ON APPLYING THE λs_e-STYLE OF UNIFICATION FOR SIMPLY-TYPED HIGHER ORDER UNIFICATION*

Mauricio Ayala-Rincón† **Fairouz Kamareddine**

Abstract

Dowek, Hardin and Kirchner developed a higher order unification (HOU) method based on the $\lambda\sigma$-style of explicit substitutions (which uses two sorts of objects: *terms* and *substitutions*). The novelty of this method rests on the possibility to resolve HOU problems by first order unification (FOU). This is achieved via a *pre-cooking* translation of the HOU problem into an FOU problem of the $\lambda\sigma$-calculus. Solutions to the FOU problem are then translated *back* into the range of the *pre-cooking* translation and subsequently to solutions of the original problem in the λ-calculus. Recently we studied unification in the λs_e-style of explicit substitutions which only uses one sort of objects: *terms*. We believe that λs_e-unification enables quicker detection of redices and has a clearer semantics. In this paper, we provide a pre-cooking translation for applying λs_e-unification to HOU in the λ-calculus. The *pre-cooking* jointly with a *back* translation complement the λs_e-unification method. We establish correctness and completeness and show why avoiding the use of substitution objects makes λs_e-HOU more efficient than $\lambda\sigma$-HOU.

1 Background

HOU via explicit substitutions as in [8] is illustrated by Figure 1 where solving a higher-order unification problem in the λ-calculus amounts to the following:

*Supported by the Brazilian CNPq research council grant number 47488/01-6.
†Partially supported by the FEMAT Brazilian foundation for research in mathematics.
Keywords: *Unification, explicit substitutions, λ-calculus, type theory*

1. The higher-order problem of the λ-calculus is translated (or precooked) into a first order problem of the $\lambda\sigma$-calculus.
2. The first order problem is solved in the $\lambda\sigma$-calculus using $\lambda\sigma$-unification.
3. Solutions obtained in step 2 are translated back into the range of the pre-cooking translation and then translated back into the λ-calculus.

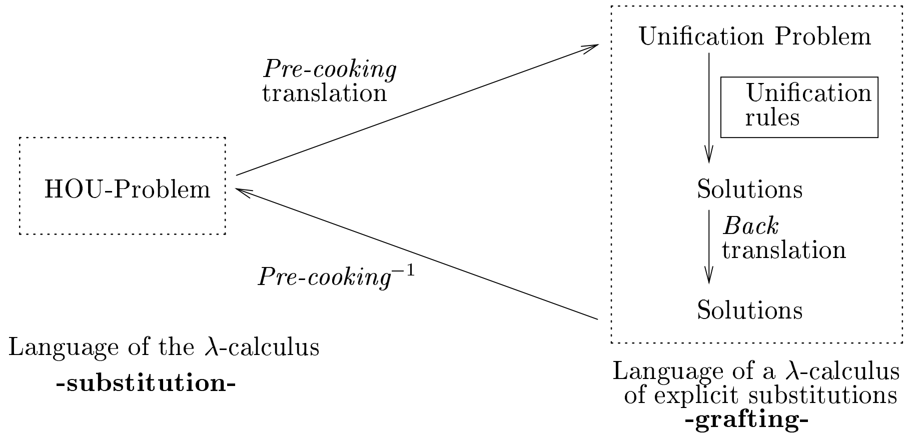

Figure 1: HOU method via calculi of explicit substitutions

In [3] we followed [8] in formalizing a unification system based on the λs_e-style in which first-order unification problems are solved in the λs_e-calculus. However, [3] only dealt with step 2 of the above 3 steps of [8]. In this paper, we close the gap and fill the other steps. We give a pre-cooking translation for applying λs_e-unification to HOU in the λ-calculus. The *pre-cooking* jointly with a *back* translation complement our λs_e-unification method. We show the correctness and completeness of our pre-cooking and back translations.

The $\lambda\sigma$- and the λs_e-calculi use de Bruijn indices instead of variable names in order to be closer to implementation and to avoid the problems that result from variable clashes. However, there are two differences between $\lambda\sigma$ and λs_e:

- $\lambda\sigma$ uses only one de Bruijn index (1) and builds the others by operations in the calculus. λs_e uses all the de Bruijn indices.
- λs_e remains close to the λ-calculus by adding updating and substitution operators and using one sort of objects: *terms*; $\lambda\sigma$ adds categorical operators like composition, cons and lift and a new sort of objects: *substitutions*.

In this paper, we focus on the advantages of using all de Bruijn indices and only term objects when implementing the λs_e-HOU approach over $\lambda\sigma$-HOU and its implementation as described in [6]. We show why avoiding the use of substitution objects makes λs_e-HOU more efficient than $\lambda\sigma$-HOU.

It should be stressed that $\lambda\sigma$ and λs_e are non-isomorphic styles of explicit substitutions [12] and hence 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.

For a set of operators $\mathcal{F}$, we assume familiarity with $\mathcal{F}$-algebras and with a term algebra $\mathcal{T}(\mathcal{F},\mathcal{X})$ built on a (countable) set of variables $\mathcal{X}$ and on $\mathcal{F}$. Variables in $\mathcal{X}$ are denoted by $X, Y, \ldots$. 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 λ-calculus [5] and with basic rewriting [4]. We denote with $\rightarrow_R^*$ the *reflexive and transitive closure* of a *reduction relation* $\rightarrow_R$ over a set A. The subscript R is usually omitted. Syntactical identity is denoted by $a = b$. We assume the usual definitions for Church Rosser (CR) and Weak Normalisation (WN) of a reduction relation.

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 $\mathcal{T}(\mathcal{F},\mathcal{X})$ 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 symbol. The **domain** of a grafting θ is defined by $Dom(\theta) = \{X \mid X\theta \neq X, X \in \mathcal{X}\}$. 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 as in: $\{X/X\theta \mid X \in Dom(\theta)\}_g$.

We assume familiarity with de Bruijn notation [7], with β- and η-reduction and with $=_\beta$ and $=_{\beta\eta}$ as well as with the $\lambda\sigma$- ($\cdot, \circ, []$ and $\uparrow$) and the λs_e-calculi (skeleton notation ψ), their typed versions and their normal form (nf, lnf and η-nf) characterizations as in [3]. $\Lambda_{dB}(\mathcal{X})$ denotes the λ-terms in de Bruijn notation over a set of (unification) *meta-variables* $\mathcal{X}$ and for a term a and a **substitution** θ, a^+ and θ^+ denote their *lifts* as in [3]. We recall that terms of the λs_e**-calculus**, whose set of rules is presented in Table 1, are given by: $\Lambda s_e \ ::= \ X \,|\, \mathbb{N} \,|\, \Lambda s_e \Lambda s_e \,|\, \lambda \Lambda s_e \,|\, \Lambda s_e \, \sigma^j \Lambda s_e \,|\, \varphi^i_k \Lambda s_e$, where $j,\, i \geq 1\,,\ \ k \geq 0\,, X \in \mathcal{X}$. The equational theory of the rewriting system λs_e defines a congruence $=_{\lambda s_e}$. The congruence obtained by dropping *σ-generation* and *Eta* is denoted by $=_{s_e}$.

2 Unification in the λs_e-calculus

In this section we review the λs_e-unification of [3]. Normal form characterizations (normal form (nf) and long normal forms (lnf)), WN and CR are essential for a unification method for λs_e, which can lead to HOU in the λ-calculus.

Table 1: The Rewriting System of the λs_e-calculus with Eta rule

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

Let $\mathcal{T}(\mathcal{F},\mathcal{X})$ be a term algebra 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 $s =^?_{\mathcal{A}} t$ for $s, t \in \mathcal{T}(\mathcal{F},\mathcal{X})$. Unification problems are written as disjunctions of existentially quantified conjunctions of atomic equational unification problems: $D = \bigvee_{j\in J} \exists \vec{w_j} \bigwedge_{i\in I_j} s_i =^?_{\mathcal{A}} t_i$. When $|J| = 1$, the unification problem is called a **unification system**. Variables in the set $\vec{w}$ of a unification system $\exists\vec{w} \bigwedge_{i\in I} s_i =^?_{\mathcal{A}} t_i$ are bound while all others 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 $\exists\vec{w} \bigwedge_{i\in I} s_i =^?_{\mathcal{A}} t_i$ is a grafting σ such that $\mathcal{A} \models \exists\vec{w} \bigwedge_{i\in I} s_i\sigma_{|\vec{w}} = t_i\sigma_{|\vec{w}}$ where $\sigma_{|\vec{w}}$ denotes the restriction of the grafting σ to the domain $\mathcal{X} \setminus \vec{w}$. A unifier of $\bigvee_{j\in J} \exists\vec{w_j} \bigwedge_{i\in I_j} s_i =^?_{\mathcal{A}} t_i$ 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 1 *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 by* $a =^?_{\lambda s_e} b$*, where* a *and* b *are* λs_e*-terms of the same sort. An equation is called trivial when it is of the form* $a =^?_{\lambda s_e} a$*.*

[3] gave a set of rewrite rule schemata that simplify unification problems and lead to a description of the set of unifiers. Basic decomposition rules for unification are applied modulo the usual boolean simplification rules as in [8].

