

A formalisation of the correspondence between Higher-Order Unification(HOU) à la Huet and HOU based on explicit substitutions calculi

Flávio L. C. de Moura^{*1}, Mauricio Ayala-Rincón^{**1} and Fairouz Kamareddine²

¹ Departamento de Matemática, Universidade de Brasília, Brasília D.F., Brasil.
flavio@mat.unb.br, ayala@mat.unb.br

² School of Mathematical and Computer Sciences, Heriot-Watt University,
Edinburgh, Scotland fairouz@macs.hw.ac.uk

Abstract. In this paper, we compare three different styles of Higher-Order Unification (HOU): the classical HOU algorithm of Huet for the pure λ -calculus in de Bruijn notation, the $\lambda\sigma$ -HOU and the λs_e -HOU algorithms for calculi of explicit substitutions. In this comparison we formally show that the $\lambda\sigma$ -HOU and the λs_e -HOU algorithms are generalisations of Huet’s algorithm, and we formally describe the relation between HOU via the $\lambda\sigma$ - and the λs_e -styles of explicit substitutions. In particular, we translate properties of Huet’s algorithm to the language of the λ -calculus in de Bruijn notation and show how the SIMPL and MATCH procedures, which form the kernel of Huet’s algorithm, can be conceived as particular strategies of the inference rules of unification via explicit substitutions calculi. For doing this, we introduce a new notation called *unification tree* which eases the presentation of Huet’s algorithm as well as the computation of the solutions.

Keywords: Higher-Order Unification, Calculi of Explicit Substitutions.

1 Introduction

Almost thirty years ago, G. Huet [Hue75,Hue02] gave the most successful and largely used Higher-Order Unification (HOU) semi-decision algorithm. The kernel of Huet’s semi-decision algorithm consists of two procedures called SIMPL and MATCH used for dealing with the so called rigid-rigid, and flexible-rigid equations, respectively, and its practical success is based on the observation that flexible-flexible equations always have solutions and consequently (for deciding whether a problem is or not unifiable) it is not necessary to explicitly present all possible unifiers. The undecidability of the HOU problem notwithstanding, Huet’s method behaves well in practice and that is the main reason why many of the modern computational systems which use HOU are based on this method, such as λ Prolog, Isabelle/HOL and Twelf and why this algorithm has been extended to several higher-order equational theories [Dow01].

* Corresponding author. Supported by Brazilian CAPES Foundation.

** Partially supported by Brazilian CNPq Council.

The most promising alternative for treating HOU problems is based on explicit substitutions calculi and was developed over the $\lambda\sigma$ -calculus almost ten years ago [DHK00]. This alternative method has been shown to be of general applicability for other calculi of explicit substitutions such as λ_{se} [ARK01]. The formal basis of some programming languages such as λ Prolog is based on explicit substitutions. This makes it more coherent to include HOU strategies directly over the associated language of explicit substitutions instead of implementing programming languages which include HOU strategies based on Huet's algorithm. Essentially, HOU via calculi of explicit substitutions consists of: firstly, precooking HOU problems presented in the language of the pure λ -calculus in de Bruijn notation; afterwards, these problems are resolved as first order unification problems modulo the equational theory which defines the calculus of explicit substitutions; and, finally, the solutions are translated back to the language of the original problem (see Fig. 1¹). Therefore, the main advantage of the use of explicit substitutions to solve HOU problems is that the substitution operation becomes a first order substitution (called grafting) and the higher-order substitutions which are solutions of the original problem can be obtained by applying the inverse of the precooking translation to the generated graftings, i.e., to the solutions of the precooked version of the original problem.

