ON APPLYING THE $\lambda s_e$-STYLE OF UNIFICATION FOR SIMPLY-TYPED HIGHER ORDER UNIFICATION IN THE PURE $\lambda$-CALCULUS

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 $\lambda$-calculus. Recently we studied unification in the $\lambda s_e$-style of explicit substitutions which only uses one sort of objects: terms. We believe that $\lambda s_e$-unification enables quicker detection of redices and has a clearer semantics. In this paper, we provide a pre-cooking translation for applying $\lambda s_e$-unification to HOU in the $\lambda$-calculus. The pre-cooking jointly with a back translation complement the $\lambda s_e$-unification method. We establish correctness and completeness and show why avoiding the use of substitution objects makes $\lambda s_e$-HOU more efficient than $\lambda\sigma$-HOU.

Keywords: Unification, explicit substitutions, $\lambda$-calculus, type theory.

1 Background

HOU via explicit substitutions as in [6] is illustrated by Figure 1 where solving a higher-order unification problem in the $\lambda$-calculus amounts to the following:

1. The higher-order problem of the $\lambda$-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 precooking translation and then translated back into the $\lambda$-calculus.
Figure 1: HOU method via calculi of explicit substitutions

In [2] the lines of [6] were followed for formalizing a unification system based on the λsₖ-style of substitutions in which one could solve first-order unification problems in the λsₑ-calculus. However, in [2] it was informally claimed that λsₑ-unification has the advantages of enabling quicker detection of reducts and of having a clearer semantics than λσ-unification and it was only dealt with the second item of the above four mentioned steps followed in [6]. In this paper, we close this gap and fill the remaining steps. We provide a pre-cooking translation for applying λsₑ-unification to HOU in the λ-calculus. The pre-cooking jointly with a back translation complement our λsₑ-unification method. We establish the correctness and completeness of our pre-cooking and back translations.

The λσ- and the λsₑ-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 λσ and λsₑ:

- λσ uses only one de Bruijn index (1) and builds the others by operations in the calculus. λsₑ uses all the de Bruijn indices.

- The λsₑ-calculus remains close to the syntax of the λ-calculus by adding updating and substitutions as new concepts and keeping the unique sort of term objects; λσ adds various categorical operators like composition, consing, and lifting and has two sorts of objects: terms and substitutions.

In this paper, we focus on the advantages of using all de Bruijn indices and only term objects when implementing the λsₑ-HOU approach over λσ-HOU and its implementation as described in [5]. We show why avoiding the use of substitution objects makes λsₑ-HOU more efficient than λσ-HOU.

It should be stressed that λσ and λsₑ are non-isomorphic styles of explicit substitutions and hence reworking the HOU method in λsₑ is not a translation
of work already done in $\lambda \sigma$. Many rules and proofs of the $\lambda s_c$-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 $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $\text{var}(t)$ denotes the set of variables occurring in $t$. We assume familiarity with $\lambda$-calculus [4] and with basic rewriting [3]. 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, $\theta$, from its domain $\mathcal{X}$ to $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is called the grafting of $\theta$. This notion is usually called first order substitution and corresponds to simple substitution without renaming. As usual, valuations and their corresponding grafting valuations are denoted by the same symbol. The domain of a grafting $\theta$ is defined by $\text{Dom}(\theta) = \{X | X \theta \neq X, X \in \mathcal{X}\}$. A valuation and its corresponding grafting $\theta$ are explicitly denoted by $\theta = \{X/X \theta | X \in \text{Dom}(\theta)\}$. When necessary, explicit representations of graftings are differentiated from substitutions by a “$g$” subscript as in: $\{X/X \theta | X \in \text{Dom}(\theta)\}_g$.

We assume familiarity with the $\lambda \sigma$- ($\cdot, \circ, []$ and $\uparrow$ operators) and $\lambda s_c$-calculi ($\varphi$ and $\sigma$ operators and skeleton notation $\psi$), their typed versions and their normal form (nf, lnf and $\eta$-nf) characterizations as in [2].

Let $\mathcal{V}$ be a (countable) set of variables (disjoint from $\mathcal{X}$) denoted by $x, y, \ldots$. Terms $\Lambda(\mathcal{V})$, of the $\lambda$-calculus with names are inductively defined by $a ::= x | (a \ a) | \lambda_x.a$. As it is well-known, first order substitution or grafting leads to problems in the $\lambda$-calculus. For example, applying the first order substitution $\{u/x\}$ to $\lambda_x.(u \ x)$ results in $\lambda_x.(x \ x)$ which is wrong. Therefore, the $\lambda$-calculus with names uses variable renaming via $\alpha$-conversion so that $(\lambda_x.(u \ x))\{u/x\}$, by renaming $x$ (say as $y$), results in the correct term $\lambda_y.(x \ y)$. In $\Lambda(\mathcal{V})$, $\beta$- and $\eta$-reduction rules are defined respectively by $(\lambda_x.a \ b) \rightarrow a\{x/b\}$ and $\lambda_x.(a \ x) \rightarrow a$, if $x \notin \mathcal{F}\text{var}(a)$, where $\mathcal{F}\text{var}(a)$ denotes the set of free variables of $a$. We use $=_{\beta_\eta}$ to denote the congruence generated by $\beta$- and $\eta$-reduction.

Bound variables in $\Lambda(\mathcal{V})$ are untouched by unification substitutions. Unification variables in the $\lambda$-calculus are free variables. Thus free variables occurring at terms of a unification problem can be partitioned into true unification variables and constants, that cannot be bound by the unifiers.
To differentiate between unification and constant variables, one could consider unification variables as **meta-variables** in $\mathcal{X}$. Thus, $\lambda$-calculus should be defined as the term algebra, $\Lambda(\mathcal{V},\mathcal{X})$, over the set of operators $\{\lambda_x. \mid x \in \mathcal{V}\} \cup \{._.\} \cup \mathcal{V}$ and the set of variables $\mathcal{X}$. In this setting, a notion of substitution could be adapted for meta-variables preserving the semantics of both $\beta$- and $\eta$-reduction. But a better notation is the one of de Bruijn indices [11] where bound variables are related to their abstractions by their relative *height*.

E.g., $\lambda_x.(\lambda z.(x \ z) \ (x \ z))$ is translated into $\lambda.(\lambda.2 \ 1 \ (1 \ 4))$.

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

**Definition 1** Let $a \in \Lambda_{dB}(\mathcal{X})$, $i \in \mathbb{N}$. The $i$-**lift** $a^{+i}$ of $a$ is defined by:

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) $n^{+i} = \begin{cases} n + 1, & \text{if } n > i \\ n, & \text{if } n \leq i \end{cases}$ 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 2** The application $\{n/b\}a$ of the substitution with $b$ of $n \in \mathbb{N} \setminus \{0\}$ on a term $a$ in $\Lambda_{dB}(\mathcal{X})$, 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) $\{n/b\}m = \begin{cases} m - 1, & \text{if } m > n \\ b, & \text{if } m = n \\ m, & \text{if } m < n \end{cases}$ when $m \in \mathbb{N}$.

**Definition 3** Let $\theta = \{X_1/a_1, \ldots, X_n/a_n\}$ be a valuation from $\mathcal{X}$ to $\Lambda_{dB}(\mathcal{X})$ and use $\theta^+$ to denote the valuation (and substitution) $\{X_1/a_1^+, \ldots, X_n/a_n^+\}$.

The corresponding substitution, also denoted by $\theta$, is defined inductively as:

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

In $\Lambda_{dB}(\mathcal{X})$, the left side of the $\eta$-reduction rule is written as $\lambda.(a' \ 1)$, where $a'$ stands for the corresponding translation of $a$ into $\Lambda_{dB}(\mathcal{X})$. In $\Lambda_{dB}(\mathcal{X})$, “$x \not\in \mathcal{Fvar}(a)$” means that in $a'$ there are no occurrences of index 1 at height zero, nor of the index 2 at height one etc., and hence there is a $b$ such that $b^+ = a$.

Thus $\beta$ and $\eta$ are defined as $(\lambda \ a \ b) \rightarrow \{1/b\}a$ resp. as $\lambda.(a' \ 1) \rightarrow b$ if $\exists b \ b^+ = a$.

## 2 Unification in the $\lambda_s$-calculus

In this section we review the $\lambda_s$-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 $\lambda s_e$-calculus, which can be applied to HOU in the $\lambda$-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 $\mathcal{F}$, $\mathcal{T}$ 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: $D = \bigvee_{j \in J} \exists w_j \bigwedge_{i \in I_j} s_i =^\mathcal{A} t_i$. When $|J| = 1$, the unification problem is called a unification system. Variables in the set $\bar{w}$ of a unification system $\exists \bar{w} \bigwedge_{i \in I} s_i =^\mathcal{A} t_i$ are bound while all others are free. $\mathcal{T}$ and $\mathcal{F}$ stand for the empty conjunction and disjunction, respectively. The empty disjunction corresponds to an unsatisfiable problem.

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

**Definition 4** A $\lambda s_e$-unification problem $P$ is a unification problem in the algebra $\mathcal{T}_{\lambda s_e}(\mathcal{X})$ modulo the equational theory of $\lambda s_e$. An equation of such a problem is denoted by $a =^\lambda s_e b$, where $a$ and $b$ are $\lambda s_e$-terms of the same sort. An equation is called trivial when it is of the form $a =^\lambda s_e a$.