Definition 2 ([3]) *Table 2 defines the* λs_e**-unification rules** *for typed* λs_e*-unification problems.*

Table 2: λs_e-unification rules

(Dec-λ)	$P \wedge \lambda_A.a =^?_{\lambda s_e} \lambda_A.b \quad \to \quad P \wedge a =^?_{\lambda s_e} b$
(Dec-App)	$P \wedge (\mathtt{n}\; a_1 \dots a_p) =^?_{\lambda s_e} (\mathtt{n}\; b_1 \dots b_p) \quad \to \quad P \bigwedge_{i=1..p} a_i =^?_{\lambda s_e} b_i$
(App-Fail)	$P \wedge (\mathtt{n}\; a_1 \dots a_p) =^?_{\lambda s_e} (\mathtt{m}\; b_1 \dots b_q) \quad \to \quad \mathbb{F}$ if $\mathtt{n} \neq \mathtt{m}$
(Dec-φ)	$P \wedge \varphi^i_k a =^?_{\lambda s_e} \varphi^i_k b \quad \to \quad P \wedge a =^?_{\lambda s_e} b$, where $\varphi^i_k a, \varphi^i_k b$ are long-normal terms
(Exp-λ)	$P \quad \to \quad \exists (A.\Gamma \vdash Y \,:\, B), P \wedge X =^?_{\lambda s_e} \lambda_A.Y$ if $Y \notin var(P)$, $(\Gamma \vdash X \,:\, A \to B) \in var(P)$ and X is a unsolved variable
(Exp-App)	$P \wedge \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(X, a_1, \dots, a_p) =^?_{\lambda s_e} (\mathtt{m}\; b_1 \dots b_q) \quad \to$ $P \wedge \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(X, a_1, \dots, a_p) =^?_{\lambda s_e} (\mathtt{m}\; b_1 \dots b_q) \quad \wedge$ $\bigvee_{r \in R_p \cup R_i} \exists H_1, \dots, H_k, X =^?_{\lambda s_e} (\mathtt{r}\; H_1 \dots H_k)$ 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 contexts $\Gamma_{H_i} = \Gamma_X$, 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)	$P \wedge X =^?_{\lambda s_e} a \quad \to \quad \{X/a\} P \wedge X =^?_{\lambda s_e} a$ if $X \in var(P), X \notin var(a)$ and $a \in \mathcal{X} \Rightarrow a \in var(P)$
(Normalize)	$P \wedge a =^?_{\lambda s_e} b \quad \to \quad P \wedge a' =^?_{\lambda s_e} b'$ 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

Since λs_e is CR and WN [11], the search can be restricted[1] to η-long nor-

[1] Use of λs_e-normal forms in *Exp-App* is not essential but simplifies the case analysis presented in the definition of the rule and its completeness proof. It can be dropped and subsequently incorporated as an efficient unification strategy, where before applying *Exp-App*, λs_e-unification problems are normalized.

mal 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 $(\mathtt{k}\ b_1 \ldots b_p)$ or $a\sigma^i b$ or $\varphi^i_k a$, where in the first case $\mathtt{k}$ can be omitted and p is zero. The *Eta* rule reduces the number of cases (or unification rules) to be considered when defining the unification algorithm, but as for the $\lambda\sigma$-calculus, it can be dropped. *Normalize* and *Dec-λ* use CR and WN of λs_e to normalize equations of the form $\lambda.a =^?_{\lambda s_e} \lambda.b$ into $a' =^?_{\lambda s_e} b'$ and *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 $A \to B$, where Y is a new variable of type B. *Dec-App* and *App-Fail* transform equations of the form $(\mathtt{n}\ a_1 \ldots a_p) =^?_{\lambda s_e} (\mathtt{m}\ b_1 \ldots b_q)$ into the empty disjunction when $\mathtt{n} \neq \mathtt{m}$, as they have no solution, or into the conjunction $\bigwedge_{i=1..p} a_i =^?_{\lambda s_e} b_i$, when $\mathtt{n} = \mathtt{m}$. Analogously, *Dec-φ* decomposes equations with leading operator φ. It can be easily checked, using the arithmetic properties of λs_e to build counterexamples, that the addition of the corresponding *Dec-σ*, *σ-Fail* and *φ-Fail* is wrong. In $\lambda\sigma$, the rule *Exp-App* advances towards solutions to equations of the form $X[a_1 \ldots a_p. \uparrow^n] =^?_{\lambda s_e} (\mathtt{m}\ b_1 \ldots b_q)$ where X is an unsolved variable of an atomic type. This process is similar for λs_e-unification problems.

Example 1 *Take the problem* $(\lambda.(\lambda.(X\ \ \mathtt{2})\ \ \mathtt{1})\ \ Y) =^?_{\lambda s_e} (\lambda.(Z\ \ \mathtt{1})\ \ U)$ *where* X, Y, Z *and* U *are meta-variables. By Normalize we get* $((X\sigma^2 Y)\sigma^1(\varphi^1_0 Y)\ \ \varphi^1_0 Y) =^?_{\lambda s_e} (Z\sigma^1 U\ \ \varphi^1_0 U)$ *which after Dec-App, -φ and Replace gives* $(X\sigma^2 Y)\sigma^1(\varphi^1_0 Y) =^?_{\lambda s_e} Z\sigma^1 Y \wedge Y =^?_{\lambda s_e} U$*. Since* X *and* Z *are variables of functional type, Exp-App and Replace give* $((\lambda.X')\sigma^2 Y)\sigma^1(\varphi^1_0 Y) =^?_{\lambda s_e} (\lambda.Z')\sigma^1 Y \wedge Y =^?_{\lambda s_e} U \wedge X =^?_{\lambda s_e} \lambda.X' \wedge Z =^?_{\lambda s_e} \lambda.Z'$*. Finally, Normalize and Dec-λ give* $(X'\sigma^3 Y)\sigma^2(\varphi^1_0 Y) =^?_{\lambda s_e} Z'\sigma^2 Y \wedge Y =^?_{\lambda s_e} U \wedge X =^?_{\lambda s_e} \lambda.X' \wedge Z =^?_{\lambda s_e} \lambda.Z'$*. Solutions are built as* $\{Y/X_1, U/X_1\}$ *union solutions for* X *and* Z *obtained by the first equation. The first equations, called Flex-Flex, are related to the* pre-unifiers *of [10]. E.g., here we can take* $\{Y/X_1, U/X_1\} \bigcup \{X/\lambda.\mathtt{n+1}, Z/\lambda.\mathtt{n}\}$*, where* $n > 2$*.*

Example 2 *from* $\lambda.(\lambda.(Y\ \ \mathtt{1})\ \ \lambda.(X\ \ \mathtt{1})) =^? \lambda.(\lambda.V\ \ \lambda.W)$ *one obtains:* $(Y[\lambda.(X\ \ \mathtt{1}).id]\ \ \lambda.(X\ \ \mathtt{1})) =^?_{\lambda\sigma} V[\lambda.W.id]$ *and* $(Y\sigma^1\lambda.(X\ \ \mathtt{1})\ \ \lambda.(\varphi^1_1\ \mathtt{1})) =^?_{\lambda s_e} V\sigma^1\lambda.W$*. By Exp-App with* $V =^?_{\lambda\sigma} (V_1\ \ V_2)$ *and* $V =^?_{\lambda s_e} (V_1\ \ V_2)$*, we get* $\lambda.(X\ \ \mathtt{1}) =^?_{\lambda\sigma} V_2[\lambda.(X\ \ \mathtt{1}).id]$ *and* $\lambda.(\varphi^1_1 X\ \ \mathtt{1}) =^?_{\lambda s_e} V_2\sigma^1\lambda.(X\ \ \mathtt{1})$*. For solutions take* $V_2 =^?_{\lambda\sigma} \mathtt{1}$ *or* $V_2 =^?_{\lambda s_e} \mathtt{1}$*.*