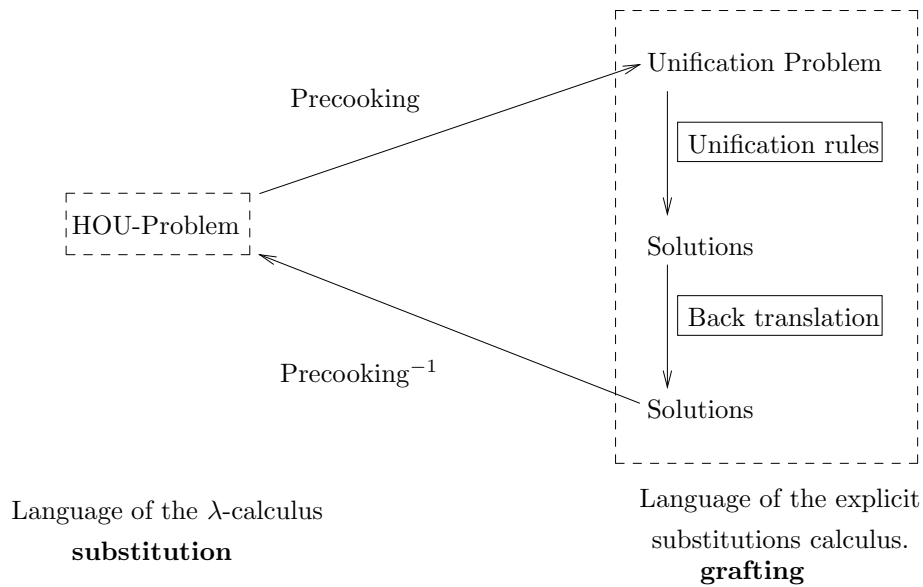


Fig. 1. HOU via calculi of Explicit substitutions.

¹ This figure comes originally from [ARK03]

In [DHK00] it has been noted that the $\lambda\sigma$ -HOU algorithm is a generalisation of Huet's method, but no complete formalisation was provided. In this paper we show that Huet's algorithm can be simulated by adopting an adequate strategy of the inference rules defined by the $\lambda\sigma$ -HOU algorithm and, in this way we formalise that the latter is a generalisation of the former. To do so, we introduce a new notation called *unification tree* which simplifies the presentation of Huet's algorithm and eases the computation of the solutions. This notation is independent of the grammar used in the λ -calculus and can be used also with the λ -calculus with names. We present the λs_e -HOU algorithm with a more compact but equivalent set of rules to those presented in [ARK01] and, finally we compare the $\lambda\sigma$ - and λs_e - styles of unification. As in the original paper [Hue75], we consider unification in the simply typed λ -calculus with and without η -conversion.

In section 2 we give some basic definitions and a brief presentation of the explicit substitutions calculi treated in this paper. In section 3 we define the *unification tree* notation which is used to describe Huet's algorithm. In section 4 we formalise the relation between Huet's algorithm and the $\lambda\sigma$ -HOU algorithm. In section 5 we describe the λs_e -HOU algorithm and then, in section 6 we formally compare the $\lambda\sigma$ - and λs_e -styles of unification. Finally, in the last section, we conclude and give directions for future work.

The current version contains, in detail, many technical proofs for the benefit of the referees which can be either omitted or sketched in a final version of the work.

2 Simply typed λ -calculus

We start this section with some basic definitions and notations used in the paper. For a more detailed explanation about the simply typed λ -calculus, de Bruijn notation, $\lambda\sigma$ -calculus and λs_e -calculus see [Bar84,dB72,ACCL91,KR97], respectively. We represent bound variables and constants by de Bruijn indexes which range over $\mathbb{N} = \{1, 2, \dots\}$, and free variables (or meta-variables) by capital letters X, Y, Z, \dots which range over the set \mathcal{X} . This separation of variables in two different classes will be important for the unification methods considered here. In fact, bound variables and constants are not concerned with the unification process and the meta-variables will play the role of the unification variables.

Definition 1. *The set $\Lambda_{dB}(\mathcal{X})$ of untyped λ -terms in de Bruijn notation is defined inductively by:*

$$a ::= \underline{n} \mid X \mid (a b) \mid \lambda. a \quad \text{where } n \in \mathbb{N} \text{ and } X \in \mathcal{X}.$$

The syntax of simply typed λ -calculus in de Bruijn notation is given by

$$\begin{array}{ll} \mathbf{Types} & A ::= K \mid A \rightarrow B \\ \mathbf{Contexts} & \Gamma ::= \text{nil} \mid A. \Gamma \\ \mathbf{Terms} & a ::= \underline{n} \mid X \mid (a b) \mid \lambda_A. a \quad \text{where } n \in \mathbb{N} \text{ and } X \in \mathcal{X}. \end{array}$$

The type of the term a is indicated by $\tau(a)$.