[2] 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 [6].

**Definition 5** ([2]) Table 1 defines the $\lambda s_e$-unification rules for typed $\lambda s_e$-unification problems.

Since $\lambda s_e$ is CR and WN [9], the search can be restricted\(^1\) to $\eta$-long normal solutions that are graftings binding functional variables into $\eta$-long normal terms of the form $\lambda a$ and atomic variables into $\eta$-long normal terms of the form $(k b_1 \ldots b_p)$ or $\varphi b$ or $\varphi^a k a$, where in the first case $k$ can be omitted and $p$ is zero. The $\text{Eta}$ 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

---

\(^1\)Use of $\lambda s_e$-normal forms in $\text{Exp-App}$ 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 $\text{Exp-App}$, $\lambda s_e$-unification problems are normalized.
Table 1: $\lambda s_e$-unification rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Dec-(\lambda))</td>
<td>( P \land \lambda A. a =<em>\lambda^1 b \rightarrow P \land a =</em>\lambda^1 b )</td>
</tr>
<tr>
<td>(Dec-App)</td>
<td>( P \land (n a_1 \ldots a_p) =<em>\lambda^1 (n b_1 \ldots b_q) \rightarrow P \land \bigwedge</em>{i=1..p} a_i =_\lambda^1 b_i )</td>
</tr>
<tr>
<td>(App-Fail)</td>
<td>( P \land (n a_1 \ldots a_p) =_\lambda^1 (m b_1 \ldots b_q) \rightarrow \bot ) if ( n \neq m )</td>
</tr>
<tr>
<td>(Dec-(\varphi))</td>
<td>if ( \psi_{i_1}^{p_1} \ldots \psi_{i_i}^{p_i}(X, a_1, \ldots, a_p) =<em>\lambda^1 (m b_1 \ldots b_q) ) then ( \psi</em>{i_1}^{p_1} \ldots \psi_{i_i}^{p_i}(X, a_1, \ldots, a_p) =<em>\lambda^1 (m b_1 \ldots b_q) \land \bigwedge</em>{r \in R_0} \exists H_1, \ldots, H_k, X =_\lambda^1 (x H_1 \ldots H_k) )</td>
</tr>
<tr>
<td>(Exp-(\lambda))</td>
<td>( P \rightarrow \exists (Y : A \Gamma \vdash B), P \land X =_\lambda^1 \lambda A. Y ) if ( Y \not\in \text{var}(P) ) or ( \lambda Y \not\in \text{var}(P) ) and ( X ) is an unsolvable variable</td>
</tr>
<tr>
<td>(Exp-App)</td>
<td>( P \land \psi_{i_1}^{p_1} \ldots \psi_{i_i}^{p_i}(X, a_1, \ldots, a_p) =<em>\lambda^1 (m b_1 \ldots b_q) \rightarrow P \land \psi</em>{i_1}^{p_1} \ldots \psi_{i_i}^{p_i}(X, a_1, \ldots, a_p) =<em>\lambda^1 (m b_1 \ldots b_q) \land \bigwedge</em>{r \in R_0} \exists H_1, \ldots, H_k, X =_\lambda^1 (x H_1 \ldots H_k) )</td>
</tr>
<tr>
<td>(Replace)</td>
<td>( P \land X =<em>\lambda^1 a \rightarrow { X/a } P \land X =</em>\lambda^1 a )</td>
</tr>
<tr>
<td>(Normalize)</td>
<td>( P \land a =<em>\lambda^1 b \rightarrow P \land a' =</em>\lambda^1 b' ) if ( a ) or ( b ) is not in ( \text{lnf} ) and ( a' ) is the ( \text{lnf} ) of ( a ) if ( a ) is not a solved variable and ( a ) otherwise. ( b' ) is defined from ( b ) identically</td>
</tr>
</tbody>
</table>

\( \lambda s_e \) and \( \text{Dec-\(\lambda\)} \) use CR and WN of \( \lambda s_e \) to normalize equations of the form \( \lambda a =_\lambda^1 \lambda b \) into \( a' =_\lambda^1 \lambda b' \) and \( \text{Replace} \) propagates the grafting \{\( X/a \)\} corresponding to equations \( X =_\lambda^1 a \). \( \text{Exp-\(\lambda\)} \) generates the grafting \{\( X/\lambda Y \)\} for a variable \( X \) of type \( A \rightarrow B \), where \( Y \) is a new variable of type \( B \). \( \text{Dec-App} \) and \( \text{App-Fail} \) transform equations of the form \( (n a_1 \ldots a_p) =_\lambda^1 (m b_1 \ldots b_q) \) into the empty disjunction when \( n \neq m \), as they have no solution, or into the conjunction \( \bigwedge_{i=1..p} a_i =_\lambda^1 b_i \), when \( n = m \). Analogously, \( \text{Dec-\(\varphi\)} \) decomposes equations with leading operator \( \varphi \). It can be easily checked, using the arithmetic properties of \( \lambda s_e \) to build counterexamples, that the addition of the corresponding \( \text{Dec-\(\sigma\)} \), \( \sigma-Fail \) and \( \text{\(\varphi\)-Fail} \) is wrong. In \( \lambda s_e \), the rule \( \text{Exp-App} \) advances towards solutions to equations of the form \( X[a_1 \ldots a_p, \uparrow^n] =_\lambda^1 (m b_1 \ldots b_q) \) where \( X \) is an unsolved variable of an atomic type. This process is similar for \( \lambda s_e \)-unification problems.

**Example 1** Take the problem \( (\lambda.(\lambda.(X \ 2) \ 1) \ Y) =_\lambda^1 (\lambda.(Z \ 1) \ U) \) where \( X \),
\[ \lambda s_c\text{-STYLE OF UNIFICATION FOR SIMPLY-TYPED HOU} \]

\[ Y, Z \text{ and } U \text{ are meta-variables. By Normalize we get } ((X \sigma^2 Y) \sigma^1 (\varphi_0^1 Y) =_\lambda \lambda (Z \sigma^1 U \varphi_0^1 U) \text{ which after Dec-App, } \varphi \text{ and Replace gives } (X \sigma^2 Y) \sigma^1 (\varphi_0^1 Y) =_\lambda \lambda Z \sigma^1 Y \land Y \sigma^2 U =_\lambda \lambda U. \text{ Since } X \text{ and } Z \text{ are variables of functional type, Exp-App and Replace give } ((\lambda X') \sigma^2 Y) \sigma^1 (\varphi_0^1 Y) =_\lambda \lambda (\lambda Z') \sigma^2 Y \land Y =_\lambda \lambda U =_\lambda \lambda X \land X =_\lambda \lambda \lambda. \text{ Finally, Normalize and Dec-\lambda give } (X' \sigma^3 Y) \sigma^2 (\varphi_0^1 Y) =_\lambda \lambda Z' \sigma^2 Y \land Y =_\lambda \lambda U =_\lambda \lambda X \land X =_\lambda \lambda \lambda. \text{ Solutions are built as } \{Y/X_1, U/X_1\} \text{ union solutions for } X \text{ and } Z \text{ obtained by the first equation. The first equations, called Flex-Flex, are related to the pre-unifiers of [8]. E.g., here we can take } \{X/Y_1, U/Y_1\} \cup \{X/\lambda \cdot n + 1, Z/\lambda \cdot n\}, \text{ where } n > 2. \]

**Example 2** from \( \lambda. (\lambda (Y 1) \lambda (X 1)) =_\lambda \lambda (\lambda. V \lambda. W) \) one obtains:

\[ (Y' \lambda (X 1).id) \lambda (X 1)) =_\lambda \lambda V[\lambda W.id] \text{ and } (Y' \sigma^1 \lambda (X 1) \lambda (\varphi^1 1)) =_\lambda \lambda V^1 \lambda W. \text{ By Exp-App with } V =_\lambda \lambda (V_1 \ V_2) \text{ and } V =_\lambda \lambda (V_1 \ V_2), \text{ we get } \lambda (X 1) =_\lambda \lambda V_2[\lambda (X 1).id] \text{ and } \lambda (\varphi 1 X 1) =_\lambda \lambda V_2^1 \lambda (\lambda (X 1).id). \text{ For solutions take } V_2 =_\lambda \lambda 1 \text{ or } V_2 =_\lambda \lambda 1. \]

**Definition 6** A unification system is in \( \lambda s_c\text{-solved form} \) if its meta-variables are atomic and it is a conjunction of non trivial equations of the forms:

(Solved) \( X =_\lambda \lambda a \), where \( X \) does not occur anywhere else in \( P \) and \( a \) is in long normal form. Both \( X \) and \( X =_\lambda \lambda a \) are said to be solved in \( P \).