Definition 3 *A unification system* P *is in* λs_e-**solved form** *if its meta-variables are atomic and it is a conjunction of non trivial equations of the forms:*

(Solved) $X =^?_{\lambda\sigma} a$, *where* X *does not occur anywhere else in* P *and* a *is in long normal form. Both* X *and* $X =^?_{\lambda\sigma} a$ *are said to be* **solved** *in* P.
(Flex-Flex) non solved equations between long normal terms whose root operator is σ *or* φ *which we represent as equations between their skeleton:*
$\psi^{j_p}_{i_p} \ldots \psi^{j_1}_{i_1}(X, a_1, \ldots, a_p) =^?_{\lambda s_e} \psi^{l_q}_{k_q} \ldots \psi^{l_1}_{k_1}(Y, b_1, \ldots, b_q)$ *with* X, Y *atomic.*

Lemma 1 ([3])

1. *Any* λs_e*-solved form has* λs_e*-unifiers;*
2. *Well-typedness: Deduction by the* λs_e*-unification rules of a well-typed equation gives rise only to well-typed equations,* $\mathbb{T}$ *and* $\mathbb{F}$*;*
3. *Equivalence of solvedness and normalization: Solved problems are normalized for the* λs_e*-unification rules. And, a system which is a conjunction of equations that cannot be reduced by* λs_e*-unification rules is solved.*

Definition 4 *Let* P *and* P' *be* λs_e*-unification problems, let "rule" denote the name of a* λs_e*-unification rule and "*$\rightarrow^{rule}$*" its corresponding deduction relation. By* **correctness** *and* **completeness** *of rule we understand* $P \rightarrow^{rule} P'$ *implies* $\mathcal{U}_{\lambda s_e}(P') \subseteq \mathcal{U}_{\lambda s_e}(P)$ *and* $P \rightarrow^{rule} P'$ *implies* $\mathcal{U}_{\lambda s_e}(P) \subseteq \mathcal{U}_{\lambda s_e}(P')$*, respectively.*

Theorem 1 (Correctness and completeness [3]) *The* λs_e*-unification rules are correct and complete.*

An analogous unification strategy to that of [8] for $\lambda\sigma$ applies in this setting. Correctness and completeness proofs for these strategies essentially do not differ because they are based on an appropriate ordering of the application of the unification rules which is independent of the calculi [2].

3 HOU in the pure λ-calculus

[3] dealt only with the λs_e-unification method (half of the box on the right hand side of Figure 1). For applying this method to HOU in the λ-calculus we need to complete the diagram by providing the pre-cooking a nd Back translations, show their correctness and completeness and establish the applicability of λs_e-unification for HOU in the pure λ-calculus. This is what we do in this section.

Observe firstly that unifying two terms a and b in the λ-calculus consists in finding a *substitution* θ such that $\theta(a) =_{\beta\eta} \theta(b)$. Thus using the notation of substitution a unifier in the λ-calculus of the problem $\lambda.X =^?_{\beta\eta} \lambda.2$ is not a term $t = \theta X$ such that $\lambda.t =^?_{\beta\eta} \lambda.2$ but a term $t = \theta X$ such that $\theta(\lambda.X) = \lambda.\theta^+(X) = \lambda.2$. 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. We illustrate in the next example how searching for substitution solutions of a HOU problem $a =^?_{\beta\eta} b$ corresponds to searching for grafting solutions of a unification problem in λs_e.

Example 3 *Consider the HOU 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 $\theta'(\lambda.(\varphi^2_0(X)\ \ 2)) =_{\lambda s_e} \lambda.2$. In the $\lambda\sigma$-calculus, $\lambda.(X\ \ 2)$ is pre-cooked into $\lambda.(X[\uparrow]\ \ 2)$. This correspondence results from one between both Eta rules (i.e., between $b[\uparrow] = a$ and $\varphi^2_0 b = a$). Then we should search for unifiers for the problem $\lambda.(\varphi^2_0(X)\ \ 2) =^?_{\lambda s_e} \lambda.2$.*

Now we apply λs_e-unification rules to the problem $\lambda.(\varphi^2_0(X)\ \ 2) =^?_{\lambda s_e} \lambda.2$. By applying Dec-$\lambda$ and Exp-λ we get $(\varphi^2_0(X)\ \ \ 2) =^?_{\lambda s_e} 2$ and subsequently $\exists Y(\varphi^2_0(X)\ \ \ 2) =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y$. Then by applying Replace and Normalize we obtain $\exists Y(\varphi^2_0(\lambda.Y)\ \ 2) =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y$ and $\exists Y(\varphi^2_1 Y)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y$. Now, we obtain $(\exists Y(\varphi^2_1 Y)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y) \wedge (Y =^?_{\lambda s_e} 1 \vee Y =^?_{\lambda s_e} 2)$ by applying Exp-app; by applying Replace: $((\varphi^2_1 1)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.1) \vee ((\varphi^2_1 2)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.2)$; and by applying 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.

In general, before the unification process, a λ-term a should be translated into a λs_e-term a' obtained by simultaneously replacing each occurrence of a meta-variable X at position i in a by $\varphi^{k+1}_0 X$, where k is the number of abstractors between the root position of a and position i. If $k = 0$ then the occurrence of X remains unchanged. The pre-cooking translation defined in [8] transcribes all occurrences of de Bruijn indices $\mathtt{n}$ into $1[\uparrow^{n-1}]$ and all occurrences of meta-variables X into $X[\uparrow^k]$, with k as above. Notice that the two pre-cooking translations can be implemented non-recursively in an efficient way.

Example 4 *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 in 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 an application of Exp-λ and 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. Normalize gives $Y\sigma^1(\mathtt{2}\ \ \mathtt{1}) =^?_{\lambda s_e} (\mathtt{2}\ \ Y\sigma^1\mathtt{1}) \wedge X =^?_{\lambda s_e} \lambda.Y$ *And by Exp-App we get* $Y\sigma^1(\mathtt{2}\ \ \mathtt{1}) =^?_{\lambda s_e} (\mathtt{2}\ \ Y\sigma^1\mathtt{1}) \wedge X =^?_{\lambda s_e} \lambda.Y \wedge (Y =^?_{\lambda s_e} \mathtt{1} \vee Y =^?_{\lambda s_e} (\mathtt{3}\ \ H_1))$.

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

Second solved system: *It is possible to obtain additional solutions from the equational system:* $(\mathtt{2}\ \ H_1\sigma^1(\mathtt{2}\ \ \mathtt{1})) =^?_{\lambda s_e} (\mathtt{2}\ \ (\mathtt{2}\ \ H_1\sigma^1\mathtt{1})) \wedge X =^?_{\lambda s_e} \lambda.(\mathtt{3}\ \ H_1)$. *In fact, by Dec-App and Exp-App we obtain* $H_1\sigma^1(\mathtt{2}\ \ \mathtt{1}) =^?_{\lambda s_e} (\mathtt{2}\ \ H_1\sigma^1\mathtt{1}) \wedge X =^?_{\lambda s_e} \lambda.(\mathtt{3}\ \ H_1) \wedge (H_1 =^?_{\lambda s_e} \mathtt{1} \vee H_1 =^?_{\lambda s_e} (\mathtt{3}\ \ H_2))$, *which by Replace and Normalize gives* $((\mathtt{2}\ \ \mathtt{1}) =^?_{\lambda s_e} (\mathtt{2}\ \ \mathtt{1}) \wedge X =^?_{\lambda s_e} \lambda.(\mathtt{3}\ \ \mathtt{1})) \vee ((\mathtt{2}\ \ H_2\sigma^1(\mathtt{2}\ \ \mathtt{1})) =^?_{\lambda s_e} (\mathtt{2}\ \ (\mathtt{2}\ \ H_2\sigma^1\mathtt{1})) \wedge X =^?_{\lambda s_e} \lambda.(\mathtt{3}\ \ (\mathtt{3}\ \ H_2)))$, *from which we obtain the second solved system corresponding to the grafting solution:* $\{X/\lambda.(\mathtt{3}\ \ \mathtt{1})\}$. *This corresponds to the solution* $F = f$*; in fact, by replacing* X *with* $\lambda.(\mathtt{3}\ \ \mathtt{1})$ *in the original unification problem we obtain* $(\lambda.(\mathtt{3}\ \ \mathtt{1})\ \ (\mathtt{2}\ \ \mathtt{1})) =^?_{\lambda s_e} (\mathtt{2}\ \ (\lambda.(\mathtt{3}\ \ \mathtt{1})\ \ \mathtt{1}))$. *Notice that indices* $\mathtt{3}$ *and* $\mathtt{2}$ *correspond to the same operator. Additionally, note that* $(\lambda.(\mathtt{3}\ \ \mathtt{1})\ \ (\mathtt{2}\ \ \mathtt{1})) \to_\beta (\mathtt{2}\ \ (\mathtt{2}\ \ \mathtt{1}))$ *and* $(\mathtt{2}\ \ (\lambda.(\mathtt{3}\ \ \mathtt{1})\ \ \mathtt{1})) \to_\beta (\mathtt{2}\ \ (\mathtt{2}\ \ \mathtt{1}))$.

