On Applying the λs_e-Style of Unification for Simply-Typed Higher Order Unification in the Pure λ-Calculus

MAURICIO AYALA-RINCÓN , *Departamento de Matemática, Universidade de Brasília, 70910-900 Brasília D.F., Brasil* ayala@mat.unb.br

FAIROUZ KAMAREDDINE , *Department of Computing and Electrical Engineering, Heriot-Watt University, Riccarton, Edinburgh EH14 4AS, Scotland* fairouz@cee.hw.ac.uk

Abstract

Dowek, Hadin and Kirchner developed a higher order unification (HOU) method based on the $\lambda\sigma$-style of explicit substitutions. The novelty of this method rests on the possibility to resolve HOU problems by first order unification. This is achieved via a *pre-cooking* translation of the HOU problem into a first order unification problem of the language of the $\lambda\sigma$-calculus. Solutions to the first order unification problem are then translated *back* into the range of the *pre-cooking* translation and subsequently to solutions of the original problem into the language of the λ-calculus. Recently we study unification in the λs_e-style of explicit substitutions. It is claimed 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 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. Their correctness and completeness are shown and additionally we show why avoiding the use of substitution objects makes λs_e-HOU more efficient than $\lambda\sigma$-HOU.

Keywords: Higher order unification, explicit substitutions, λ-calculus, type and rewriting theory

1 Background

HOU via explicit substitutions [6], as mentioned in the abstract, is illustrated by Figure 1. Here we show how to apply our λs_e-unification method in [2] for resolving HOU problems. 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, $\lambda\sigma$ uses only one de Bruijn index (1) and builds the others by operations in the calculus. λs_e uses all the de Bruijn indices. Another difference between both calculi is that the λs_e-calculus attempts to remain as close as possible to the syntax of the λ-calculus and hence only adds updating and substitutions as two new concepts and keeps the unique sort of term objects; $\lambda\sigma$ adds various categorical operators like composition, consing, and lifting and has two sorts of objects: terms and substitutions. 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 [5]. It should be stressed that $\lambda\sigma$ and λs_e are two different styles of explicit substitutions 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. A full version of the article containing all proofs can be found at www.cce.hw.ac.uk/ultra/publications.html.

For a set of operators $\mathcal{F}$, we assume familiarity with the notions of an $\mathcal{F}$-algebra and of 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 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 [4] and with the basic notions of rewriting theory as in [3]. For a *reduction relation* $\rightarrow_R$ over a set A, we denote with $\rightarrow_R^*$ the *reflexive and transitive closure* of $\rightarrow_R$. 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 the domain $\mathcal{T}(\mathcal{F},\mathcal{X})$ is called the **grafting** of θ. This notion is usually called first order

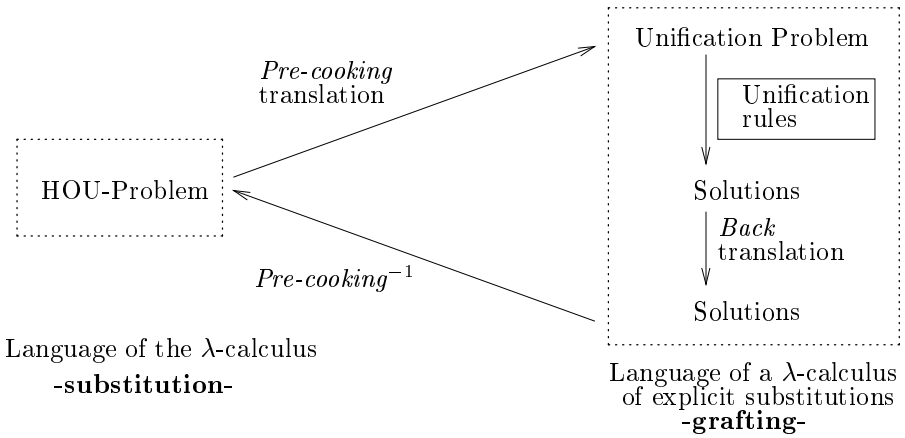

FIG. 1. HOU method via calculi of explicit substitutions

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 $Dom(\theta) = \{X \mid X\theta \neq X, X \in \mathcal{X}\}$. 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: $\{X/X\theta \mid X \in Dom(\theta)\}_g$.