Definition 2. The updating functions $U_k^i : \Lambda_{dB}(\mathcal{X}) \rightarrow \Lambda_{dB}(\mathcal{X})$, for $k \geq 0$ and $i \geq 1$ are defined inductively by:

$$\begin{aligned} (a) \ U_k^i(X) &= X, \text{ for } X \in \mathcal{X} & (b) \ U_k^i(a \ b) &= U_k^i(a) \ U_k^i(b) \\ (c) \ U_k^i(\lambda_A.a) &= \lambda_A.U_{k+1}^i(a) & (d) \ U_k^i(\underline{n}) &= \begin{cases} \underline{n} + \underline{i} - \underline{1}, & \text{if } n > k \\ \underline{n}, & \text{if } n \leq k \end{cases} \end{aligned}$$

Definition 3. Let $a \in \Lambda_{dB}(\mathcal{X})$, $i \in \mathbb{N}$. The term a^+ , called lift of a , is defined by $a^+ = a^{+0}$, where a^{+i} is inductively defined by:

$$\begin{aligned} (a) \ X^{+i} &= X, \text{ for } X \in \mathcal{X} & (b) \ (a \ b)^{+i} &= a^{+i} \ b^{+i} \\ (c) \ (\lambda_A.a)^{+i} &= \lambda_A.a^{+(i+1)} & (d) \ \underline{n}^{+i} &= \begin{cases} \underline{n} + \underline{1}, & \text{if } n > i \\ \underline{n}, & \text{if } n \leq i \end{cases} \end{aligned}$$

Definition 4. The application of the substitution of b for $n \in \mathbb{N}$ on a term a in $\Lambda_{dB}(\mathcal{X})$, denoted by $a\{\underline{n}/b\}$ is defined inductively by:

$$\begin{aligned} (a) \ X\{\underline{n}/b\} &= X, \text{ for } X \in \mathcal{X} & (b) \ (a_1 \ a_2)\{\underline{n}/b\} &= (a_1\{\underline{n}/b\})(a_2\{\underline{n}/b\}) \\ (c) \ (\lambda_A.a)\{\underline{n}/b\} &= \lambda_A.a\{\underline{n} + \underline{1}/b^+\} & (d) \ \underline{m}\{\underline{n}/b\} &= \begin{cases} \underline{m} - \underline{1}, & \text{if } m > n \\ b, & \text{if } m = n \\ \underline{m}, & \text{if } m < n \end{cases} \end{aligned}$$

Definition 5. Let $\bar{\theta}$ be a valuation (i.e., a function) from \mathcal{X} to $\Lambda_{dB}(\mathcal{X})$. The associated substitution θ is defined by the rules:

$$\begin{aligned} (a) \ X\theta &= X\bar{\theta} & (b) \ \underline{n}\theta &= \underline{n} & (c) \ (a \ b)\theta &= a\theta \ b\theta & (d) \ (\lambda_A.a)\theta &= \lambda_A.(a\theta^+) \end{aligned}$$

where $\theta^+ := \{X_1/a_1^+, \dots, X_n/a_n^+\}$, when $\theta = \{X_1/a_1, \dots, X_n/a_n\}$.

In the λ -calculus, normal forms and η -long forms play an important role:

Definition 6. 1. Every λ -term in β -normal form (β -nf) has the form

$$\lambda_{A_1} \dots \lambda_{A_n} . (h \ e_1 \dots e_p)$$

where $n, p \geq 0$, h is a variable (or a constant) called the head of the term and e_1, \dots, e_p are λ -terms in β -nf called its arguments. The $\lambda_{A_1}, \dots, \lambda_{A_n}$ are the external abstractors of the term and the sub-term $\lambda_{A_1} \dots \lambda_{A_n} . h$ is its heading. The head of the term h is said to be bound if it is a de Bruijn index, say \underline{k} , with $1 \leq k \leq n$, or free if it is a meta-variable; the head h is a constant if $k > n$.