(Flex-Flex) non solved equations between long normal terms whose root operator is \( \sigma \) or \( \varphi \) which we represent as equations between their skeleton:

\[ \psi_k^p \ldots \psi_k^p (X, a_1, \ldots , a_p) =_\lambda \lambda \psi_k^q \ldots \psi_k^q (Y, b_1, \ldots , b_q) \text{ with } X, Y \text{ atomic.} \]

**Lemma 1** ([2])

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

**Definition 7** Let \( P \) and \( P' \) be \( \lambda s_c\text{-unification problems, let “rule” denote the name of a } \lambda s_c\text{-unification rule and “}\rightleftharpoons “ \text{ its corresponding deduction relation. By correctness and completeness of rule we understand } P \rightarrow \text{rule } P' \text{ implies } \mathcal{U}_{\lambda s_c}(P') \subseteq \mathcal{U}_{\lambda s_c}(P) \text{ and } P \rightarrow \text{rule } P' \text{ implies } \mathcal{U}_{\lambda s_c}(P) \subseteq \mathcal{U}_{\lambda s_c}(P'), \text{ respectively.} \]

**Theorem 1** (Correctness and completeness [2]) The \( \lambda s_c\text{-unification rules are correct and complete.} \)

An analogous unification strategy to that of [6] for \( \lambda \sigma \) applies as well in this setting. Correctness and completeness proofs for these strategies essentially do not differ because they are based on an order of the application of the unification rules which is independent of the calculi [1].
3 HOU in the pure $\lambda$-calculus

[2], dealt only with the $\lambda s_c$-unification method (half of the box on the right hand side of Figure 1). For applying this method to HOU in $\lambda$-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 $\lambda s_c$-unification for HOU in pure $\lambda$-calculus. This is what we do in this section.

Observe firstly that unifying two terms $a$ and $b$ in the $\lambda$-calculus consists in finding a substitution $\theta$ such that $\theta(a) =_B \theta(b)$. Thus using the notation of substitution in Definitions 2 and 3, a unifier in the $\lambda$-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 $\lambda$-terms $a, b \in \Lambda_{AB}(X)$, 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 $\lambda s_c$.

**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 $\theta$ to the $\Lambda_{AB}(X)$-term $\lambda. (X \ 2)$ gives $\theta(\lambda. (X \ 2)) = \lambda. (\theta^+(X) \ 2)$. Then in the $\lambda s_c$-calculus we are searching for a grafting $\theta'$ such that $\theta'(\lambda. (\varphi^2_0(X) \ 2)) =^?_{\lambda s_c} \lambda. 2$. 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 $\hat{b}[\uparrow] = a$ and $\varphi^2_0 b = a$). Then we should search for unifiers for the problem $\lambda. (\varphi^2_0(X) \ 2) =^?_{\lambda s_c} \lambda. 2$.

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

In this way substitution solutions $\{X/\lambda. 1\}$ and $\{X/\lambda. 2\}$ are found.

Finally, note that $\text{Defs} \ 2$, $3$ and $\beta$-reduction give $\{X/\lambda. 1\}(\lambda. (X \ 2)) = \lambda. (\lambda. (X \ 2) \ 2 = \lambda. (\lambda. 1 \ 2) = \beta \lambda. 2$ and $\{X/\lambda. 2\}(\lambda. (X \ 2)) = \lambda. (\lambda. 2 \ 2) = \lambda. (\lambda. 3 \ 2) = \beta \lambda. (1/2)(3) = \lambda. 2$.

In general, before the unification process, a $\lambda$-term $a$ should be translated into a $\lambda s_c$-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 abstractions 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 $n$ into $1[|n-1|$ and all occurrences of meta-variables $X$ into $X[|n|$, with $k$ as above. Notice that the two pre-cooking translations can be implemented non-terminally in an efficient way.

**Example 4** Consider the HOU problem $F(f(a)) =^? f(F(a))$. In $\lambda_{AB}(X)$ it can be seen as $(X (2 1)) =^?_{\beta_1} (2 (X 1))$, where both $X$ and $2$ are of type $A \rightarrow A$ and 1 is of type A. Since there are no abstractions at the terms of the equational problem, the equation remains unchanged: $(X (2 1)) =^?_{\lambda_{se}} (2 (X 1))$.

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

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

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

Third solved system: By continuing the application of $\text{Dec-App}$, $\text{Exp-App}$, $\text{Replace}$ and $\text{Normalize}$ we obtain grafting solutions corresponding to $F = f f f$, $F = f f f f$, etc. to the equational system $((2 H_2^{\sigma 1} (2 1)) =^?_{\lambda_{se}} (2 (2 H_2^{\sigma 1} \land X =^?_{\lambda_{se}} \lambda (3 (3 H_2)))$ we obtain the third solved system giving the grafting solution $\{X/\lambda (3 (3 1))\}$ corresponding to the solution $F = f f$. 
The unification process continues infinitely producing solved systems corresponding to the grafting solutions \{X/\lambda.\underline{3} \ (\underline{3} \ (\underline{3} \ 1))\} (i.e. \(F = ffff\)), \{X/\lambda. \underline{3} \ (\underline{3} \ (\underline{3} \ 1))\} (i.e. \(F = ffff\)), etc.

Now we can define our pre-cooking translation.

**Definition 8 (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 2). We give every variable \(X\) of type \(A\) in the same type and context \(\Gamma\) in the \(\lambda_{se}\)-calculus. The pre-cooking of \(a\) from \(\Lambda_{dB}(\mathcal{X})\) to the \(\lambda_{se}\)-calculus is defined by:

1) \(PC(\lambda_{B} \cdot a, n) = \lambda_{B}.PC(a, n + 1)\)
2) \(PC((a \ b), n) = (PC(a, n) \ PC(b, n))\)
3) \(PC(k, n) = k\)
4) \(PC(X, n) = \begin{cases} X, & \text{if } n = 0 \\ \phi_{0}^{n+1} X, & \text{otherwise} \end{cases}\)

**Lemma 2 (Type preservation)** If \(\Gamma \vdash_{\Lambda_{dB}(\mathcal{X})} a : T\), then \(\Gamma \vdash_{\lambda_{se}} a_{pc} : T\).

**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 \(\Gamma\), then \(A_1 \ldots A_n, \Gamma \vdash_{\lambda_{se}} PC(a, n) : T\). This is done by induction on the structure of terms, for all \(n\).

Cases \(a = k\) and \(a = (a_1 \ a_2)\) are simple. Case \(a = \lambda_{B} \cdot b\), then \(T = B \rightarrow C\) and \(B, A_1 \ldots A_n, \Gamma \vdash_{\Lambda_{dB}(\mathcal{X})} b : C\). Thus \(B, A_1 \ldots A_n, \Gamma \vdash_{\lambda_{se}} PC(b, n + 1) : C\) and \(A_1 \ldots A_n, \Gamma \vdash_{\lambda_{se}} PC(\lambda_{B} \cdot b, n) = \lambda_{B}.PC(b, n + 1) : B \rightarrow C\). Case \(a = X\) then as \(\Gamma \vdash_{\Lambda_{dB}(\mathcal{X})} X : T\), \(\Gamma \vdash_{\lambda_{se}} X : T\) and \(A_1 \ldots A_n, \Gamma \vdash_{\lambda_{se}} \phi_{0}^{n+1}(X) : T\).

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:

\(a\{X_1/b_1, \ldots, X_p/b_p\}\) is proved by induction on the structure of terms for all \(i\) (see Appendix A). The case \(i = 0\) corresponds to the proposition. In contrast to the related proof in [6] where substitution objects \([1 \ldots k. \ u^{i+k}]\) are necessary for proving the critical case of \(a = X\) our proof uses pure term objects by selecting the appropriate super and subscripts for \(\phi\) (i.e., \(\phi_{k}^{i+1}\)).

The following proposition, also proved by induction on the structure of terms (see Appendix A), presents necessary facts for relating the existence of solutions for unification problems in the pure \(\lambda\)-calculus and in the \(\lambda_{se}\)-calculus.

**Proposition 2** Let \(a\) and \(b\) be terms in \(\Lambda_{dB}(\mathcal{X})\). Then:

1. \(a \rightarrow_{\beta} b\) implies \(a_{pc} \rightarrow_{\lambda_{se}} b_{pc}\)
2. If \(a\) is \(\beta\eta\)-nf then \(a_{pc}\) is \(\lambda_{se}\)-nf
3. \(a \rightarrow_{\eta} b\) implies \(a_{pc} \rightarrow_{\eta_{a}} b_{pc}\)
4. \(a =_{\beta\eta} b\) if and only if \(a_{pc} =_{\lambda_{se}} b_{pc}\)
Again, our proof differs from that of [6] in that we avoid complicated substitution objects because we profit from the semantics of $\varphi$ in the $\lambda s_c$-calculus. Finally we relate solutions and their existence in the $\lambda$-calculus to those of $\lambda s_c$.

**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 $a\{X_1/N_1, \ldots, X_p/N_p\} = b_\beta b\{X_1/N_1, \ldots, X_p/N_p\}$ if and only if there exist $\lambda s_c$-terms $M_1, \ldots, M_p$ where $a_{pc}\{X_1/M_1, \ldots, X_p/M_p\} = b_{pc}\{X_1/M_1, \ldots, X_p/M_p\}$.

**Proof.** If $\{X_i/N_i\}_{i=1..p}$ is a solution of the unification problem $a = b_\beta b$ then $a\{X_i/N_i\} = b_{pc}\{X_i/N_i\}$. By Proposition 2A, $(a\{X_i/N_i\})_{pc} = (b\{X_i/N_i\})_{pc}$. By Proposition 1, $a_{pc}\{X_i/N_{ipc}\}_g = b_{pc}\{X_i/N_{ipc}\}_g$. If $a_{pc}\{X_i/M_i\}_g = b_{pc}\{X_i/M_i\}_g$ we select terms $N_i$, $i = 1, \ldots, p$, in the pre-cooking range such that $N_i = M_i$ and take $M_i$ in $\Lambda_{dB}(\mathcal{X})$ such that $M_{ipc} = N_i$. Hence, $a_{pc}\{X_i/M_{ipc}\}_g = b_{pc}\{X_i/M_{ipc}\}_g$. By Proposition 1, $(a\{X_i/M_i\})_{pc} = (b\{X_i/M_i\})_{pc}$. Hence by Proposition 2 $a\{X_i/M_i\} = b_{\beta} b\{X_i/M_i\}$.