We assume familiarity with the $\lambda\sigma$- ($\cdot, \circ, []$ and $\uparrow$ operators) and λs_e-calculi (φ and σ operators and skeleton notation ψ), their typed versions and their normal form (nf, lnf and η-nf) characterizations as in [2].

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 $\Lambda(\mathcal{V})$ 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 $\mathcal{X}$. 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 is the one of de Bruijn indices [11] 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 by:
$a ::= \mathtt{n} \mid X \mid (a\ \ a) \mid \lambda.a$ where $X \in \mathcal{X}$ and $\mathtt{n} \in \mathbb{N} \setminus \{0\}$.

DEFINITION 1.1
Let $a \in \Lambda_{dB}(\mathcal{X})$, $i \in \mathbb{N}$. The **i-lift** of a, a^{+i}, is defined by cases 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 1.2
The application of the **substitution** with b of $n \in \mathbb{N}\backslash\{0\}$ on a term a in $\Lambda_{dB}(\mathcal{X})$, denoted $\{n/b\}a$, is defined inductively as: **a)** $\{n/b\}X = X$, for $X \in \mathcal{X}$; **b)** $\{n/b\}(a_1\ \ a_2) = (\{n/b\}a_1\ \ \{n/b\}a_2)$;
c) $\{n/b\}\lambda.a_1 = \lambda.\{n+1/b^+\}a_1$; **d)** $\{n/b\}\mathtt{m} = \begin{cases} \mathtt{m}-1, & \text{if } m > n; \\ b, & \text{if } m = n; \\ \mathtt{m}, & \text{if } m < n \end{cases}$ when $m \in \mathbb{N}$.

DEFINITION 1.3
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: **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 valuation $\{X_1/a_1^+, \ldots, X_n/a_n^+\}$ and its associated substitution.

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, 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 $\exists b\ b^+ = a$. We use $=_{\beta\eta}$ to denote the congruence generated by β- and *eta*-reduction.

2 Unification in the λs_e-calculus

In this section we review the λs_e-unification method of [2]. Normal form characterizations (cf. normal form (nf) and long normal forms (lnf)), jointly with WN and CR properties are the essential requirements to develop a unification method for the λs_e-calculus, which can be applied to HOU in the λ-calculus.

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 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 $\exists \vec{w} \bigwedge_{i\in I} s_i =^?_{\mathcal{A}} t_i$ 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.

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 $a =^?_{\lambda s_e} a$.

In [2] we present a set of rewrite rule schemata used to simplify unification problems. The objective of applying these rules is to obtain a description of the set of unifiers. Since λs_e is CR and WN [8], 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 $(\mathtt{k}\ b_1 \ldots b_p)$ or $a\sigma^i b$ or $\varphi^i_k a$, where in the first case $\mathtt{k}$ can be omitted and p is zero. From these rules *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 $A \to B$, where Y is a new variable of type B. Rules *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. In (the notation of) the $\lambda\sigma$-calculus, 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. The corresponding rule has the analogous role for λs_e-unification problems.

EXAMPLE 2.1
Let $(\lambda.(\lambda.(X\ \ 2)\ \ 1)\ \ Y) =^?_{\lambda s_e} (\lambda.(Z\ \ 1)\ \ U)$ be a unification problem, where X, Y, Z and U are meta-variables. 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 after *Dec-App*, *Dec-*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, applying twice *Exp-App* and *Replace* we obtain $((\lambda.X')\sigma^2 Y)\sigma^1(\varphi_0^1 Y) =^?_{\lambda s_e} (\lambda.Z')\sigma^1 Y \wedge Y =^?_{\lambda s_e} U \wedge X =^?_{\lambda s_e} \lambda.X' \wedge Z =^?_{\lambda s_e} \lambda.Z'$. Finally, after *Normalize* and *Dec-*λ we obtain $(X'\sigma^3 Y)\sigma^2(\varphi_0^1 Y) =^?_{\lambda s_e} Z'\sigma^2 Y \wedge Y =^?_{\lambda s_e} U \wedge X =^?_{\lambda s_e} \lambda.X' \wedge Z =^?_{\lambda s_e} \lambda.Z'$. Solutions are built as $\{Y/X_1, U/X_1\}$ union solutions for X and Z obtained by the first equation. Equations as the first one, that are called *Flex-Flex*, are related with the notion of *pre-unifiers* in [7]. In this case we can take, for instance, $\{Y/X_1, U/X_1\} \bigcup \{X/\lambda.\mathtt{n}+1, Z/\lambda.\mathtt{n}\}$, where $n > 2$. •