2. A λ -term in β -nf is rigid if its head is a constant or a bound variable. Otherwise, i.e., if it is a meta-variable, the term is flexible.
3. Let $a \in \Lambda_{dB}(\mathcal{X})$ be a λ -term in de Bruijn notation of type $A_1 \rightarrow \dots \rightarrow A_m \rightarrow B$ with B atomic. The η -long form of a β -nf term a , written a' , is inductively defined as follows:
 - if $a = \lambda_A.b$ then $a' = \lambda_A.b'$.
 - if $a = (\underline{n} \ b_1 \dots b_q)$ then $a' = \lambda_{A_1} \dots \lambda_{A_m} . (\underline{n} + \underline{m} \ c_1 \dots c_q \ \underline{m}' \dots \underline{1}')$, where c_j ($1 \leq j \leq q$) is the η -long form of the normal form of $U_0^{m+1}(b_j)$.
 - if $a = (X \ b_1 \dots b_q)$ then $a' = \lambda_{A_1} \dots \lambda_{A_m} . (X \ c_1 \dots c_q \ \underline{m}' \dots \underline{1}')$, where c_j ($1 \leq j \leq q$) is the η -long form of the normal form of $U_0^{m+1}(b_j)$.

2.1 The $\lambda\sigma$ -calculus

The λ -calculus is based on a notion of substitution that belongs to a meta-language. Such a notion is necessary because the substitution process adopts renaming of bound variables in order to avoid variable capture. A natural solution to define a substitution which belongs to the language is to extend the language and turn the substitution process explicit. The first mechanism that explicitated the substitutions was the $\lambda\sigma$ -calculus [ACCL91] that we briefly present in the following. The rewriting rules of the $\lambda\sigma$ -calculus are given in Table 1.

<i>(Beta)</i>	$(\lambda.a)b \longrightarrow a [b \cdot id]$
<i>(App)</i>	$(a b)[s] \longrightarrow (a [s]) (b [s])$
<i>(Abs)</i>	$(\lambda.a)[s] \longrightarrow \lambda(a [\underline{1} \cdot (s \circ \uparrow)])$
<i>(Clos)</i>	$(a [s])[t] \longrightarrow a [s \circ t]$
<i>(VarCons)</i>	$\underline{1}[a \cdot s] \longrightarrow a$
<i>(Id)</i>	$a[id] \longrightarrow a$
<i>(Assoc)</i>	$(s \circ t) \circ u \longrightarrow s \circ (t \circ u)$
<i>(Map)</i>	$(a \cdot s) \circ t \longrightarrow a [t] \cdot (s \circ t)$
<i>(IdL)</i>	$id \circ s \longrightarrow s$
<i>(IdR)</i>	$s \circ id \longrightarrow s$
<i>(ShiftCons)</i>	$\uparrow \circ (a \cdot s) \longrightarrow s$
<i>(VarShift)</i>	$\underline{1} \cdot \uparrow \longrightarrow id$
<i>(SCons)</i>	$\underline{1}[s] \cdot (\uparrow \circ s) \longrightarrow s$
<i>(Eta)</i>	$\lambda.(a \underline{1}) \longrightarrow b$ if $a =_{\sigma} b[\uparrow]$

Table 1. The $\lambda\sigma$ Rewriting System with **Eta**-conversion

Definition 7. *The syntax of typed $\lambda\sigma$ -calculus is given by*

Types	$A ::= K \mid A \rightarrow B$	
Contexts	$\Gamma ::= nil \mid A.\Gamma$	
Terms	$a ::= \underline{1} \mid X \mid (a b) \mid \lambda_A.a \mid a[s]$	where $X \in \mathcal{X}$
Substitutions	$s ::= id \mid \uparrow \mid a.s \mid s \circ s$	