Third solved system: *By continuing the application of Dec-App, Exp-App, Replace and Normalize we obtain grafting solutions corresponding to* $F = fff$, $F = ffff$, *etc. to the equational system* $((\mathtt{2}\ \ H_2\sigma^1(\mathtt{2}\ \ \mathtt{1})) =^?_{\lambda s_e} (\mathtt{2}\ \ (\mathtt{2}\ \ H_2\sigma^1\mathtt{1})) \wedge X =^?_{\lambda s_e} \lambda.(\mathtt{3}\ \ (\mathtt{3}\ \ H_2)))$ *we obtain the third solved system giving the grafting solution* $\{X/\lambda.(\mathtt{3}\ \ (\mathtt{3}\ \ \mathtt{1}))\}$ *corresponding to the solution* $F = ff$.

The unification process continues infinitely producing solved systems corresponding to the grafting solutions $\{X/\lambda.(\mathtt{3}\ \ (\mathtt{3}\ \ (\mathtt{3}\ \ \mathtt{1})))\}$ *(i.e.* $F = fff$*),*
$\{X/\lambda.(\mathtt{3}\ \ (\mathtt{3}\ \ (\mathtt{3}\ \ (\mathtt{3}\ \ \mathtt{1}))))\}$ *(i.e.* $F = ffff$*), etc.*

Definition 5 (Pre-cooking) *Take* $a \in \Lambda_{dB}(\mathcal{X})$ *where* $\Gamma \vdash_{\Lambda_{dB}(\mathcal{X})} a : T$ *(according to* (Var)*,* $(Varn)$*,* $(Lambda)$*,* (App)*, and* $(Meta)$ *of Table 3). We give every variable* X *of type* A *in* a *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)* $PC(X, n) = \begin{cases} X, & \text{if } n = 0 \\ \varphi_0^{n+1}X, & \text{otherwise} \end{cases}$

Lemma 2 (Type preservation) *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 every variable in a is given the same type and context Γ, 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. Cases $a = \mathtt{k}$ and $a = (a_1 \ \ a_2)$ are simple. Case $a = \lambda_B.b$, 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 $A_1 \ldots A_n, \Gamma \vdash_{\lambda s_e} PC(\lambda_B.b, n) = \lambda_B.PC(b, n+1) : B \to C$. Case $a = X$ then as $\Gamma \vdash_{\Lambda_{dB}(\mathcal{X})} X : T$, $\Gamma \vdash_{\lambda s_e} X : T$ and $A_1 \ldots A_n, \Gamma \vdash_{\lambda s_e} \varphi_0^{n+1}(X) : T$.

□

Proposition 1 relates substitution and grafting and justifies pre-cooking.

Proposition 1 (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. $PC(a\{X_1/b_1^{+^i}, \ldots, X_p/b_p^{+^i}\}, i) = PC(a, i)\{X_1/{b_1}_{pc}, \ldots, X_p/{b_p}_{pc}\}_g$ is proved by induction on the structure of terms for all i. The case $i = 0$ corresponds to the proposition.

□

In contrast to the related proof in [8] where substitution objects $[1...k. \uparrow^{i+k}]$ are necessary for proving the critical case of $a = X$ our proof uses pure term objects by selecting the appropriate super- and sub-scripts for φ (i.e., φ_k^{i+1}).

The next 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 2 *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* $\beta\eta$*-nf then* a_{pc} *is* λs_e*-nf*
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. Proved by induction on the structure of terms. For the first item, for instance, we prove by induction on a the more general fact that for all k, $(\lambda^{k+1}.a \ \ b) \to_\beta (\lambda^k.a)\{1/b\}$ implies $((\lambda^{k+1}.a) \ \ b)_{pc} \to^*_{\lambda s_e} ((\lambda^k.a)\{1/b\})_{pc}$. Our case of interest is when $k = 0$.

□

Again, our proof differs from that of [8] in that we avoid complicated substitution objects because we profit from the semantics of φ in the λs_e-calculus. Finally we relate solutions and their existence in the λ-calculus to those of λs_e.

Proposition 3 (Correspondence between solutions) *Let a, b 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* $M_1,\ldots,M_p$ *where* $a_{pc}\{X_1/M_1,\ldots,X_p/M_p\}_g =_{\lambda s_e} b_{pc}\{X_1/M_1,\ldots,X_p/M_p\}_g$.

Proof. If $\{X_i/N_i\}_{i=1..p}$ 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 Proposition 2.4, $(a\{X_i/N_i\})_{pc} =_{\lambda s_e} (b\{X_i/N_i\})_{pc}$. By Proposition 1, $a_{pc}\{X_i/N_{i_{pc}}\}_g =_{\lambda s_e} b_{pc}\{X_i/N_{i_{pc}}\}_g$.
If $a_{pc}\{X_i/M'_i\}_g =_{\lambda s_e} b_{pc}\{X_i/M'_i\}_g$ we select terms N_i, $i = 1,...,p$, in the pre-cooking range such that $N_i =_{\lambda s_e} M'_i$ and take M_i in $\Lambda_{dB}(\mathcal{X})$ such that $M_{i_{pc}} = N_i$. Hence, $a_{pc}\{X_i/M_{i_{pc}}\}_g =_{\lambda s_e} b_{pc}\{X_i/M_{i_{pc}}\}_g$. By Proposition 1, $(a\{X_i/M_i\})_{pc} =_{\lambda s_e} (b\{X_i/M_i\})_{pc}$. Hence by Proposition 2 $a\{X_i/M_i\} =_{\beta\eta} b\{X_i/M_i\}$.

$\square$

In addition to pre-cooking, we need a *Back* translation for giving descriptions of solutions of the original pre-cooked problems. That means, that for any unification problem P, derived by applying the λs_e-unification rules to the pre-cooking $a_{pc} =^?_{\lambda s_e} b_{pc}$, we have to reassemble a problem Q in the image of the pre-cooking translation with the same solutions as P. Subsequently, Q should be translated to the λ-calculus, by applying the inverse of the pre-cooking translation, into a HOU problem R (see Figure 1). Then the solutions of P coincide with the solutions of Q and are the pre-cooking of the solutions of R, which coincide with the solutions of the original HOU problem $a =^?_{\beta\eta} b$. In this way the set of solutions is given as solved forms. By the correspondence between solutions (Proposition 3), we have that if $a =^?_{\beta\eta} b$ has a solution then so does its pre-cooking $a_{pc} =^?_{\lambda s_e} b_{pc}$. Here we do not present the proof of the converse which can be done similarly to that of the $\lambda\sigma$-HOU approach in [8].

The λs_e-unification rules are extended with the following rules:

(Anti-Exp-λ) $P \to \exists Y(P \wedge X =^?_{\lambda s_e} (\varphi^2_0 Y \;\; \mathtt{1}))$ if $(X \; : \; A.\Gamma'_X \vdash A_X) \in var(P)$,
where $(Y \; : \; \Gamma'_X \vdash A \to A_X) \notin var(P)$

(Anti-Dec-λ) $P \wedge a =^?_{\lambda s_e} b \to P \wedge \lambda_A.a =^?_{\lambda s_e} \lambda_A.b$
if $a =^?_{\lambda s_e} b$ is well-typed in a context $A.\Gamma$

Proposition 4 (Correctness and completeness of the *Anti*-rules) *The rules of* λs_e*-unification, Anti-Exp-λ and Anti-Dec-λ are correct and complete.*

Proof. By Theorem 1 we only examine the two new rules. Correctness follows by inspection of the new rules. For completeness, observe that grafting solutions of $P \wedge a =^?_{\lambda s_e} b$, where the former equation is well-typed in the context

$A.\Gamma$, are also solutions of $P \wedge \lambda_A.a =^?_{\lambda s_e} \lambda_A.b$ (which is now the last well-typed equation in the context Γ). For *Anti-Exp-*λ, suppose that θ is a grafting solution of problem P and select $\theta' = \theta \cup \{Y/\lambda_A.\theta X\}$. Then $\theta'(\varphi^2_0 Y \ \ \mathtt{1}) = (\varphi^2_0 \lambda_A.\theta X \ \ \mathtt{1}) =_{\lambda s_e} (\lambda_A.\varphi^2_1 \theta X \ \ \mathtt{1}) =_{\lambda s_e} (\varphi^2_1 \theta X)\sigma^1 \mathtt{1}$. We analyse the former term. On one hand, φ^2_1 increases by one all free de Bruijn indices occurring at θX except those corresponding to the variable of the free de Bruijn index $\mathtt{1}$. On the other hand, "$\cdot\sigma^1\mathtt{1}$" decrements by one all free occurrences of de Bruijn indices in $\varphi^2_1\theta X$ except those untouched by φ^2_1. Then $(\varphi^2_1\theta X)\sigma^1\mathtt{1} =_{\lambda s_e} \theta X$.