In addition to pre-cooking, we need a Back translation for giving descriptions of the original pre-cooked problems. That means, that for any unification problem $P$, derived by applying the $\lambda s_c$-unification rules to the pre-cooking $a_{pc} = b_{pc}$, 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 $\lambda$-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 $a = b_{\beta}$. In this way the set of solutions is given as solved forms. By the correspondence between solutions (Proposition 3), we have that if $a = b_{\beta}$ has a solution then so does its pre-cooking $a_{pc} = b_{pc}$. Here we do not present the proof of the converse which can be done similarly to that of the $\lambda \sigma$-HOU approach in [6].

The $\lambda s_c$-unification rules are extended with the following rules:

(Anti-Exp-$\lambda$) $P \rightarrow \exists Y (P \land X = \lambda s_b (\varphi_0 Y \ 1))$ if $(X : A. \Gamma_X \vdash A_X) \in \text{var}(P)$, $\text{mbox}$ \hspace{1cm} \text{where } (Y : \Gamma_Y \vdash A \rightarrow A_X) \notin \text{var}(P)$

(Anti-Dec-$\lambda$) $P \land a = \lambda s_b \rightarrow P \land A.a = \lambda s_b \lambda A.b$ if $a = b$ is well-typed in a context $A. \Gamma$

**Proposition 4 (Correctness and completeness of the Anti-rules)** The rules of $\lambda s_c$-unification, Anti-Exp-$\lambda$ and Anti-Dec-$\lambda$ 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 \( P \land a \equiv \lambda_{\delta_n} \ b \), where the former equation is well-typed in the context \( A, \Gamma \), are also solutions of \( P \land \lambda_{A_1} a \equiv \lambda_{\delta_n} \lambda_{A_2} b \) (which is now the last well-typed equation in the context \( \Gamma \)).

For \( \text{Anti-Exp-}\lambda \), suppose that \( \theta \) is a grafting solution of problem \( P \) and select \( \theta' = \theta \cup \{ Y/Y_1 \lambda_{A} \theta X \} \).

Then \( \theta' (\varphi_0^2 Y \ 1) = (\varphi_0^2 \lambda_{A} \theta X \ 1) = \lambda_{\delta_n} (\lambda_{A_1} \varphi_1^2 \theta X \ 1) = \lambda_{\delta_n} (\varphi_1^2 \theta X) \sigma^1 1 \).

We analyse the former term. On one hand, \( \varphi_1^2 \) increases by one all free de Bruijn indices occurring at \( \theta X \) except those corresponding to the variable of the free de Bruijn index 1. On the other hand, \( "\sigma^1 1" \) decrements by one all free occurrences of de Bruijn indices in \( \varphi_1^2 \theta X \) except those untouched by \( \varphi_1^2 \). Then \( (\varphi_1^2 \theta X) \sigma^1 1 = \lambda_{\delta_n} \theta X \). \( \square \)

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

**Proposition 5** Let \( a \equiv \lambda_{\delta_n} b \) be a HOU problem well-typed in a context \( \Gamma \) and \( P \) derived by the \( \lambda_{S_e} \)-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 context \( \Delta \), then \( \Delta \) is an extension of \( \Gamma \);  
2) for every variable \( Y \), its context \( \Gamma_Y \) is an extension of \( \Gamma \);  
3) for every subterm \( \psi_{i_p} \ldots \psi_{i_1} (X, a_1, \ldots, a_p) \) in \( P \) we have \( p \leq |\Gamma_Y| - |\Gamma| + 1 \).

**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 \( \lambda_{S_e} \)-unification rules. Suppose that \( P \) satisfies these invariants. Then since the rules \( \text{Anti-Exp-}\lambda \) and \( \text{Anti-Dec-}\lambda \) are applied only on variables and equations, whose contexts are strict extensions of \( \Gamma \), the first and second invariants are maintained. The third invariant is maintained too, since subterms of the form \( \psi_{i_p} \ldots \psi_{i_1} (X, a_1, \ldots, a_p) \) are either already of this form in \( P \) or obtained by the two new rules as \( \varphi_0^2 Y \) in whose case \( p = 1 \leq |\Gamma_Y| - |\Gamma| + 1 \) holds, since \( \Gamma_Y \) is an extension of \( \Gamma \). \( \square \)

**Proposition 6** (Building Back Pre-cooking images) Let \( P \) be a problem derived from the application of the \( \lambda_{S_e} \)-unification rules to the pre-cooking of a given HOU problem \( a \equiv \lambda_{\delta_n} b \). The system resulting from normalization of \( P \) by applying the Back strategy is the pre-cooking of a problem in the \( \lambda \)-calculus.

This is proved by simple examination of the effects of the rules \( \text{Anti-Dec-}\lambda \) and \( \text{Anti-Exp-}\lambda \) over \( P \) (see Appendix A).

**Corollary 1** (Soundness of the construction of solutions) Let \( a \equiv \lambda_{\delta_n} b \) a HOU problem such that its pre-cooking, normalised with the \( \lambda_{S_e} \)-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 $\lambda$-solved form (in the sense of [12]) and the solutions of $R$ are solutions of the original HOU problem.

**Proof.** The pre-cooking of a substitution solution in the $\lambda$-calculus, $\theta$, of $R$, is a solution of $R_{pc}$ and then of $Q$ (Proposition 3). Additionally, $\theta_{pc}$ is a solution of $P$ (Proposition 4) and then of $a_{pc} = \lambda_{s_c} b_{pc}$ (Theorem 1). Then $\theta$ is solution of $a =^?_{\beta\eta} b$ (the converse of Proposition 3).

**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 the context $\Gamma$. Any solution of the initial problem can be obtained as the one of a system in $\lambda$-solved form resulting from the application of the $\lambda_{s_c}$-unification rules, followed by the Back strategy and the inverse of the pre-cooking translation.

**Proof.** We use the same notation as in the previous Corollary. $\theta$ is a substitution solution of $R$, if and only if $\theta_{pc}$ is a solution of $R_{pc}$, if and only if $\theta_{pc}$ is a solution of $Q$, if and only if $\theta_{pc}$ is a solution of $P$, if and only if $\theta_{pc}$ is a solution of $a_{pc} = \lambda_{s_c} b_{pc}$, if and only if $\theta$ is a solution of $a =^?_{\beta\eta} b$. □

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 $\lambda_{s_c}$-HOU one.

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 $\lambda_{s_c}$-calculus, the $\lambda_{s_c}$-typing rules and, of course, the $\lambda_{s_c}$-unification rules. Table 2 gives the decorated version of the typing rules for the $\lambda_{s_c}$-calculus.

The typing rules $Var$ and $Varn$ can be reduced to a sole decorated rule of the form $A_1^{\Gamma_1} \cdots A_n^{\Gamma_n}$ 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 $\Gamma_X$ and a unique type $A_X$ such that the rule holds. This is done in order to obtain compatibility
Table 2: Undecorated and decorated typing rules for the $\lambda s_e$-calculus

<table>
<thead>
<tr>
<th>(Var)</th>
<th>$A, \Gamma \vdash 1 : A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Var)</td>
<td>$\Gamma \vdash n : B$</td>
</tr>
<tr>
<td>(Lambda)</td>
<td>$A, \Gamma \vdash \lambda a : A \rightarrow B$</td>
</tr>
<tr>
<td>(App)</td>
<td>$\Gamma \vdash b : A \rightarrow B$, $\Gamma \vdash a : A$</td>
</tr>
<tr>
<td>(Sigma)</td>
<td>$\Gamma \vdash b : B$, $\Gamma \vdash a : A$</td>
</tr>
<tr>
<td>(Phi)</td>
<td>$\Gamma \vdash \varphi_k a : A$</td>
</tr>
<tr>
<td>(Meta)</td>
<td>$\Gamma \vdash X : A_X$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$1^A, \Gamma$</th>
<th>$n_B^\Gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(n + 1)^A, \Gamma$</td>
<td>$b^B_{A\rightarrow B}$</td>
</tr>
<tr>
<td>$(\lambda a, b : A \rightarrow B)^A_{\rightarrow B}$</td>
<td>$(\lambda a, b : A \rightarrow B)^B_{A\rightarrow B}$</td>
</tr>
<tr>
<td>$(\lambda a, b : A \rightarrow B)^{A_{&lt; I}, B, \Gamma_{&gt; I}}_{A \rightarrow B}$</td>
<td>$(\lambda a, b : A \rightarrow B)^{A_{&lt; I}, B, \Gamma_{&gt; I}}_{A \rightarrow B}$</td>
</tr>
<tr>
<td>$(\lambda a, b : A \rightarrow B)^{A_{&gt; I}}_{A \rightarrow B}$</td>
<td>$(\lambda a, b : A \rightarrow B)^{A_{&gt; I}}_{A \rightarrow B}$</td>
</tr>
<tr>
<td>$(\varphi_k a : A)^{A_{&lt; I}, B, \Gamma_{&gt; I}}_{A \rightarrow B}$</td>
<td>$(\varphi_k a : A)^{A_{&lt; I}, B, \Gamma_{&gt; I}}_{A \rightarrow B}$</td>
</tr>
<tr>
<td>$(\varphi_k a : A)^{A_{&lt; I}, B, \Gamma_{&gt; I}}_{A \rightarrow B}$</td>
<td>$(\varphi_k a : A)^{A_{&lt; I}, B, \Gamma_{&gt; I}}_{A \rightarrow B}$</td>
</tr>
<tr>
<td>$X^A_X$</td>
<td>$X^A_X$</td>
</tr>
</tbody>
</table>