The typing rules are as follows:

$$\begin{array}{ll}
(\text{var}) & A.\Gamma \vdash \underline{1} : A \\
(\text{app}) & \frac{\Gamma \vdash a : A \rightarrow B \quad \Gamma \vdash b : A}{\Gamma \vdash (a \ b) : B} \\
(\text{id}) & \Gamma \vdash \text{id} \triangleright \Gamma \\
(\text{cons}) & \frac{\Gamma \vdash a : A \quad \Gamma \vdash s \triangleright \Gamma'}{\Gamma \vdash a.s \triangleright A.\Gamma'} \\
(\text{lambda}) & \frac{A.\Gamma \vdash a : B}{\Gamma \vdash \lambda_A.a : A \rightarrow B} \\
(\text{clos}) & \frac{\Gamma \vdash s \triangleright \Gamma' \quad \Gamma' \vdash a : A}{\Gamma \vdash a[s] : A} \\
(\text{shift}) & A.\Gamma \vdash \uparrow \triangleright \Gamma \\
(\text{comp}) & \frac{\Gamma \vdash s'' \triangleright \Gamma'' \quad \Gamma'' \vdash s' \triangleright \Gamma'}{\Gamma \vdash s' \circ s'' \triangleright \Gamma'}
\end{array}$$

In addition, to each meta-variable X we associate a unique type A and, we assume that for each type there exists an infinite set of meta-variables with that type. We add the following type rule for meta-variables:

$$(\text{meta}) \quad \Gamma \vdash X : A, \quad \text{where } \Gamma \text{ is any context.}$$

In this calculus, when a substitution s is applied to a term a we internalise this as $a[s]$. Simultaneous substitutions are represented as lists of terms with the usual operator *cons* (written as “.”) and an operator for the empty list (written *id* which represents the identity substitution) and the operator \uparrow which represents the infinite substitution $\underline{2}, \underline{3}, \dots$. Although the $\lambda\sigma$ -calculus codifies the de Bruijn index \underline{n} as $\underline{1}[\uparrow^{n-1}]$, for the sake of clarity, we will not use such a codification.

Definition 8. Let $a \in \Lambda_{\lambda\sigma}(\mathcal{X})$. We inductively define $|a|$, the size of a , by:

- if $a = X$ or $a = \underline{1}$ then $|a| = 1$
- if $a = (b \ c)$ then $|a| = |b| + |c|$
- if $a = \lambda.b$ then $|a| = 1 + |b|$
- if $a = b[s]$ then $|a| = |b| + ||s||$, where the size of a substitution s , written as $||s||$, is inductively defined as:
 - if $s = \uparrow$ or $s = \text{id}$ then $||s|| = 0$
 - if $s = c.d$ then $||s|| = |c| + ||d||$
 - if $s = u \circ v$ then $||s|| = ||u|| + ||v||$

The notions of $\lambda\sigma$ -normal form and **Eta**-long form in the $\lambda\sigma$ -calculus are given in what follows:

Proposition 1 ([Río93]). Any $\lambda\sigma$ -term in normal form is of one of the following forms:

1. $\lambda.a$, where a is in $\lambda\sigma$ -normal form.
2. $(a \ b_1 \dots b_q)$, where a and b_i are in $\lambda\sigma$ -normal form and a is either $\underline{1}$, $\underline{1}[\uparrow^n]$, X or $X[s]$ where s is a substitution term in $\lambda\sigma$ -normal form and different from *id*.
3. $a_1 \dots a_p. \uparrow^n$, where a_1, \dots, a_p are in $\lambda\sigma$ -normal form and $a_p \neq n$.

Definition 9 (Eta-long form [DHK00]). Let a be a $\lambda\sigma$ -term of type $A_1 \rightarrow \dots \rightarrow A_n \rightarrow B$ in context Γ and in $\lambda\sigma$ -normal form. The **Eta-long form** of a , written as a' , is defined by:

1. If $a = \lambda_A.b$ then $a' = \lambda_A.b'$.
2. If $a = (\underline{\mathbf{k}} b_1 \dots b_q)$ then $a' = \lambda_{A_1} \dots \lambda_{A_n} . (\underline{\mathbf{k} + \mathbf{n}} c_1 \dots c_q \underline{\mathbf{n}' \dots \mathbf{1}'})$, where c_i is the **Eta-long form** of the normal form of $b_i[\uparrow^n]$.
3. If $a = (X[s] b_1 \dots b_q)$ then $a' = \lambda_{A_1} \dots \lambda_{A_n} . (X[s'] c_1 \dots c_q \underline{\mathbf{n}' \dots \mathbf{1}'})$, where c_i is the **Eta-long form** of the normal form of $b_i[\uparrow^n]$ and if $s = d_1 \dots d_r . \uparrow^k$ then $s' = e_1 . \dots . e_r . \uparrow^{k+n}$ where e_i is the **Eta-long form** of $d_i[\uparrow^n]$.