DEFINITION 2.2
A unification system P is a λs_e-**solved form** if all its meta-variables are of atomic type and it is a conjunction of non trivial equations of the following forms:

(*Solved*) $X =^?_{\lambda\sigma} a$, where the variable 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: $\psi_{i_p}^{j_p} \ldots \psi_{i_1}^{j_1}(X, a_1, \ldots, a_p) =^?_{\lambda s_e} \psi_{k_q}^{l_q} \ldots \psi_{k_1}^{l_1}(Y, b_1, \ldots, b_q)$ with X, Y of atomic type.

In [2] it was proved that: 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) 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.

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 2.3 (Correctness and completeness [2])
The λs_e-unification rules are correct and complete.

An analogous unification strategy to the one for $\lambda\sigma$ presented in [6] applies as well 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 in a certain way independent of the calculi [1].

3 HOU in the pure λ-calculus

[2], reviewed in Section 2, dealt with half of the box on the right hand side of Figure 1. That is, only with the λs_e-unification method. For applying this method to HOU in λ-calculus we need to complete the diagram by providing the pre-cooking and Back translations, show their correctness and completeness and establish the applicability of λs_e-unification for HOU in pure λ-calculus.

Initially we present one example on how to apply our λs_e-unification method in order to solve HOU problems in the pure λ-calculus. Then we present adequate pre-cooking and back translations (see Figure 1).

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 in Definitions 1.2 and 1.3, 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.

Before defining our pre-cooking translation from $\Lambda_{dB}(\mathcal{X})$ into the λs_e-calculus we motivate how the searching for substitution solutions of a HOU problem $a =^?_{\beta\eta} b$ corresponds to the searching for grafting solutions of a unification problem in λs_e.

EXAMPLE 3.1
Consider the HOU problem $\lambda.(X\ \ 2) =^?_{\beta\eta} \lambda.2$, where 2 and X are of type A and $A \rightarrow A$, 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 $\lambda.(X[\uparrow]\ \ 2)$. This correspondence results from

one between both *Eta* rules (i.e., between $b[\uparrow] = a$ and $\varphi_0^2 b = a$). 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, we obtain $(\exists Y(\varphi_1^2 Y)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y) \wedge (Y =^?_{\lambda s_e} 1 \vee Y =^?_{\lambda s_e} 2)$ by applying*Exp-app*; by applying *Replace*: $((\varphi_1^2 1)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.1) \vee ((\varphi_1^2 2)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.2)$; and 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.

To complete the analysis note that Definitions 1.2, 1.3 and β-reduction in $\Lambda_{dB}(\mathcal{X})$ give $\{X/\lambda.1\}(\lambda.(X\ \ 2))$ $= \lambda.(\{X/(\lambda.1)^+\}(X)\ \ 2) = \lambda.(\lambda.1^{+1}\ \ 2) = \lambda.(\lambda.1\ \ 2) =_\beta \lambda.2$ and $\{X/\lambda.2\}(\lambda.(X\ \ 2)) = \lambda.(\{X/(\lambda.2)^+\}(X)\ \ 2)$ $= \lambda.(\lambda.2^{+1}\ \ 2) = \lambda.(\lambda.3\ \ 2) =_\beta \lambda.\{1/2\}(3) = \lambda.2$. •

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 $\varphi_0^{k+1}X$, 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 [6] transcribes all occurrences of de Bruijn indices $\mathtt{n}$ into $\mathtt{1}[\uparrow^{n-1}]$ 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.

DEFINITION 3.2 (Pre-cooking)
Let $a \in \Lambda_{dB}(\mathcal{X})$ such that $\Gamma \vdash_{\Lambda_{dB}(\mathcal{X})} a : T$. With 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) $PC(X, n) = \begin{cases} X, \text{ if } n = 0 \\ \varphi_0^{n+1}X, \text{ otherwise} \end{cases}$

LEMMA 3.3 (Type preservation)
If $\Gamma \vdash_{\Lambda_{dB}(\mathcal{X})} a : T$, then $\Gamma \vdash_{\lambda s_e} a_{pc} : T$.