between typing and grafting. We suppose that for each pair $(\Gamma, A)$ 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$^2$. Examples of these rules are: (Shift) $\uparrow^{A_1, A_2} \Gamma$, (Comp)$s_\sigma^\Gamma$, $t^\sigma_\delta \vdash (s_\sigma^\Gamma \circ t^\sigma_\delta)^A_{A_\delta}$; (Clos)$a^\Gamma_{A_\delta}$, $s^\Gamma_{A_\delta} \vdash (a^\Gamma_{A_\delta})^{A_\delta}_{A}$. This kind of explicit decoration was done for the $\lambda \sigma$-HOU approach in [5], but maintaining this discipline in the $\lambda s_e$-calculus is more economical in both space and time. Let us compare the previous linear decoration of a de Bruijn index, $n$, in $\lambda s_e$ and its corresponding $\lambda \sigma$-term $1[\uparrow^{n-1}]$:

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

$$(\text{comp}) \uparrow^{A_{n-1}, A_n}_{\Gamma_{A_n}} \times (\text{shift}) \uparrow^{A_{n-2}, A_n}_{\Gamma_{A_n}}$$

$$(\text{comp}) \uparrow^{A_{n-1}, A_n}_{\Gamma_{A_n}} \circ \uparrow^{A_{n-2}, A_n}_{\Gamma_{A_n}} \circ \cdots \circ \uparrow^{A_1, A_n}_{\Gamma_{A_n}} \times (\text{shift}) \uparrow^{A_{n-1}, A_n}_{\Gamma_{A_n}}$$

$$(\text{comp}) \uparrow^{A_{n-1}, A_n}_{\Gamma_{A_n}} \circ \uparrow^{A_{n-2}, A_n}_{\Gamma_{A_n}} \circ \cdots \circ \uparrow^{A_1, A_n}_{\Gamma_{A_n}} \times (\text{shift}) \uparrow^{A_{n-2}, A_n}_{\Gamma_{A_n}}$$

$$(\text{comp}) \uparrow^{A_{n-1}, A_n}_{\Gamma_{A_n}} \circ \uparrow^{A_{n-2}, A_n}_{\Gamma_{A_n}} \circ \cdots \circ \uparrow^{A_1, A_n}_{\Gamma_{A_n}} \times (\text{shift}) \uparrow^{A_{n-1}, A_n}_{\Gamma_{A_n}}$$

This 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 de-

---

$^2$This is denoted in the undecorated setting as $s \triangleright \Gamma$. 
development of the ELAN implementation of the method is related to the sole de Bruijn index 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 $\lambda s_c$, 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 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 $\lambda s_c$-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 $\lambda s_c$ and $\lambda s_c$-unification rules are supposed decorated and, of course, they preserve types and contexts. Initial decoration can be done using the algorithm in Table 3. This algorithm is based on a straightforward propagation of the decoration of subterms composing a $\lambda s_c$-term according to the decorated $\lambda s_c$-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 previous algorithm runs in time linear on the size of the initial $\lambda s_c$-term and on the magnitude of its de Bruijn indices. For this algorithm it is necessary to know the main context, 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 context expressions generated from the typing rules of the $\lambda s_c$-calculus.

Our previous remarks point out the advantage of $\lambda s_c$ 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 $1[\uparrow^{n-1}]$ of Example 5. Its explicit decoration is, of course, quadratic. Consequently we can state the following.

**Lemma 3 (Linear against quadratic decorations)** Pre-cooked $\lambda$-terms in the $\lambda s_c$-calculus have linear decorations on the size of the $\lambda$-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 $\lambda s_c$ calculi.  

Note that the size of decorated $\lambda$-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 λσ-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 6** \((λ_{A,*}(λ_{A,*}X_A^{A,A,A,G})_{A→A}^{A,A,G} A^{A,G} 1_A^{A,G})_{A→A}^{A,G} 1^{A,G} A^{A,G} 1^{A,G})^{A,G}\) is the
decorated version of \((λ_{A,*}(λ_{A,*}X)_{1} 1). Compare the corresponding decorated terms
in the λsc- and λσ-calculi after two applications of Beta.

In the λsc: \(\rightarrow\)Beta \(((λ_{A,*}(X_A^{A,A,A,G})_{A→A}^{A,A,G} 1_A^{A,G})_{A→A}^{A,G} 1^{A,G})^{A,G}\)
\(\rightarrow\)Beta \(((X_A^{A,A,A,G} 1_A^{A,G})_{A→A}^{A,A,G} 1^{A,G})_{A→A}^{A,G} 1^{A,G})^{A,G}\).

In λσ: \(\rightarrow\)Beta \(((λ_{A,*}(X_A^{A,A,A,G})_{A→A}^{A,A,G} 1_A^{A,G})_{A→A}^{A,G} 1^{A,G})_{A→A}^{A,G} 1^{A,G})_{A→A}^{A,G}\).

\(\rightarrow\)Beta \(((X_A^{A,A,A,G})_{A→A}^{A,A,G} 1_A^{A,G})_{A→A}^{A,A,G} 1^{A,G})^{A,G}\).

**Table 3:** Type checking / decorating algorithm for the λsc-calculus

<table>
<thead>
<tr>
<th>INPUT: a a λsc-term and Γ a context.</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTATION: ⊥ denotes unknown types and contexts.</td>
<td></td>
</tr>
<tr>
<td>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 ⊥. Afterwards, apply nondeterministically to the decorated term the following rules until an irreducible term is obtained.</td>
<td></td>
</tr>
</tbody>
</table>

\([(\text{Varn})]\)  \(n_{\perp}^{A_1,...,A_n,Γ} \rightarrow n_{A_1,...,A_n,Γ}\)

\([(λ−)]\)  \((λ_{A,*}a^1_{\perp})^{Γ} \rightarrow (λ_{A,*}a^1_{A,Γ})^{Γ}\)

\([(λ→)]\)  \((λ_{A,*}a^1_{B,Γ})_{A→B}^{Γ} \rightarrow (λ_{A,*}a^1_{B,Γ})^{Γ}\)

\([(\text{app−}]\)\)  \(a^1_{\perp} b^1_{\perp}^{Γ} \rightarrow (a^1_{B,Γ} b^1_{Γ})^{Γ}\)

\([(\text{app→}]\)\)  \(a^1_{\perp} b^1_{Γ}^{Γ} \rightarrow (a^1_{B,Γ} b^1_{Γ})^{Γ}\)

\([(σ−)]\)  \((a^1_{B,Γ} σ b^1_{B,Γ})^{Γ} \rightarrow (a^1_{B,Γ} σ b^1_{Γ})^{Γ}\)

\([(σ→)]\)  \((a^1_{B,Γ} σ b^1_{Γ})^{Γ} \rightarrow (a^1_{B,Γ} σ b^1_{Γ})^{Γ}\)

\([(φ−)]\)  \((φ^1_{k} a^1_{\perp})_{Γ} \rightarrow (φ^1_{k} a^1_{\perp})_{Γ}\)

\([(φ→)]\)  \((φ^1_{k} a^1_{\perp})_{Γ} \rightarrow (φ^1_{k} a^1_{\perp})_{Γ}\)

\([(Meta)]\)  \(X^{Γ,X} \rightarrow X^{Γ,X}\)

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 the λσ-calculus 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 the \(\lambda\sigma\)-calculus (i.e., \((\lambda_A.a \ b) \rightarrow a[b.id]\)) which generates two new substitution subterms to be marked in a decorated term: \(id\) and \(b.id\), while for the Beta rule of the \(\lambda s_e\)-calculus, \((\lambda_A.a \ b) \rightarrow a\sigma^1 b\), the number of subterms is reduced by one. Critical is the case of the Abs rule of the \(\lambda\sigma\)-calculus, \((\lambda_A.a)[s] \rightarrow \lambda_A.a[1.(\sigma \uparrow)]\), 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 \(\lambda s_e\) are \(\sigma\)-app-transition, \(\varphi\)-app-transition, \(\sigma\sigma\)-transition and \(\varphi\sigma\)-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 \(\lambda s_e\)-calculus are supposed decorated. For example, the decorated Eta rule has the following form: \((\text{Eta}) \ (\lambda_A.(a^A \Gamma_B 1^A \Gamma_B)^{\Gamma}_A \rightarrow b^\Gamma_B \text{ if } a^A \Gamma_B = \sigma \ (\varphi_0^B)^{\Gamma}_A \Rightarrow \) \(\lambda_A.(a^A \Gamma_B)^{\Gamma}_A \rightarrow b^\Gamma_B \).