2.2 The λ_{s_e} -calculus

The λs -style [KR95] of explicit substitutions uses a philosophy of de Bruijn's *Automath* [NGdV94] elaborated in the new item notation [KN96]. The philosophy states that terms are built by applications (a function applied to an argument), abstraction (a function), substitution or updating. The advantages of this philosophy include remaining as close as possible to the familiar λ -calculus. The rewriting rules of the λ_{s_e} -calculus [KR97] are given in Table 10.

Definition 10. Terms of the λ_{s_e} -calculus are given by:

$\Lambda_{s_e} ::= \underline{\mathbf{n}} \mid X \mid \Lambda_{s_e} \Lambda_{s_e} \mid \lambda . \Lambda_{s_e} \mid \Lambda_{s_e} \sigma^j \Lambda_{s_e} \mid \varphi_k^i \Lambda_{s_e}$, where $n, j, i \geq 1$, $k \geq 0$ and $X \in \mathcal{X}$.

The typing rules for the λ_{s_e} -calculus are as follows:

$$\begin{array}{ll}
 \text{(var)} & A . \Gamma \vdash \underline{\mathbf{1}} : A \\
 \text{(varn)} & \frac{\Gamma \vdash \underline{\mathbf{n}} : B}{A . \Gamma \vdash \underline{\mathbf{n} + \mathbf{1}} : B} \\
 \text{(app)} & \frac{\Gamma \vdash a : A \rightarrow B \quad \Gamma \vdash b : A}{\Gamma \vdash (a b) : B} \\
 \text{(lambda)} & \frac{A . \Gamma \vdash a : B}{\Gamma \vdash \lambda_A . a : A \rightarrow B} \\
 \text{(sigma)} & \frac{\Gamma_{\geq i} \vdash b : B \quad \Gamma_{< i} . B . \Gamma_{\geq i} \vdash a : A}{\Gamma \vdash a \sigma^i b : A} \\
 \text{(phi)} & \frac{\Gamma_{\leq k} . \Gamma_{\geq k+i} \vdash a : A}{\Gamma \vdash \varphi_k^i a : A}
 \end{array}$$

In addition, to each meta-variable X we associate a unique type A and, we assume that for each type there exists an infinite set of meta-variables with that type. We add the following type rule for meta-variables:

$$\text{(meta)} \quad \Gamma \vdash X : A, \quad \text{where } \Gamma \text{ is any context.}$$

Definition 11 (λ_{s_e} -normal form [ARK03]). Let a be a λ_{s_e} -term. Then a is in λ_{s_e} -normal form iff:

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

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

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

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

Definition 12 (Eta-long form [ARK03]). Let a be a λs_e -term of type $A_1 \rightarrow \dots A_n \rightarrow B$ in the environment Γ and in λs_e -normal form. The Eta-long form of a , written a' , is inductively defined by:

- if $a = \lambda_A.b$ then $a' = \lambda_A.b'$
- if $a = (b_1 \dots b_p)$ then $a' = \lambda_{A_1} \dots \lambda_{A_n}.(c_1 \dots c_p \underline{n}' \dots \underline{1}')$, where c_i is the Eta-long form of the λs_e -normal form of $\varphi_0^{n+1} b_i$.
- if $a = b \sigma^j c$ then $a' = \lambda_{A_1} \dots \lambda_{A_n}.(d' \sigma^{j+i} e' \underline{n}' \dots \underline{1}')$, where d' and e' are the Eta-long form of the λs_e -normal form of $\varphi_0^{n+1} b$ and $\varphi_0^{n+1} c$, respectively.
- if $a = \varphi_k^i b$ then $a' = \lambda_{A_1} \dots \lambda_{A_n}.(c' \varphi_k^i \underline{n}' \dots \underline{1}')$, where c' is the Eta-long form of the λs_e -normal form of $\varphi_0^{n+1} b$.