□

The rule *Anti-Dec-*λ is applied only to equations whose contexts are *strict extensions* of Γ, i.e. of the form $A_1 \ldots A_n.\Gamma$ for $n > 0$. The rule *Anti-Exp-*λ only applies to variables, whose contexts are strict extensions of Γ. The **Back** strategy consists of applying the two new rules and the rule *Replace* eagerly.

Proposition 5 *Let $a =^?_{\beta\eta} b$ be a HOU problem well-typed in a context Γ and P derived by the λs_e-unification rules from its pre-cooking. By applying the Back strategy on P we obtain a system Q satisfying t he following invariants:*
1) if an equation is well-typed in context Δ, then Δ is an extension of Γ;
2) for every variable Y, its context Γ_Y is an extension of Γ;
3) for every subterm $\psi^{j_p}_{i_p} \ldots \psi^{j_1}_{i_1}(X, a_1, \ldots, a_p)$ in P we have $p \le |\Gamma_Y| - |\Gamma| + 1$.

Proof. We omit the proof of the fact that P satisfies these invariants. This is done by induction on the structure of the derivation via the λs_e-unification rules. Suppose that P satisfies these invariants. Then since the rules *Anti-Exp-*λ and *Anti-Dec-*λ are applied only on variables and equations, whose contexts are strict extensions of Γ, the first and second invariants are maintained. The third invariant is maintained too, since subterms of the form $\psi^{j_p}_{i_p} \ldots \psi^{j_1}_{i_1}(X, a_1, \ldots, a_p)$ are either already of this form in P or obtained by the two new rules as $\varphi^2_0 Y$ in whose case $p = 1 \le |\Gamma_Y| - |\Gamma| + 1$ holds, since Γ_Y is an extension of Γ.

□

Proposition 6 (Building Back Pre-cooking images) *Let P be a problem derived from the application of the λs_e-unification rules to the pre-cooking of a given HOU problem $a =^?_{\beta\eta} b$. The system resulting from normalization of P by applying the Back strategy is the pre-cooking of a problem in the λ-calculus.*

Proof. This is proved by simple examination of the effects of the rules *Anti-Dec-λ* and *Anti-Exp-λ* over P. In P every context Γ_X is an extension of Γ and every equation is well-typed in an extension of Γ. Thus applying *Anti-Dec-λ* and *Anti-Exp-λ* and then *Replace* to all the variables and equations whose contexts are not Γ (thus strict extensions of Γ), we obtain an equational problem in λs_e such that all equations are well-typed in the context Γ and also all variables occurring in the problem have context Γ. The obtained problem is the pre-cooking of a problem in the λ-calculus. In fact, if Γ is a context and b an s_e-normal form as above whose variables have context Γ, then b belongs to the image of the pre-cooking translation. This is proved as follows. Every occurrence of a variable X belongs to a subterm of the form $\psi_{i_p}^{j_p} \ldots \psi_{i_1}^{j_1}(X, a_1, \ldots, a_p)$. We have that $p \leq |\Gamma_X| - |\Gamma| + 1$ and since $\Gamma_X = \Gamma$, $p = 1$ or $p = 0$. For the interesting case, $p = 1$, this term is of the form $\psi_i^j(X, a)$. The former term cannot be of the form $X\sigma^i a$, because in this case the context of a corresponds to $\Gamma_{>i}$ and the whole term is of type A_X in the context $\Gamma_{<i}.\Gamma_{>i}$, that is not an extension of Γ. Consequently the term is necessarily of the form $\varphi_i^j(X)$. Suppose that $\Gamma = \Delta_{\leq i}.\Delta_{\geq i+j}$. Then $\varphi_i^j(X)$ is of type A_X and its context corresponds to Δ, that is an extension of Γ whenever $i = 0$; i.e., $\Delta = A_1 \ldots A_{j-1}.\Gamma$. Thus we can conclude that b is in the image of the pre-cooking translation.

$\square$

Corollary 1 (Soundness of the construction of solutions) *Let $a =_{\beta\eta}^? b$ a HOU problem such that its pre-cooking, normalised with the λs_e-unification rules gives a disjunction of systems that has one of its components, say P, solved. Let Q be the system resulting by normalising P with the Back strategy and let $R = PC^{-1}(Q)$. Then R is a λ-solved form (in the sense of [15]) and the solutions of R are solutions of the original HOU problem.*

Proof. The pre-cooking of a substitution solution θ of R in the λ-calculus, is a solution of R_{pc} and then of Q (Proposition 3). But, θ_{pc} is a solution of P (Proposition 4) and then of $a_{pc} =_{\lambda s_e}^? b_{pc}$ (Theorem 1). Hence θ is solution of $a =_{\beta\eta}^? b$ (the converse of Proposition 3).

$\square$

Theorem 2 (Completeness of the construction of solutions) *Let $a =_{\beta\eta}^? b$ a HOU problem such that its pre-cooking is well-typed in context Γ. Any* *solution of the initial problem can be obtained as the one of a system in λ-solved form resulting from the application of the λs_e-unification rules, followed by the Back strategy and the inverse of the pre-cooking translation.*

Proof. We use the same notation as in Corollary 1. θ is a substitution solution of R, if and only if θ_{pc} is a solution of R_{pc}, if and only if θ_{pc} is a solution of Q, if and only if θ_{pc} is a solution of P, if and only if θ_{pc} is a solution of $a_{pc} =^?_{\lambda s_e} b_{pc}$, if and only if θ is a solution of $a =^?_{\beta\eta} b$.

□

4 Efficiency considerations

We precise here why the use of the sole de Bruijn index $\mathtt{1}$ and of substitution objects make the $\lambda\sigma$-HOU approach less efficient than the λs_e-HOU one. Our comparisons are based on *naive* implementations obtained directly from the inference rules, but for actual implementations many of the problems pointed out may be circumvented. For instance, encodings of de Bruijn indices "$\mathtt{1}[\uparrow^n]$" in $\lambda\sigma$ may be easily avoided in a reasonable implementation of $\lambda\sigma$-HOU. But these simple observations are interesting since one of the objectives of explicit substitutions is to be close to implementation. Advantages of the $\lambda\sigma$-calculus in simultaneously applying different β-reductions [13] are not considered here.

For the sake of clarity, we have omitted above both types and contexts. But for the analysis of the HOU method above it is necessary to know both the types and contexts of all subexpressions during the unification process. Therefore terms "decorated" with types and contexts for all their subterms are necessary for any reasonable implementation. The general idea is to assign types and contexts to all subexpressions at the beginning of the unification process and to maintain this notation during the process via decorated versions of the λs_e-calculus, the λs_e-typing rules and, of course, the λs_e-unification rules. Table 3 gives the decorated version of the typing rules for the λs_e-calculus.

The typing rules *Var* and *Varn* can be reduced to a sole decorated rule of the form $\mathtt{n}^{A_1 \ldots A_n . \Gamma}_{A_n}$ making the decoration of de Bruijn indices a straightforward process which is linear in both time and space in n.