The following proposition which relates substitution in $\Lambda_{dB}(\mathcal{X})$ and grafting in λs_e, justifies pre-cooking.

PROPOSITION 3.4 (Semantics of pre-cooking)
Let $a, b_1, \ldots, b_p$ be terms of $\Lambda_{dB}(\mathcal{X})$. We have:

$$(a\{X_1/b_1, \ldots, X_p/b_p\})_{pc} = a_{pc}\{X_1/b_{1_{pc}}, \ldots, X_p/b_{p_{pc}}\}_g.$$

In contrast to the corresponding proof in [6], where substitution objects are necessary for proving the critical case of $a = X$ (i.e., substitutions of the form $[1 \ldots k.\ \uparrow^{i+k}]$) 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 3.5
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 $a_{pc} =_{\lambda s_e} b_{pc}$.

Again, our proof differs from the corresponding one in [6] 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 3.6 (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 $M_1, \ldots, M_p$ such that $a_{pc}\{X_1/M_1, \ldots, X_p/M_p\}_g$ $=_{\lambda s_e} b_{pc}\{X_1/M_1, \ldots, X_p/M_p\}_g$.

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 3.6), 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 following similar steps to those of the $\lambda\sigma$-HOU approach in [6] but adapted to the λs_e-calculus.

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

(Anti-Exp-λ) $P \to \exists Y\,(P \wedge X =^?_{\lambda s_e} (\varphi^2_0 Y\ \ 1))$ 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 an environment $A.\Gamma$

PROPOSITION 3.7 (Correctness and completeness of the *Anti*-rules)
The λs_e-unification rules together with the *Anti-Exp-λ* and *Anti-Dec-λ* rules are correct and complete.

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

PROPOSITION 3.8
Let $a =^?_{\beta\eta} b$ be a HOU problem well-typed in an environment 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 the following invariants: 1) if an equation is well-typed in environment Δ, then Δ is an extension of Γ;
2) for every variable Y, its environment Γ_Y is an extension of Γ;
3) for every subterm $\psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(X, a_1, \dots, a_p)$ in P we have $p \leq |\Gamma_Y| - |\Gamma| + 1$.

PROPOSITION 3.9 (Building Back Pre-cooking images)
Let $a =^?_{\beta\eta} b$ be a HOU problem and P an equational problem derived from its pre-cooking by using the λs_e-unification rules. The system resulting from normalization of P by applying the *Back* strategy is the pre-cooking of a problem in the λ-calculus.

Using previous correctness and completeness results we obtain the following Corollary and Theorem.

COROLLARY 3.10 (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 [12]) and the solutions of R are solutions of the original HOU problem.

THEOREM 3.11 (Completeness of the construction of solutions)
Let $a =^?_{\beta\eta} b$ a HOU problem such that its pre-cooking is well-typed in the environment 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.

4 Considerations about the implementation

We precise here why the use of the sole de Bruijn index `1` and of substitution objects make the $\lambda\sigma$-HOU approach less efficient than the λs_e-HOU one.

For the sake of clarity, we have omitted above both types and environments. But for the analysis of the HOU method above it is necessary to know both the types and environments of all subexpressions during the unification process. Therefore terms "decorated" with types and environments for all their subterms are necessary for any reasonable implementation. The general idea is to assign types and environments to all subexpressions at the beginning of the unification process and to maintain this notational discipline during the process via decorated versions of the λs_e-calculus, the λs_e-typing rules and, of course, the λs_e-unification rules. We present the decorated version of the typing rules for the λs_e-calculus in the Table 1.

TABLE 1. Undecorated and decorated typing rules for the λs_e-calculus

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

The typing rules *Var* and *Varn* can be reduced to a sole decorated rule of the form $\mathtt{n}_{A_n}^{A_1\ldots A_n.\Gamma}$ 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 environment Γ_X and a unique 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 environments[1]. Examples of these rules are: *(Shift)* $\uparrow_{\Gamma}^{A.\Gamma}$; *(Comp)*$s_{\Gamma}^{\Theta}\ ,\ t_{\Theta}^{\Delta} \vdash (s_{\Gamma}^{\Theta} \circ t_{\Theta}^{\Delta})_{\Gamma}^{\Delta}$; *(Clos)*$a_A^{\Delta}\ ,\ s_{\Delta}^{\Gamma} \vdash (a_A^{\Delta}[s_{\Delta}^{\Gamma}])_A^{\Gamma}$. This kind of explicit decoration was done for the $\lambda\sigma$-HOU approach in [5], 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 4.1
The decoration of $\mathtt{1}[\uparrow^{n-1}]$ uses quadratic space and time.