Except for this rule, application of the rules of the \(\lambda s_e\)-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 \(\text{Eta}\) rule can be implemented according to the correspondence between the two \(\text{Eta}\) rules and following the idea suggested for the \(\lambda\sigma\)-HOU approach in [5]. We can extend the language of the \(\lambda s_e\)-calculus with a dummy symbol \(\diamond\) and verify for occurrences of this symbol after \(s_e\)-normalizing the term \((a^A \rightarrow B\sigma^1 \varphi^\Gamma_A)_{A \rightarrow B}\). In the case that the previous term has no occurrences of \(\diamond\) the \(\text{Eta}\) rule applies being the reduct that \(s_e\)-normalization.

In practice we have the easy to implement rule:
\[(\text{Eta}) \ (\lambda_A.(a^A \rightarrow B 1^A \rightarrow B)^{\Gamma}_A \rightarrow s_e\text{-normalization}((a^A \rightarrow B\sigma^1 \varphi^\Gamma_A)_{A \rightarrow B}))\] if \(\diamond\) doesn’t occur in this term

**Lemma 4** The previous implementation of the \(\text{Eta}\) rule is correct.

**Proof.** (Sketch). Note firstly that \((a^A \rightarrow B\sigma^1 \varphi^\Gamma_A)_{A \rightarrow B}\) results from \(\text{Beta}\) reduction of \(((\lambda_A.a^A \rightarrow B)^{\Gamma}_A \rightarrow A \rightarrow B \diamond_{\varphi^\Gamma_A})_{A \rightarrow B}\). After propagating the \(\sigma\) operator all de Bruijin indices in the term are decremented by one except those corresponding to the variable of the outermost abstractor which are replaced with \(\varphi_0^\Gamma\diamond\). This is proved by induction on the structure of terms and the superscript of the \(\sigma\) operator that is incremented mainly via the \(\sigma\lambda\)-transition rule. Terms of the form \(\varphi_0^\Gamma\diamond\) are obtained by applying the \(\sigma\)-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 \( a_{A\rightarrow B}^{\Gamma} \). This corresponds to the condition of the original \textit{Eta} rule, since the application of \( \varphi^2_0 \) to \( \lambda \)-terms increments by one all de Bruijn indices. This can be proved by induction on the structure of terms. \( \square \)

Turning back to \( \lambda \sigma \text{-HOU} \) [5], the condition in the implementation of the \textit{Eta} rule is: \textit{“if \( \diamond \) doesn’t occur in the \( \sigma \)-normalization(\( (a_{A\rightarrow B}^{\Gamma}((\varphi^\Gamma_A \cdot \text{id}_B^\Gamma)^{\Gamma}_{A\rightarrow B})^{\Gamma}_{A\rightarrow B}) \)”}

This implementation is less efficient than in the \( \lambda s_e \)-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 \( a_{A\rightarrow B}^{\Gamma} \) we need to apply the rules \textit{Abs} and \textit{App} that are expansive, as mentioned early. More precisely, the rule \textit{Abs}, \( (\lambda_A.a)[s] \rightarrow \lambda_A.(a[1.(s \uparrow)]) \), enlarges the number of substitution objects to be marked in decorated terms from one \( (s) \) to four: \( s, \uparrow, \sigma \uparrow, \text{ and } 1.(s \uparrow) \); and the rule \textit{App}, \( (a \ b)[s] \rightarrow (a[s] \ b[s]) \), from one to two. In contrast, in the \( \lambda s_e \)-calculus the corresponding propagation of the \( \sigma \) operator is executed by applying the rules \( \sigma \)-\( \lambda \)-transition and \( \sigma \text{-app-transition} \). The \( \sigma \)-\( \lambda \)-transition, \( (\lambda_A.a)\sigma^ib \rightarrow \lambda_A.aa\sigma^{i+1}b \), does not enlarge the number of subterms to be marked. And the \( \sigma \text{-app-transition} \), \( (a_1 \ a_2)\sigma^ib \rightarrow (a_1\sigma^ib \ a_2\sigma^ib) \), increases the number of subterms to be marked by two as the \textit{App} rule, but without including substitution objects.

5 Conclusions
Following the \( \lambda \sigma \text{-HOU} \) approach introduced in [6], we have developed a pre-cooking translation that transcribes pure \( \lambda \)-terms in de Bruijn notation into \( \lambda s_e \)-terms, for which the search of grafting solutions corresponds to substitution solutions in the pure \( \lambda \)-calculus.

Our pre-cooking translation transcribes a term \( a \) by replacing each occurrence of a meta-variable \( X \) with \( \varphi^k_{0}X \) 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 [6] transcribes each occurrence of a de Bruijn index \( n \) in \( a \) into \( 1[\uparrow^{n-1}] \). Conformity of the two pre-cooking translations is therefore evident. But our proofs differ from those of [6] in that we don’t need the use of complex substitution objects because of the appropriate semantics and flexibility of the \( \varphi \) operator in the \( \lambda s_e \)-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 \( \lambda \)- and the \( \lambda s_e \)-calculus. In these proofs,
only a correct selection of the scripts for the operator \( \varphi \) 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 \( \lambda s_e \) into a description of solutions of the corresponding HOU problems in the pure \( \lambda \)-calculus.

Furthermore, by comparing the implementation of our method and that of the \( \lambda\sigma\)-HOU given in [5], we observed that pre-cooked \( \lambda \)-terms in the \( \lambda s_e \)-calculus have linear decorations on the size of the \( \lambda \)-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 \( \lambda s_e \)-calculus, where all de Bruijn indices are included.

But it is not the sole use of all de Bruijn indices that makes the \( \lambda s_e \) 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 \( \lambda s_e \)-terms smaller than those of \( \lambda\sigma \)-terms. Moreover, the size of decorated \( \lambda \)-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 \( \lambda s_e \) makes the proofs easier.

Much work remains to be done and in particular, a prototype implementation of this method is necessary (See Appendix B). Furthermore the specialization of the \( \lambda s_e \)-HOU for the important decidable and unitary fragment of the higher-order patterns has to be studied as it has been done for the \( \lambda\sigma \) in [7] and a formal distinction, from the practical point of view, between the \( \lambda s_e \)-calculus (and our procedure) and the suspension calculus developed by Nadathur and Wilson in [10] (and used in the implementation of the higher order logical programming language \( \lambda \)Prolog) should be elaborated. This is meaningful, since the \( \lambda s_e \)-calculus and the calculus of [10] have correlated nice properties. For instance the laziness in the substitution needed in implementations of \( \beta \)-reduction,
that arises naturally in the $\lambda_{se}$-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 $\lambda_{se}$-HOU approach in that practical framework.

References


A Detailed proofs

We stress here that the detailed proofs are included for the benefits of the referees and will be excluded from the final version due to the page limit of 20.

Proof. of the Proposition 1

\[ PC(a\{X_1/b_1^+, \ldots, X_p/b_p^+\}, i) = PC(a, i)\{X_1/b_1, \ldots, X_p/b_p\}g \]

is proved by induction on the structure of terms for all \( i \). Note that case \( i = 0 \) corresponds to: \( (a\{X_j/b_j\})_{pc} = PC(a\{X_j/b_j^{+0}\}, 0) = PC(a, 0)\{X_j/b_j\}g = a_{pc}\{X_j/b_j\}g \). Throughout, we use IH for the induction hypothesis.

- **\( a = \lambda b \).** Observe that

\[ PC((\lambda b)\{X_j/b_j^+\}, i) = PC((\lambda b)\{X_j/b_j^{+1}\}, i + 1) \]

By IH, \( PC((\lambda b)\{X_j/b_j^{+1}\}, i) = \lambda (PC(b(X_j/b_j^{+1}), i)) = (PC(b(X_j/b_j^{+1}), i)\{X_j/b_j\}g) = PC((\lambda b)\{X_j/b_j\})g \).

- **\( a = (a_1, a_2) \).**

\[ PC((a_1, a_2)\{X_j/b_j^+\}, i) = PC(a_1\{X_j/b_j^+\}, i)\{X_j/b_j\}g = PC(a_2\{X_j/b_j\})g \]

By IH, the former term is equal to \( \lambda PC((a_1, a_2)\{X_j/b_j\})g \). To conclude, the last expression is equal to \( (PC(\lambda b_1, i)\{X_j/b_j\})g = PC((\lambda b_1, i)\{X_j/b_j\})g \).

- **\( a = n \).**

\[ PC(n\{X_j/b_j^+\}, i) = PC(n, i)\{X_j/b_j\}g = PC(n, i)\{X_j/b_j\}g \]

- **\( a = X \).** Either \( X = X_j \) for some \( 1 \leq j \leq p \), or \( X \neq X_j \) for all \( 1 \leq j \leq p \).

We prove the interesting case where \( X = X_j \) for some \( 1 \leq j \leq p \). We should prove \( PC(b_j^+, i) = PC(X_j, i)\{X_j/b_j\}g = \varphi_{0}^{i+1}b_j \).

Proof. of the Proposition 2

Throughout we use IH for the induction hypothesis.

First, we prove by induction on \( a \) the more general fact that for all \( k \), \( (\lambda^{k+1} a) b \rightarrow_{\beta} (\lambda^{k+1} a)1/b \) implies \( (\lambda^{k+1} a) b_{pc} \rightarrow_{\lambda_{pc}} (\lambda^{k} a)\{1/b\}pc \). Our case of interest is when \( k = 0 \). Note that \( (\lambda^{k+1} a) b_{pc} = ((\lambda^{k+1} PC(a, k + 1)) b_{pc}) \rightarrow_{\lambda_{pc}} \lambda^{k}.(PC(a, k + 1)\sigma^{k+1}b_{pc}) \) and \( (\lambda^{k} a)\{1/b\} = \lambda^{k}.(a\{k + 1/b^{k}\}) \).