The rule *Meta* is added to type open terms and should be understood as follows: for every metavariable X, there exists a unique context Γ_X and a unique

Table 3: Undecorated and decorated typing rules for the λs_e-calculus

(Var)	$A.\Gamma \vdash \mathtt{1} : A$	$\mathtt{1}^{A.\Gamma}_{A}$
$(Varn)$	$\dfrac{\Gamma \vdash \mathtt{n} : B}{A.\Gamma \vdash \mathtt{n}+\mathtt{1} : B}$	$\dfrac{\mathtt{n}^{\Gamma}_{B}}{(\mathtt{n}+\mathtt{1})^{A.\Gamma}_{B}}$
$(Lambda)$	$\dfrac{A.\Gamma \vdash b : B}{\Gamma \vdash \lambda_A.b : A \to B}$	$\dfrac{b^{A.\Gamma}_{B}}{(\lambda_A.b^{A.\Gamma}_{B})^{\Gamma}_{A\to B}}$
(App)	$\dfrac{\Gamma \vdash b : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash (b\ \ a) : B}$	$\dfrac{b^{\Gamma}_{A\to B}\ ,\ a^{\Gamma}_{A}}{(b^{\Gamma}_{A\to B}\ \ a^{\Gamma}_{A})^{\Gamma}_{B}}$
$(Sigma)$	$\dfrac{\Gamma_{\geq i} \vdash b : B \quad \Gamma_{<i}.B.\Gamma_{\geq i} \vdash a : A}{\Gamma \vdash a\,\sigma^i b : A}$	$\dfrac{b^{\Gamma_{\geq i}}_{B}\ ,\ a^{\Gamma_{<i}.B.\Gamma_{\geq i}}_{A}}{(a^{\Gamma_{<i}.B.\Gamma_{\geq i}}_{A}\,\sigma^i b^{\Gamma_{\geq i}}_{B})^{\Gamma}_{A}}$
(Phi)	$\dfrac{\Gamma_{\leq k}.\Gamma_{\geq k+i} \vdash a : A}{\Gamma \vdash \varphi^i_k a : A}$	$\dfrac{a^{\Gamma_{\leq k}.\Gamma_{\geq k+i}}_{A}}{(\varphi^i_k a^{\Gamma_{\leq k}.\Gamma_{\geq k+i}}_{A})^{\Gamma}_{A}}$
$(Meta)$	$\Gamma_X \vdash X : A_X$	$X^{\Gamma_X}_{A_X}$

type A_X such that the rule holds. This is done in order to obtain compatibility between typing and grafting. We suppose that for each pair (Γ, A) there exists an infinite set of variables X such that $\Gamma_X = \Gamma$ and $A_X = A$.

In $\lambda\sigma$ the corresponding rules are adapted for the manipulation of substitution objects. Types of substitutions are contexts (denoted in the undecorated setting as $s \triangleright \Gamma$). Examples of these rules are: *(Shift)* $\uparrow^{A.\Gamma}_{\Gamma}$; *(Comp)*$s^{\Theta}_{\Gamma}\ ,\ t^{\Delta}_{\Theta} \vdash (s^{\Theta}_{\Gamma} \circ t^{\Delta}_{\Theta})^{\Delta}_{\Gamma}$; *(Clos)*$a^{\Delta}_{A}\ ,\ s^{\Gamma}_{\Delta} \vdash (a^{\Delta}_{A}[s^{\Gamma}_{\Delta}])^{\Gamma}_{A}$. This kind of explicit decoration was done $\lambda\sigma$-HOU in [6], but maintaining this discipline in the λs_e-calculus is more economical in both space and time. Let us compare the previous linear decoration of a de Bruijn index, $\mathtt{n}$, in λs_e and its corresponding $\lambda\sigma$-term $1[\uparrow^{n-1}]$:

Example 5 *The decoration of* $\mathtt{1}[\uparrow^{n-1}]$ *uses quadratic space and time.*

$$
\textit{(comp)}\dfrac{\textit{(shift)}\uparrow^{A_{n-1}.A_n.\Gamma}_{A_n.\Gamma}\ ,\ \textit{(shift)}\uparrow^{A_{n-2}\ldots\Gamma}_{A_{n-1}.A_n.\Gamma}}{\textit{(comp)}\dfrac{(\uparrow^{A_{n-1}.A_n.\Gamma}_{A_n.\Gamma} \circ \uparrow^{A_{n-2}\ldots\Gamma}_{A_{n-1}.A_n.\Gamma})^{A_{n-2}\ldots\Gamma}_{A_n.\Gamma}\ ,\quad \textit{(shift)}\uparrow^{A_{n-3}\ldots A_n.\Gamma}_{A_{n-2}\ldots\Gamma}}{\vdots}}
$$

$$
\textit{(comp)}\dfrac{\qquad}{(\cdots(\uparrow^{A_{n-1}.A_n.\Gamma}_{A_n.\Gamma} \circ \uparrow^{A_{n-2}\ldots\Gamma}_{A_{n-1}.A_n.\Gamma})^{A_{n-2}\ldots\Gamma}_{A_n.\Gamma} \circ \cdots)^{A_1\ldots A_n.\Gamma}_{A_n.\Gamma}\ ,\quad \textit{(var)}\mathtt{1}^{A_n.\Gamma}_{A_n}}
$$

$$
\textit{(clos)}\dfrac{\qquad}{(\mathtt{1}^{A_n.\Gamma}_{A_n}[(\cdots(\uparrow^{A_{n-1}.A_n.\Gamma}_{A_n.\Gamma} \circ \uparrow^{A_{n-2}\ldots\Gamma}_{A_{n-1}.A_n.\Gamma})^{A_{n-2}\ldots\Gamma}_{A_n.\Gamma} \circ \cdots)^{A_1\ldots A_n.\Gamma}_{A_n.\Gamma}])^{A_1\ldots A_n.\Gamma}_{A_n}}
$$

In [8] as well as in [6] all the development of the ELAN implementation of the method is related to the sole de Bruijn index $\mathtt{1}$, the shift operator $\uparrow$ and composition, which makes that approach inefficient when compared with ours.

Another problem in the decoration of substitution objects of the $\lambda\sigma$-calculus is that they are decorated with two contexts that are lists of types. While the main marks in the decoration of a term object are a sole context and its type. This makes decorations of λs_e-terms cheaper than those of $\lambda\sigma$-terms.

As previously mentioned, decoration of expressions and subexpressions is only done at the beginning of the unification process, since the λs_e and λs_e-unification rules are supposed decorated and, of course, they preserve types and contexts. Initial decoration can be done using the algorithm in Table 4. This algorithm is based on a straightforward propagation of the decoration of subterms composing a λs_e-term according to the decorated λs_e-typing rules. The kernel of the algorithm consists of a set of rules that propagate contexts and types between the decoration marks of the term processed conforming to its structure outermost (named $\Downarrow$) and innermost (named $\Uparrow$).

The above algorithm runs in time linear on the size of the λs_e-term and on the magnitude of its de Bruijn indices. For this algorithm one needs the main context, but linear algorithms can be built without it, based on the decomposition of the undecorated input into a first order unification problem of type and context expressions generated from the typing rules of the λs_e-calculus.

Our previous remarks point out the advantage of λs_e in using all de Bruijn indices, which avoids quadratic decorations in the size of the input as in the $\lambda\sigma$-HOU approach. In fact, we can take again $\mathtt{1}[\uparrow^{n-1}]$ of Example 5. Its explicit decoration is, of course, quadratic. Consequently we can state the following.

Lemma 3 (Linear against quadratic decorations) *Pre-cooked λ-terms in the λs_e-calculus have linear decorations on the size of the λ-terms and the magnitude of their de Bruijn indices, while in $\lambda\sigma$ these decorations are quadratic.*

Proof. The proof is done by induction on the structure of terms based on the decorated typing rules for the simply-typed $\lambda\sigma$ and λs_e calculi.

□

Note that the size of decorated λ-terms increases in an inadequate way when normalizing via $\lambda\sigma$, because the decoration of substitution objects is not only expensive but also expansive in size and time. Furthermore, this expansion of decorated terms in the $\lambda\sigma$-HOU approach is independent of the use of other de Bruijn indices than $\mathtt{1}$ itself, and depends only on the use of substitution objects.