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

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

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

This, of course, could be improved in the $\lambda\sigma$-HOU approach, but as far as we know, improvements have not been incorporated. In [6] as well as in [5] all the development of the 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. Of course, we believe some improvements in this sense were done in the implementation of the $\lambda\sigma$-HOU, but from the theoretical point of view our approach is the first one that has treated this problem in a natural way, because in λs_e, all de Bruijn indices are included.

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

---

[1] This is denoted in the undecorated setting as $s \triangleright \Gamma$.

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 environments. Initial decoration can be done using the algorithm in Table 2. 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 environments and types between the decoration marks of the term processed conforming to its structure outermost (named as $\Downarrow$) and innermost (named as $\Uparrow$).

The previous algorithm runs in time linear on the size of the initial λs_e-term and on the magnitude of its de Bruijn indices. For this algorithm it is necessary to know the main environment, but linear algorithms can be built without such information, based on the decomposition of the undecorated input into a first order unification problem of type and environment 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 4.1. Its explicit decoration is, of course, quadratic. Consequently we can state the following.

LEMMA 4.2 (Linear versus 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.

Moreover, notice here that the size of decorated λ-terms increases in an inadequate way when normalizing via the $\lambda\sigma$-calculus, 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 1 itself, and depends only on the use of substitution objects.

EXAMPLE 4.3
To illustrate this consider the decorated λ-term $(\lambda_A.(\lambda_A.X\ \ \mathtt{1})\ \ \mathtt{1})$:
$((\lambda_A.((\lambda_A.X^{A.A.A.\Gamma}_A)^{A.A.\Gamma}_{A\to A}\ \ \mathtt{1}^{A.A.\Gamma}_A)^{A.A.\Gamma}_A)^{A.\Gamma}_{A\to A}\ \ \mathtt{1}^{A.\Gamma}_A)^{A.\Gamma}_A$ and compare the corresponding decorated terms in the λs_e- and $\lambda\sigma$-calculi after two applications of *Beta*. In the λs_e we have:
$\to_{Beta}\ ((\lambda_A.(X^{A.A.A.\Gamma}_A\sigma^1\mathtt{1}^{A.A.\Gamma}_A)^{A.A.\Gamma}_A)^{A.\Gamma}_{A\to A}\ \ \mathtt{1}^{A.\Gamma}_A)^{A.\Gamma}_A\ \to_{Beta}\ ((X^{A.A.A.\Gamma}_A\sigma^1\mathtt{1}^{A.A.\Gamma}_A)^{A.A.\Gamma}_A\sigma^1\mathtt{1}^{A.\Gamma}_A)^{A.\Gamma}_A$ and in the $\lambda\sigma$-calculus we have: $\to_{Beta}\ ((\lambda_A.(X^{A.A.A.\Gamma}_A[(\mathtt{1}^{A.A.\Gamma}_A.id^{A.A.\Gamma}_{A.A.\Gamma})^{A.A.\Gamma}_{A.A.A.\Gamma}])^{A.A.\Gamma}_A)^{A.\Gamma}_{A\to A}\ \mathtt{1}^{A.\Gamma}_A)^{A.\Gamma}_A\to_{Beta}$
$((X^{A.A.A.\Gamma}_A[(\mathtt{1}^{A.A.\Gamma}_A.id^{A.A.\Gamma}_{A.A.\Gamma})^{A.A.\Gamma}_{A.A.A.\Gamma}])^{A.A.\Gamma}_A[(\mathtt{1}^{A.\Gamma}_A.id^{A.\Gamma}_{A.\Gamma})^{A.\Gamma}_{A.A.\Gamma}])^{A.\Gamma}_A$. •