- **\( a = n \).** The interesting case occurs when \( n = k + 1 \): \( \lambda^{k}(k + 1)\{k + 1/b^{k}\}) = \lambda^{k}.b^{n} \).

Additionally, \( (\lambda^{k+1} n) b_{pc} = (\lambda^{k+1} n) b_{pc} \rightarrow_{\lambda_{pc}} \lambda^{k}.(n\sigma^{k+1}b_{pc}) = \)
\[ \lambda^k \varphi_0^{b^k} b_{pc} \text{ and } (\lambda^k b^k)^{pc} = \lambda^k PC(b^k, k). \] In order to prove that \( \lambda^k \varphi_0^{b^k} b_{pc} \rightarrow_{\lambda s_e} \lambda^k PC(b^k, k) \) we show the more general fact that \( \lambda^{k+i} \varphi_1^{k+i} PC(b, i) \rightarrow_{\lambda s_e} \lambda^{k+i} PC((b^i)^+, k+i) \). We use structural induction on \( b \) for all \( k > 0 \) and \( i \geq 0 \).

- \( b = m \). Firstly, \( \lambda^{k+i} \varphi_1^{k+i} PC(m, i) = \lambda^{k+i} \varphi_1^{k+i} m = \left\{ \begin{array}{ll} \lambda^{k+i} m + k, & \text{if } n > i \\ \lambda^{k+i} m, & \text{if } m \leq i \end{array} \right. \)

Secondly, by definition of \( \tilde{i} \)-lift, \( \lambda^{k+i} PC(m^{(i)+}, k+i) = \left\{ \begin{array}{ll} \lambda^{k+i} m + k, & \text{if } n > i \\ \lambda^{k+i} m, & \text{if } m \leq i \end{array} \right. \)

- \( b = X \). Either \( i = 0 \) or \( i > 0 \). If \( i = 0 \): \( \lambda^k \varphi_0^{X^k} PC(X, 0) = \lambda^k \varphi_0^{X^k} X \) and \( \lambda^k PC(X^k, k) = \lambda^k PC(X, k) = \lambda^k \varphi_0^{X^k} X \).

If \( i > 0 \): \( \lambda^{k+i} \varphi_1^{k+i} PC(X, i) = \lambda^{k+i} \varphi_1^{k+i} X \rightarrow_{\lambda s_e} \lambda^{k+i} \varphi_1^{k+i+1} X \) and \( \lambda^{k+i} PC(X, k+i) = \lambda^{k+i} \varphi_1^{k+i+1} X \).

- \( b = \lambda c \). We have that \( \lambda^{k+i} \varphi_1^{k+i} PC(\lambda c, i) = \lambda^{k+i} \varphi_1^{k+i} \lambda PC(c, i+1) \rightarrow_{\lambda s_e} \lambda^{k+i} \varphi_1^{k+i} PC(c, i+1) \). The former term \( \lambda s_e \)-derives into \( \lambda^{k+i+1} PC(b^{(i)+}, k+i+1) \) by IH.

- \( b = (b_1 b_2) \). \( \lambda^{k+i} \varphi_1^{k+i} PC(b_j, i) \rightarrow_{II} \lambda^{k+i} PC(b_j^{(i)+}, k+i) \) for \( j = 1, 2 \).

- \( a = X \). \( \lambda^k (PC(X, k+1) \sigma^{k+1} b_{pc}) = \lambda^k (\varphi_0^{X^k+1} X \sigma^{k+1} b_{pc}) \rightarrow_{\lambda s_e} \lambda^k \varphi_0^{X^k+1} X \) and \( \lambda^k PC(X\{k+1/b^k\}, k) = \lambda^k PC(X, k) = \lambda^k \varphi_0^{X^k+1} X \).

- \( a = \lambda b \). \( PC(\lambda b, k) = \lambda PC(b, k+1) \) and since \( b \) should be \( \beta \eta \)-normal, \( PC(b, k+1) \) is \( \lambda s_e \)-normal. Then \( \lambda PC(b, k+1) \) is \( \lambda s_e \)-normal too.

- \( a = (b c) \). As \( b \) and \( c \) are \( \beta \eta \)-normal, \( (PC(a, k) PC(b, k)) \) is \( \lambda s_e \)-normal.

Second. We prove the more general fact that if \( a \) is \( \beta \eta \)-normal then \( PC(a, k) \) is \( \lambda s_e \)-normal for all \( k \). By structural induction on the structure of \( a \) for all \( k \):

- \( a = \eta \) is obvious.

- \( a = X \) then \( PC(X, k) = \varphi_0^{X^k} X \) which is \( \lambda s_e \)-normal.

- \( a = \lambda b \). \( PC(\lambda b, k) = \lambda PC(b, k+1) \) and since \( b \) should be \( \beta \eta \)-normal, \( PC(b, k+1) \) is \( \lambda s_e \)-normal. Then \( PC(b, k+1) \) is \( \lambda s_e \)-normal too.

- \( a = (b c) \). As \( b \) and \( c \) are \( \beta \eta \)-normal, \( (PC(a, k) PC(b, k)) \) is \( \lambda s_e \)-normal.

Third. We prove by induction on the structure of \( a \) the more general fact that for all \( k \), if \( \lambda(\lambda^k a 1) \rightarrow_\eta d \) then \( \lambda(\lambda(\lambda^k a 1))_{pc} \rightarrow_{eta} d_{pc} \). Note that \( d \) is \( \lambda^k \) where \( (\lambda^k c)^+ = \lambda^k c^{+k} = \lambda^k a \). Then \( c^{+k} = a \). Our case of interest is for \( k = 0 \).

- \( a = n \). Note that \( \lambda(\lambda^k n 1) \rightarrow_\eta \lambda^k m \) whenever \( \lambda^k n = (\lambda^k m)^+ = \lambda^k (m^{+k}) = \) if \( m \leq k \) then \( \lambda^k m \) else \( \lambda^k m + 1 \). Then \( m = n \) if \( n \leq k \) and \( m = n-1 \) if \( n-1 > k \). Observe that \( n \neq k+1 \). Thus, \( PC(\lambda(\lambda^k n 1), 0) = \lambda(\lambda^k PC(n, k+1) 1) = \lambda(\lambda^k n 1) \rightarrow_{eta} n \) if \( n \leq k \) then \( \lambda^k n \) else if \( n-1 > k \) then \( \lambda^k n - 1 \), since \( \varphi_0^{2\lambda^k n} m = \lambda^k \varphi_0^{2\lambda^k m} = m \) if \( m \leq k \) then \( \lambda^k m \) else if \( m > k \) then \( \lambda^k m + 1 \).

- \( a = X \). Firstly, notice that \( \lambda(\lambda^k X 1) \rightarrow_\eta \lambda^k X \) since \( (\lambda^k X)^+ = \lambda^k X + k \) = \( \lambda^k X \). Note also that since \( \varphi_0^{2\lambda^k X} \varphi_0^{\lambda^k+1} X = \lambda^k \varphi_0^{\lambda^k+1} X \), we have \( PC(\lambda(\lambda^k X 1), 0) = \lambda(\lambda^k PC(X, k+1) PC(1, 1)) = \lambda(\lambda^k \varphi_0^{\lambda^k+1} X 1) \rightarrow_{eta} \lambda^k \varphi_0^{\lambda^k+1} X \).

- \( a = \lambda b \). Suppose \( \lambda(\lambda^k \varphi_0 a 1) \rightarrow_\eta \lambda^k d \). Then \( \lambda^k \varphi_0 a = (\lambda^k d)^+ = \lambda^k d^{+k} \) and \( a = d^{+k} \). Then \( PC(\lambda(\lambda^k \varphi_0 a 1), 0) = \lambda(\lambda^k PC(\lambda a, k+1) 1) \rightarrow_{eta} \eta \).
\[ \lambda^k . PC(d, k) = PC(\lambda^{k+1}.c, 0), \text{ for some } c. \]

- \( a = (b \ c) \) if \( \lambda^k.(a_1 \ a_2) = \lambda^k.(c_1 \ c_2) \) if \( \lambda^k.(a_1 \ a_2) = (\lambda^k.(c_1 \ c_2))^+ = \lambda^k.(c_1^k \ c_2^k) \) and \( c_i^k = a_i \) for \( i = 1, 2 \). Then \( \lambda.(\lambda^k.a_i) \rightarrow \eta \lambda^k.c_i \), \( i = 1, 2 \).

Thus, \( PC(\lambda.(\lambda^k.(a_1 \ a_2)), 0) = \lambda.(\lambda^k.(PC(a_1, k + 1) \ PC(a_2, k + 1)) \ 1) \) and by IH, \( \lambda.(\lambda^k.(PC(a_1, k + 1) \ PC(a_2, k + 1)) \ 1) \rightarrow_\text{eta} \lambda^k.(PC(c_1, k) \ PC(c_2, k)) = PC(\lambda^k.(c_1 \ c_2), 0) \).