Example 6 $((\lambda_A.((\lambda_A.X_A^{A.A.A.\Gamma})_{A\to A}^{A.A.\Gamma}\ \mathtt{1}_A^{A.A.\Gamma})_A^{A.A.\Gamma})_{A\to A}^{A.\Gamma}\ \ \mathtt{1}_A^{A.\Gamma})_A^{A.\Gamma}$ *is the decorated version of* $(\lambda_A.(\lambda_A.X\ \ \mathtt{1})\ \ \mathtt{1})$. *Compare the corresponding decorated terms in the* λs_e- *and* $\lambda\sigma$-*calculi after two applications of Beta.*
In the λs_e: $\to_{Beta}$ $((\lambda_A.(X_A^{A.A.A.\Gamma}\sigma^1\mathtt{1}_A^{A.A.\Gamma})_A^{A.A.\Gamma})_{A\to A}^{A.\Gamma}\ \ \mathtt{1}_A^{A.\Gamma})_A^{A.\Gamma}$
$\to_{Beta}$ $((X_A^{A.A.A.\Gamma}\sigma^1\mathtt{1}_A^{A.A.\Gamma})_A^{A.A.\Gamma}\sigma^1\mathtt{1}_A^{A.\Gamma})_A^{A.\Gamma}$.
In $\lambda\sigma$: $\to_{Beta}$ $((\lambda_A.(X_A^{A.A.A.\Gamma}[(\mathtt{1}_A^{A.A.\Gamma}.id_{A.A.\Gamma}^{A.A.\Gamma})_{A.A.A.\Gamma}^{A.A.\Gamma}])_A^{A.A.\Gamma})_{A\to A}^{A.\Gamma}\ \ \mathtt{1}_A^{A.\Gamma})_A^{A.\Gamma}\to_{Beta}$ $((X_A^{A.A.A.\Gamma}[(\mathtt{1}_A^{A.A.\Gamma}.id_{A.A.\Gamma}^{A.A.\Gamma})_{A.A.A.\Gamma}^{A.A.\Gamma}])_A^{A.A.\Gamma}[(\mathtt{1}_A^{A.\Gamma}.id_{A.\Gamma}^{A.\Gamma})_{A.A.\Gamma}^{A.\Gamma}])_A^{A.\Gamma}$.

Table 4: Type checking / decorating algorithm for the λs_e-calculus

INPUT: a a λs_e-term and Γ a context.
OUTPUT: If a is well-typed in Γ then a corresponding decorated term a', whose main context is Γ. Else report that a is ill-typed in Γ.
NOTATION: $\perp$ denotes unknown types and contexts.
ALGORITHM: Initially, a is decorated in such a way that the sole context known is its main one marked as Γ. All other types and contexts in the decoration of a are marked as $\perp$. Afterwards, apply nondeterministically to the decorated ter m the following rules until an irreducible term is obtained.

(Varn)	$\mathtt{n}_\perp^{A_1.\dots.A_n.\Gamma}\to\mathtt{n}_{A_n}^{A_1.\dots.A_n.\Gamma}$
($\lambda-\Downarrow$)	$(\lambda_A.a_\perp^\perp)_\perp^\Gamma\to(\lambda_A.a_\perp^{A.\Gamma})_\perp^\Gamma$
($\lambda-\Uparrow$)	$(\lambda_A.a_B^{A.\Gamma})_\perp^\Gamma\to(\lambda_A.a_B^{A.\Gamma})_{A\to B}^\Gamma$
(app-$\Downarrow$)	$(a_\perp^\perp\ \ b_\perp^\perp)_\perp^\Gamma\to(a_\perp^\Gamma\ \ b_\perp^\Gamma)_\perp^\Gamma$
(app-$\Uparrow$)	$(a_{A\to B}^\Gamma\ \ b_A^\Gamma)_\perp^\Gamma\to(a_{A\to B}^\Gamma\ \ b_A^\Gamma)_B^\Gamma$
($\sigma-\Downarrow$)	$(a_\perp^\perp\sigma^i b_\perp^\perp)_\perp^\Gamma\to(a_\perp^{\Gamma_{<i}.\perp.\Gamma_{\geq i}}\sigma^i b_\perp^{\Gamma_{\geq i}})_\perp^\Gamma$
($\sigma-\Rightarrow$)	$(a_\perp^{\Gamma_{<i}.\perp.\Gamma_{\geq i}}\sigma^i b_B^{\Gamma_{\geq i}})_\perp^\Gamma\to(a_\perp^{\Gamma_{<i}.B.\Gamma_{\geq i}}\sigma^i b_B^{\Gamma_{\geq i}})_\perp^\Gamma$
($\sigma-\Uparrow$)	$(a_A^{\Gamma_{<i}.B.\Gamma_{\geq i}}\sigma^i b_B^{\Gamma_{\geq i}})_\perp^\Gamma\to(a_A^{\Gamma_{<i}.B.\Gamma_{\geq i}}\sigma^i b_B^{\Gamma_{\geq i}})_A^\Gamma$
($\varphi-\Downarrow$)	$(\varphi_k^i a_\perp^\perp)_\perp^\Gamma\to(\varphi_k^i a_\perp^{\Gamma_{\leq k}.\Gamma_{\geq k+i}})_\perp^\Gamma$
($\varphi-\Uparrow$)	$(\varphi_k^i a_A^{\Gamma_{\leq k}.\Gamma_{\geq k+i}})_\perp^\Gamma\to(\varphi_k^i a_A^{\Gamma_{\leq k}.\Gamma_{\geq k+i}})_A^\Gamma$
(Meta)	$X_\perp^{\Gamma_X}\to X_{A_X}^{\Gamma_X}$

Finally, if the main type of the resulting decorated term a' is known then return a'. Else report that a is ill-typed under context Γ.

This expansion problem in $\lambda\sigma$ results from the fact that some rules used in the generation of substitution objects increase the number of subterms which are substitution objects. In Example 6, we only used the *Beta* rule of $\lambda\sigma$ (i.e., $(\lambda_A.a\ \ \ b)\to a[b.id]$) which generates two new substitution subterms to be marked in a decorated term: *id* and *b.id*, while for the *Beta* rule of λs_e,

$(\lambda_A.a\ \ b) \to a\sigma^1 b$, the number of subterms is reduced by one. Critical is the case of the *Abs* rule of $\lambda\sigma$, $(\lambda_A.a)[s] \to \lambda_A.a[\mathtt{1}.(s\circ\uparrow)]$, that enlarges the number of subterms to be marked in decorated terms from four to eight. Rules that enlarge the number of subterms to be decorated in λs_e are σ*-app-*, φ*-app-*, σ*-*σ*-* and φ*-*σ*-transition*; i.e., all those related to the *App* rule of $\lambda\sigma$, that enlarges the number of subterms to be decorated from five to seven.

All the rules of the λs_e-calculus are supposed decorated. For example, the Eta rule has the following form: *(Eta)* $(\lambda_A.(a^{A.\Gamma}_{A\to B}\ \ \mathtt{1}^{A.\Gamma}_{A})^{A.\Gamma}_{B})^{\Gamma}_{A\to B} \to b^{\Gamma}_{A\to B}$ if $a^{A.\Gamma}_{A\to B} =_{s_e} (\varphi^2_0\, b^{\Gamma}_{A\to B})^{A.\Gamma}_{A\to B}$. Except for this rule, application of the rules of the λs_e-calculus is easy to decide: rules are either non-conditional or have simple arithmetic conditions that can be resolved via any arithmetic deduction algorithm usually built-in between any interesting programming language.

The test for applying the *Eta* rule can be implemented according to the correspondence between the two *Eta* rules and following the idea suggested for the $\lambda\sigma$-HOU approach in [6]. We can extend the language of the λs_e-calculus with a dummy symbol $\diamond$ and verify for occurrences of this symbol after s_e-normalizing the term $(a^{A.\Gamma}_{A\to B}\sigma^1\diamond^{\Gamma}_{A})^{\Gamma}_{A\to B}$. In the case that the previous term has no occurrences of $\diamond$ the *Eta* rule applies being the reduct that s_e-normalization. In practice we have the easy to implement rule:

$$\textit{(Eta)}\ \ (\lambda_A.(a^{A.\Gamma}_{A\to B}\ \ \mathtt{1}^{A.\Gamma}_{A})^{A.\Gamma}_{B})^{\Gamma}_{A\to B} \ \to\ s_e\textit{-normalization}((a^{A.\Gamma}_{A\to B}\sigma^1\diamond^{\Gamma}_{A})^{\Gamma}_{A\to B})$$
$$\text{if } \diamond \text{ does not occur in this term}$$

Lemma 4 *The previous implementation of the Eta rule is correct.*

Proof. (Sketch). Note firstly that $(a^{A.\Gamma}_{A\to B}\sigma^1\diamond^{\Gamma}_{A})^{\Gamma}_{A\to B}$ results from *Beta* reduction of $((\lambda_A.a^{A.\Gamma}_{A\to B})^{\Gamma}_{A\to A\to B}\ \ \diamond^{\Gamma}_{A})^{\Gamma}_{A\to B}$. After propagating the σ operator all de Bruijn indices in the term are decremented by one except those corresponding to the variable of the outermost abstractor which are replaced with $\varphi^i_0\diamond$. This is proved by induction on the structure of terms and the superscript of the σ operator that is incremented mainly via the σ*-*λ*-transition* rule. Terms of the form $\varphi^i_0\diamond$ are obtained by applying the σ*-destruction rule.*

Secondly, notice that in the case that no occurrences of $\diamond$ remain in the resulting term, by incrementing all de Bruijn indices by one we obtain a term that is s_e equivalent to $a^{A.\Gamma}_{A\to B}$. This corresponds to the condition of the original *Eta* rule, since the application of φ^2_0 to λ-terms increments by one all de Bruijn indices. This can be proved by induction on the structure of terms.