This expansion problem in the $\lambda\sigma$-calculus is a consequence of the fact that some rules used in the generation of substitution objects increase the number of subterms which are substitution objects. In Example 4.3, we only used the *Beta* rule of the $\lambda\sigma$-calculus(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 the λs_e-calculus, $(\lambda_A.a\ \ b)\to a\sigma^1 b$, the number of subterms is reduced by one. Critical is the case of the *Abs* rule of the $\lambda\sigma$-calculus, $(\lambda_A.a)[s]\to\lambda_A.a[1.(s\circ\uparrow)]$, 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 the λs_e are *σ-app-transition*, *φ-app-transition*, *σ-σ-transition* and *φ-σ-transition*; i.e., all those related to the *App* rule of the $\lambda\sigma$-calculus, 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 decorated 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 [5]. 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:
(Eta) $(\lambda_A.(a^{A.\Gamma}_{A\to B}\ \ \mathtt{1}^{A.\Gamma}_A)^{A.\Gamma}_B)^{\Gamma}_{A\to B}\ \to\ s_e\text{-}normalization((a^{A.\Gamma}_{A\to B}\sigma^1\diamond^{\Gamma}_A)^{\Gamma}_{A\to B})$ if $\diamond$ doesn't occur in this term.

LEMMA 4.4
The previous implementation of the *Eta* rule is correct.

TABLE 2. Type checking / decorating algorithm for the λs_e-calculus

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

(Varn)	$\mathtt{n}_{\perp}^{A_1.\ldots.A_n.\Gamma} \to \mathtt{n}_{A_n}^{A_1.\ldots.A_n.\Gamma}$
($\lambda-\Downarrow$)	$(\lambda_A.a_{\perp}^{\perp})_{\perp}^{\Gamma} \to (\lambda_A.a_{\perp}^{A.\Gamma})_{\perp}^{\Gamma}$
($\lambda-\Uparrow$)	$(\lambda_A.a_B^{A.\Gamma})_{\perp}^{\Gamma} \to (\lambda_A.a_B^{A.\Gamma})_{A\to B}^{\Gamma}$
(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'. Otherwise report that a is ill-typed under environment Γ.

Turning back to $\lambda\sigma$-HOU (see [5]), the condition in the implementation of the *Eta* rule is seen as: "*if* $\diamond$ *doesn't occur in the* σ*-normalization*$((a_{A\to B}^{A.\Gamma}[(\diamond_A^{\Gamma}.id_{\Gamma}^{\Gamma})_{A.\Gamma}^{\Gamma}])_{A\to B}^{\Gamma}$)"

This implementation is less efficient than in the λs_e-calculus and once more the problem depends on 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 $a_{A\to B}^{A.\Gamma}$ we need to apply the rules *Abs* and *App* that are expansive, as mentioned early. More precisely, the rule *Abs*, $(\lambda_A.a)[s] \to \lambda_A.(a[1.(s\circ\uparrow)])$, enlarges the number of substitution objects to be marked in decorated terms from one (s) to four: s, $\uparrow$, $s\circ\uparrow$, and $1.(s\circ\uparrow)$; 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 approach introduced in [6], we have developed 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.

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 [6] 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 the corresponding ones in [6] 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 3.4) and the proof of Proposition 3.5 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.

Furthermore, by comparing the implementation of our method and that of the $\lambda\sigma$-HOU given in [5], we observed that 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. For that, we don't make any consideration about 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 [5], this has remained inefficient. From the theoretical point of view, our approach is the first one that has treated this problem in a natural way, because of 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 environments that are lists of types. While the main marks in the decoration of a term object are a sole environment and its type. This makes decorations of λs_e-terms smaller than the ones 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 in the $\lambda\sigma$-calculus are expensive in that they enlarge the number of substitution objects to be marked in decorated terms. Also, the lack of substitution objects in λs_e makes the proofs easier.

Much work remains to be done and in particular, to be conclusive, a prototype implementation of this method is necessary. 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 [10, 9] (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 [10, 9] 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 the rewrite rules of Nadathur and Wilson. Establishing these connections is important for estimating the appropriateness of the λs_e-HOU approach in that practical framework.

References

[1] M. Ayala-Rincón and F. Kamareddine. 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)*, pages 3–17, Norwich, England, 2000.

[2] M. Ayala-Rincón and F. Kamareddine. Unification via the λs_e-Style of Explicit Substitution. *Logical Journal of the Interest Group in Pure and Applied Logics - IGPL*, 9(4):521–555, 2001.

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

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

[5] P. Borovanský. 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 *Lecture Notes on Computer Science*, pages 363–368. Springer Verlag, 1995.

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

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

[8] 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.

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

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

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

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