in $\Lambda_{dB}(\mathcal{X})$ such that $a\{X_1/N_1,\ldots,X_p/N_p\} =_{\beta\eta} b\{X_1/N_1,\ldots,X_p/N_p\}$ if and only if there exist λs_e-terms $M_1,\ldots,M_p$ such that $a_{pc}\{X_1/M_1,\ldots,X_p/M_p\}_g =_{\lambda s_e} b_{pc}\{X_1/M_1,\ldots,X_p/M_p\}_g$.*

Proof. On the one side, suppose that $\{X_i/N_i\}_{i=1..p}$ is a solution of the unification problem $a =^?_{\beta\eta} b$. Then $a\{X_i/N_i\} =_{\beta\eta} b\{X_i/N_i\}$. By the fourth item of the Proposition 56, $(a\{X_i/N_i\})_{pc} =_{\lambda s_e} (b\{X_i/N_i\})_{pc}$ and by the Proposition 55 we obtain that $a_{pc}\{X_i/N_{i_{pc}}\}_g =_{\lambda s_e} b_{pc}\{X_i/N_{i_{pc}}\}_g$.

On the other side, suppose that $a_{pc}\{X_i/M'_i\}_g =_{\lambda s_e} b_{pc}\{X_i/M'_i\}_g$. We select terms N_i, $i = 1,...,p$, in the range of the pre-cooking translation such that $N_i =_{\lambda s_e} M'_i$ and let M_i be terms in $\Lambda_{dB}(\mathcal{X})$ such that $M_{i_{pc}} = N_i$. Then $a_{pc}\{X_i/M_{i_{pc}}\}_g =_{\lambda s_e} b_{pc}\{X_i/M_{i_{pc}}\}_g$ and by the Proposition 55 we obtain that $(a\{X_i/M_i\})_{pc} =_{\lambda s_e} (b\{X_i/M_i\})_{pc}$. This jointly with the Proposition 56 implies that $a\{X_i/M_i\} =_{\beta\eta} b\{X_i/M_i\}$. □

6 Conclusions

Following the $\lambda\sigma$-unification approach introduced in [DHK00], we have developed an effective strategy for implementing the λs_e-unification rules presented in [ARK00]. Additionally, we presented a pre-cooking translation that transcribes pure λ-terms in de Bruijn notation into λs_e-terms, for which the search of grafting solutions corresponds to substitution solutions in the pure λ-calculus.

Note that correctness and completeness proofs for the $\lambda\sigma$- and the λs_e-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 λs_e-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 λs_e-calculus.

More concretely, our pre-cooking translation transcribes a term a by replacing each occurrence of a meta-variable X with $\varphi_0^{k+1}X$ while the $\lambda\sigma$-calculus uses $X[\uparrow^k]$, where k 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 $\mathtt{n}$ in a into $1[\uparrow^{n-1}]$. 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 φ operator in the λs_e-calculus. This can be observed in the proof of the correct semantics of the pre-cooking translation (Proposition 55) and the proof of Proposition 56 which relates the existence of unification solutions in the λ- and the λs_e-calculus. In these proofs, only a correct selection of the scripts for the operator φ 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 λs_e-calculus into a description of solutions of the corresponding HOU problems in the pure λ-calculus.

Additionally, a formal distinction, from the practical point of view, between the λs_e-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 λProlog) should be elaborated. This is meaningful, since the λs_e-calculus and the calculus of [NW98,NW99] have correlated nice properties. For instance the laziness in the substitution needed in implementations of β-reduction, that arises naturally in the λs_e-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 λs_e-HOU approach in that practical framework.

References

[ACCL91] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit Substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[ARK00] M. Ayala-Rincón and F. Kamareddine. Unification via λs_e-Style of Explicit Substitution. In *Second International Conference on Principles and Practice of Declarative Programming*, Montreal, Canada, September 2000. Technical Report *Higher Order Unification via λs-Style of Explicit Substitution*, Computer and Electrical Engineering, Heriot-Watt University, Dec. 1999. Available at `http://www.cee.hw.ac.uk/ultra`.

[Bar84] H. Barendregt. *The Lambda Calculus : Its Syntax and Semantics (revised edition)*. North Holland, 1984.

[BN98] F. Baader and T. Nipkow. *Term Rewriting and* All That. Cambridge University Press, 1998.

[CHL96] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence Properties of Weak and Strong Calculi of Explicit Substitutions. *Journal of the ACM*, 43(2):362–397, 1996. Also as *Rapport de Recherche* INRIA 1617, 1992.

[DHK00] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Higher-order Unification via Explicit Substitutions. *Information and Computation*, 157(1/2):183–235, 2000.

[KR97] F. Kamareddine and A. Ríos. Extending a λ-calculus with Explicit Substitution which Preserves Strong Normalisation into a Confluent Calculus on Open Terms. *Journal of Functional Programming*, 7:395–420, 1997.

[NGdV94] R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer. *Selected papers on Automath*. North-Holland, Amsterdam, 1994.

[NW98] G. Nadathur and D. S. Wilson. A Notation for Lambda Terms A Generalization of Environments. *Theoretical Computer Science*, 198:49–98, 1998.

[NW99] G. Nadathur and D. S. Wilson. 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.

[Río93] A. Ríos. *Contribution à l'étude des λ-calculs avec substitutions explicites*. PhD thesis, Université de Paris 7, 1993.