□

Turning back to $\lambda\sigma$-HOU [6], the condition in the implementation of the *Eta* rule is: "*if* $\diamond$ *doesn't occur in the* σ*-normalization*$((a^{A.\Gamma}_{A\to B}[(\diamond^{\Gamma}_{A}.id^{\Gamma}_{\Gamma})^{\Gamma}_{A.\Gamma}])^{\Gamma}_{A\to B})$."

This implementation is less efficient than in the λs_e-calculus because of the use of substitution objects in the $\lambda\sigma$-calculus. This is a simple consequence of the fact that when propagating the above substitution objects between the structure of $a^{A.\Gamma}_{A\to B}$ we need to apply the rules *Abs* and *App* that are expansive, as mentioned early. More precisely, the rule *Abs*, $(\lambda_A.a)[s] \to \lambda_A.(a[\mathtt{1}.(s\circ\uparrow)])$, enlarges the number of substitution objects to be marked in decorated terms from one (s) to four: s, $\uparrow$, $s\circ\uparrow$, and $\mathtt{1}.(s\circ\uparrow)$; and the rule *App*, $(a\ \ b)[s] \to (a[s]\ \ b[s])$, from one to two. In contrast, in the λs_e-calculus the corresponding propagation of the σ operator is executed by applying the rules σ-λ-*transition* and σ-*app-transition*. The σ-λ-*transition*, $(\lambda_A.a)\sigma^i b \to \lambda_A.a\sigma^{i+1}b$, does not enlarge the number of subterms to be marked. And the σ-*app-transition*, $(a_1\ \ a_2)\sigma^i b \to (a_1\sigma^i b\ \ a_2\sigma^i b)$, increases the number of subterms to be marked by two as the *App* rule, but without including substitution objects.

5 Conclusions

Following the $\lambda\sigma$-HOU of [8], we have developed a pre-cooking translation that maps 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.

Our pre-cooking translation transcribes a term a by replacing each occurrence of a meta-variable X with $\varphi^{k+1}_0 X$ 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 [8] transcribes each occurrence of a de Bruijn index $\mathtt{n}$ in a into $\mathtt{1}[\uparrow^{n-1}]$. Conformity of the two pre-cooking translations is therefore evident. But our proofs differ from those of [8] 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 1) and the proof of Proposition 2 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 is the case in the $\lambda\sigma$-HOU approach.

Pre-cooking is complemented with a *back* translation that enables the reconstruction of solved forms of unification problems in λs_e into a description of solutions of the corresponding HOU problems in the pure λ-calculus.

By comparing direct (*naive*) implementations of our method and that of the $\lambda\sigma$-HOU of [6], we observed that pre-cooked λ-terms in λs_e have linear decorations on the size of the λ-terms and the magnitude of their de Bruijn indices, while in $\lambda\sigma$ these decorations are quadratic. For this, we make no considerations about the use of efficient data structures. For a reasonable implementation of the $\lambda\sigma$-HOU approach, a variation of the $\lambda\sigma$-calculus which includes all de Bruijn indices should be used, but according to the implementation of that method in [6], this has remained inefficient. From the theoretical point of view, our approach is the first to treat this problem in a natural way, thanks to the simple syntax of the λs_e-calculus where all de Bruijn indices are included.

But it is not the sole use of all de Bruijn indices that makes the λs_e approach more efficient. Another problem in the decoration of substitution objects of the $\lambda\sigma$-calculus is that they are decorated with two contexts that are lists of types. While the main marks in the decoration of a term object are a sole context and its type. This makes decorations of λs_e-terms smaller than those of $\lambda\sigma$-terms. Moreover, the size of decorated λ-terms increases in an inadequate way when normalizing via the $\lambda\sigma$-calculus, because some rules of $\lambda\sigma$ are expensive in that they enlarge the number of substitution objects to be marked in decorated terms. The lack of substitution objects in λs_e makes the proofs easier.

Much work remains to be done and a prototype implementation of this method is necessary. It would be relevant to consider whether a specialization of the λs_e-HOU for the important decidable and unitary fragment of the higher-order patterns, as it has been done for $\lambda\sigma$ in [9], has practical benefits. Furthermore, a formal distinction, from the practical point of view, between the λs_e-calculus (and our procedure) and the *suspension* calculus developed in [14] (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 [14] have correlated nice properties. Recently, it has been proved that the λs_e is more efficient than the suspension calculus in simulating a sole step of β-reduction [1], but in contrast the suspension calculus appears more adequate for simulating simultaneous steps of β-reduction as pointed out in [13]. Studying these differences is important for estimating the appropriateness of the λs_e-HOU approach in that practical framework.

References

[1] Ayala-Rincón, M.; de Moura, F. C.; Kamareddine, F., *Comparing Calculi of Explicit Substitutions with Eta-reduction*, In Proceedings Ninth Workshop on Logic, Language, Information and Computation (WoLLIC 2002), volume 67 of *ENTCS*. Elsevier Science Publishers, (2002).

[2] Ayala-Rincón, M.; Kamareddine, F., *Strategies for Simply-Typed Higher Order Unification via* λs_e*-Style of Explicit Substitution*, In R. Kennaway, editor, Third International Workshop on Explicit Substitutions Theory and Applications to Programs and Proofs (WESTAPP 2000), Norwich, England, (2000), 3-17.

[3] Ayala-Rincón, M.; Kamareddine, F., *Unification via the* λs_e*-Style of Explicit Substitution*, The Logical Journal of the Interest Group in Pure and Applied Logics, 9(4), (2001), 489–523.

[4] Baader, F.; Nipkow, T., *Term Rewriting and All That*, Cambridge, (1998).

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

[6] Borovanský, P., *Implementation of Higher-Order Unification Based on Calculus of Explicit Substitutions*, In M. Bartošek, J. Staudek, and J. Wiedermann, editors, Proceedings of the SOFSEM'95: Theory and Practice of Informatics, volume 1012 of *LNCS*, Springer-Verlag, (1995), 363-368.

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

[8] Dowek, G.; Hardin, T.; Kirchner, C., *Higher-order Unification via Explicit Substitutions*, Information and Computation, 157(1/2), (2000), 183–235,

[9] Dowek,G.; Hardin, T.; Kirchner, C.; Pfenning, F., *Unification via Explicit Substitutions: The Case of Higher-Order Patterns*, In Proc. of the Joint Int. Conf. and Symposium on Logic Programming, MIT press, (1996), 259-273.

[10] Huet, G. P., *A Unification Algorithm for Typed λ-Calculus*, Theoretical Computer Science, 1, (1975), 27–57.

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

[12] Kamareddine, F.; Ríos, A., *Relating the $\lambda\sigma$- and λs-Styles of Explicit Substitutions*, Journal of Logic and Computation, 10(3), (2000), 349–380.

[13] Liang, C.; Nadathur, G., *Tradeoffs in the Intensional Representation of Lambda Terms*, In S. Tison, editor, Rewriting Techniques and Applications (RTA 2002), volume 2378 of *LNCS*, Springer-Verlag, (2002), 192-206.

[14] Nadathur, G.; Wilson, D. S., *A Notation for Lambda Terms A Generalization of Environments*, Theoretical Computer Science, 198, (1998), 49–98.

[15] Snyder, W.; Gallier, J., *Higher-Order Unification Revisited: Complete Sets of Transformations*, Journal of Symbolic Computation, 8, (1989), 101–140.

M. Ayala-Rincón
Departamento de Matemática
Universidade de Brasília
70910-900 Brasília, Brasil
`ayala@mat.unb.br`

F. Kamareddine
Scholl of Mathematical and
Computer Sciences
Heriot-Watt University, Ricccarton
Edinburgh EH14 4AS, Scotland
`fairouz@macs.hw.ac.uk`