Fourth. On one side, that \( a =_\eta b \) implies \( a_{pc} =_{s_c} b_{pc} \) is proved by induction on the length of the proof of \( a =_\eta b \) using the previous first and second items. On the other side, suppose that \( a_{pc} =_{s_c} b_{pc} \) and select \( a' \) and \( b' \) normal forms of \( a \) and \( b \), respectively. By previous items, terms \( a_{pc} \) and \( b_{pc} \) reduce to \( a'_{pc} \) and \( b'_{pc} \), respectively. Consequently, \( a'_{pc} =_{s_c} b'_{pc} \) and \( a'_{pc} = b'_{pc} \) since these terms should be \( s_c \)-normal. To conclude, by the fact that the pre-cooking translation is injective on \( \Lambda_{dB}(X) \), \( a' = b' \), we obtain that \( a =_\eta b. \)

**Proof of the Proposition 6**

In \( P \) every context \( \Gamma_X \) is an extension of \( \Gamma \) and every equation is well-typed in an extension of \( \Gamma \). Thus applying \( \text{Anti-Dec-}\lambda \) and \( \text{Anti-Exp-}\lambda \) and then \( \text{Replace} \) to all the variables and equations whose contexts are not \( \Gamma \) (thus strict extensions of \( \Gamma \)), we obtain an equational problem in \( \lambda s_c \) such that all equations are well-typed in the context \( \Gamma \) and also all variables occurring in the problem have context \( \Gamma \). The obtained problem is the pre-cooking of a problem in the \( \lambda \)-calculus. In fact, if \( \Gamma \) is a context and \( b \) an \( s_c \)-normal form as above whose variables have context \( \Gamma \), 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 \( \psi_{i_1}^j \ldots \psi_{i_p}^j(X, a_1, \ldots, a_p) \). 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 \( X^{s_i}a \), 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 \( \Gamma \). Consequently the term is necessarily of the form \( \varphi_i^j(X) \). Suppose that \( \Gamma = \Delta_{\leq i}.\Delta_{>i+j} \). Then \( \varphi_i^j(X) \) is of type \( A_X \) and its context corresponds to \( \Delta \), that is an extension of \( \Gamma \) 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.

**B ELAN implementation**

Like Appendix A, this appendix is included for the benefits as the referees and will be excluded from the final version due to the page limit of 20.

As for the \( \lambda\sigma \)-HOU method and its implementation in \([5]\), the rewriting logic based programming environment ELAN is adequate for the case of the \( \lambda s_c \)-calculus. To illustrate this, we show below how one can specify the language of the \( \lambda s_c \) and its decorated rules in ELAN.
The following ELAN description specifies the language of terms built from decorated terms.

@ : ( variable ) term; \ meta-variables
@ : ( int ) term; \ de Bruijn indices
la(@,@) : ( type dterm ) term; \ abstraction
ap(@,@) : ( dterm dterm ) term; \ application
s(@,@,@) : ( dterm int dterm ) term; \ sigma operator
p(@,@,@) : ( int int dterm ) term; \ phi operator

Functional types are built from atomic types via the use of the functional symbol “->”. Contexts as lists of types via the constructors “nilc” and “.”. Terms (built from decorated terms) are decorated via the operators “|” and “.".

@ : ( atomicitype ) type;
@ -> @ : ( type type ) type assocRight pri 1;
(@ -> @) : ( type type ) type alias @ -> @;
nilc : context;
@ .' @ : ( type context type ) dterm;

The decorated rules of the \( \lambda s \)-calculus can be implemented as follows:

rules for dterm
m,n : dterm;
M,N,P,Q : term;
a,b,c : type;
Ga,Om,De,Xi,Fi,Ka,Zi : context;
i,j,k : int;

global
[Si_generation]
ap(la(a, M |- a.Ga : b) |- Ga : a->b, N |- Ga : a) |- Ga : b
=> s(M |- a.Ga : b, 1, N |- Ga : a) |- Ga : b
end

[Si_La_transition]
s(la(a, M |- a.Ga : b) |- Ga : a->b, i, N |- Om : c)
  |- De : a->b
=> la(a,s(M |- a.Ga : b, i+1, N |- Om : c) |- a.De:b)
  |- De : a->b
if Om eqc greac(i,Ga)
if c eqt select(i,Ga)
if De eqc elimc(i,Ga)
\[ \lambda_s \text{-style of unification for simply-typed HOU} \]
if i<=j
if 0m eqc greac(i,Ga)
if c eqt select(i,Ga)
if De eqc elimc(i,Ga)
if Xi eqc greac(j+1,Ga)
if b eqt select(j+1,Ga)
if Fi eqc elimc(j,elimc(i,Ga)) end

[Si_Pi_transition1]
s(p(k,i,M |- 0m : a) |- Ga : a, j, N |- De : b ) |- Xi : a
=> p(k,i-1,M |- 0m : a) |- Xi : a
if j>k and j<k+i
if 0m eqc appdc(lessc(k+1,Ga),greac(k+i-1,Ga))
if De eqc greac(j,Ga)
if b eqt select(j,Ga)
if Xi eqc elimc(j,Ga) end

[Si_Pi_transition2]
s(p(k,i,M |- 0m : a) |- Ga : a, j, N |- De : b ) |- Xi : a
=> p(k,i,s(M |- 0m : a, j-i+1, N |- De : b) |- Ka : a)
    |- Xi : a
if j>=k+i
if 0m eqc appdc(lessc(k+1,Ga),greac(k+i-1,Ga))
if De eqc greac(j,Ga)
if b eqt select(j,Ga)
if Xi eqc elimc(j,Ga)
where Ka:=()appdc(lessc(k+1,Xi),greac(k+i-1,Xi)) end

[Pi_Si_transition]
p(k,i,s(M |- 0m : a, j, N |- De : b) |- Xi : a) |- Ga : a
=> s(p(k+1,i,M |- 0m : a) |- Fi : a, j, p(k+1-j,i, N |- De : b) |- Zi : b) |- Ga : a
if k+1>=j
if Xi eqc appdc(lessc(k+1,Ga),greac(k+i-1,Ga))
if 0m eqc insertc(j,b,Xi)
if De eqc greac(j,0m)
where Fi:=()insertc(j,b,Ga)
where Zi:=()greac(j,Fi) end

[Eta]
|  |- Ga : a->b
=> m where m:=(pseudon)s(M |- a.Ga : a->b,1,Dummy |- Ga : a)
    |- Ga : a->b
if not dummyIn(m) end
\[ \lambda s_e \text{-style of unification for simply-typed HOU} \]

end

Notice how the subjacent arithmetical conditions of these rules may be naturally represented in ELAN. Consider for instance the \( \sigma \lambda \text{-transition} \) of the typed \( \lambda s_e \)-calculus

\[
(\lambda_A.m)^{\sigma^i n} \rightarrow \lambda_A.(m^{\sigma^{i+1} n})
\]

its decorated version corresponds to

\[
((\lambda_A.M \vdash A.\Gamma : B) \vdash \Gamma : A \rightarrow B \sigma^i N \vdash \Gamma_{\geq i} : \Gamma_{= i}) \vdash \Gamma_{< i}.\Gamma_{> i} : A \rightarrow B \rightarrow (\lambda_A.(M \vdash A.\Gamma : B \sigma^{i+1} N \vdash \Gamma_{> i} : \Gamma_{= i}) \vdash A.\Gamma_{< i}.\Gamma_{> i} : B) \vdash \Gamma_{< i}.\Gamma_{> i} : A \rightarrow B
\]

where \( M \) and \( N \) are supposed to be \( \lambda s_e \)-terms all whose intern subterms are decorated. Using the notation of the Section 4 this can be written as

\[
((\lambda_A.M^{A.\Gamma \sigma^i}_{B} \vdash A \rightarrow B \sigma^i N^{\Gamma_{\geq i}}_{\Gamma_{= i}}\Gamma^{\Gamma_{< i} \Gamma_{> i}}_{A \rightarrow B}) \vdash (\lambda_A.(M^{A.\Gamma \sigma^{i+1} N^{\Gamma_{> i}}_{\Gamma_{= i}} \Lambda^{A.\Gamma_{< i} \Gamma_{> i}}_{B} A \rightarrow B))_{A \rightarrow B})
\]

For our ELAN implementation of this rule we use equational operators for types “eqt” as well as for contexts “eqc”. We also use constructive operators like “select(i,Ga)”, “greac(i,Ga)” and “elimc(i,Ga)” to select the \( i^{th} \) type of a context “Ga”, eliminate the first \( i \) types and the \( i^{th} \) type of the context “Ga”, respectively. Other operators like “insertc(i,a, Ga)”, “lessc(i,Ga)” and “appdc(Ga,Om)” are similarly implemented and used for inserting the type “a” at position \( i \) of a context “Ga”, for selecting the first \( i \) types of a context “Ga” and for appending contexts.

Additionally, notice how the Eta-conversion has been implemented according to the idea presented in the Lemma 4. Two points are interesting for the implementation of this rule. The first one is the strategy “pseudon” that propagates the dummy symbol \( \phi \), represented here as the Dummy constant, between the current decorated term by normalisation with respect to adequate variations of the rules involved with the substitution operator \( \sigma \) (e.g. \( \sigma \text{-destruction}, \sigma \lambda \text{-transition}, \sigma \text{-app-transition}, \sigma \sigma \text{-transition}, \sigma \varphi \text{-transition1}, \sigma \varphi \text{-transition2}) and the boolean operator dummyIn that simply check for occurrences of the Dummy constant after the previous normalisation.

From this easy implementation of the decorated rules one can verify that all arithmetical constraints involved in the \( \lambda s_e \)-calculus can be easily implemented in a rewrite logic based language like ELAN.