3 Unification Trees: a structured presentation of Huet's HOU algorithm

In this section, we introduce the *unification tree* structure for giving an adequate presentation of Huet's algorithm over the λ -calculus in de Bruijn notation. By using this structure we can exhibit the interconnection between the two main procedures of Huet's algorithm naturally. This clarifies the description and simplifies the comparison between explicit substitutions based HOU styles and Huet's method. We start with the definition of a unification problem.

Definition 13. *A unification problem in the pure λ -calculus is a conjunction of equations of the form $a =^? b$, where a and b are two λ -terms of the same type. One such equation is called rigid-rigid (resp. flexible-flexible) if both a and b are rigid (resp. flexible) terms, and flexible-rigid if a is flexible and b is rigid or vice-versa. A equation of the form $a =^? a$ is called trivial.*

Definition 14. *Given a unification problem $a_1 =^? b_1 \wedge \dots \wedge a_n =^? b_n$, a solution (or a unifier) for this problem is a substitution σ such that the normal form of $a_i\sigma$ is equal, up to renaming of meta-variables, to the normal form of $b_i\sigma$, for all $i = 1, \dots, n$. In other words, $a_i\sigma =_{\beta\eta} b_i\sigma, \forall i = 1, \dots, n$.*

In addition to Definition 14, it is important to remark that the HOU problem is concerned with finding *all* distinct substitutions that are solutions of a given unification problem.

The unification tree notation is obtained from the matching tree of Huet [Hue75] by adding labels to the unification problems as well as to the generated substitutions. These labels provide information about the position of the unification problems and of the substitutions in the matching tree (see Fig. 2).

A unification tree for a given unification problem P is given by:

1. Label P with ϵ (the empty position) as a subscript, i.e., P_ϵ . This subscript means that this problem is labeling the root of the matching tree.
2. For a node labeled with P_q , its sibling node is labeled with \overline{P}_q whenever the unification problem derives by applying the procedure SIMPL defined in the next section. This step is represented by a curly line in the unification tree since the subscript remains the same after a simplification step.
3. For a node labeled with P_q containing a flexible-rigid equation, call $\sigma_{q1}, \sigma_{q2}, \dots, \sigma_{qk}$ the incremental substitutions² generated by an application of the procedure MATCH to this equation.
4. The sibling nodes of P_q , written P_{q1}, \dots, P_{qk} are defined by the composition $P_{qi} := \overline{P}_q\sigma_{qi}$, for $i = 1, \dots, k$.

Using this notation is straightforward to conclude that the substitution σ_{12315} was generated by the unification problem P_{1231} . The solutions of unification

² An *incremental substitution* is a substitution that knows only about the heading of the term to be substituted; all the internal structure of the term is already unknown and, therefore it is represented by meta-variables.

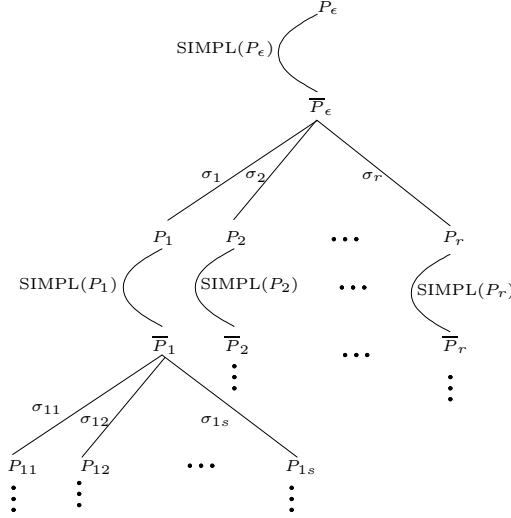


Fig. 2. The Unification Tree

problems can be easily computed by composing the generated incremental substitutions from the root of the unification tree to a success leaf. For instance, if P_{1223} is a success node but P_{122} is not, then the substitution solution corresponding to this success path is given by the composition $\sigma_1\sigma_{12}\sigma_{122}\sigma_{1223}$. In the following we describe Huet’s algorithm using the unification tree notation.

3.1 The procedure SIMPL

The procedure SIMPL receives as argument a unification problem P_q containing at least one rigid-rigid equation and returns either a terminal (success or failure) status or an equivalent “simplified” unification problem, called \overline{P}_q , containing at least one flexible-rigid equation. In the following we give the algorithmic description of SIMPL.

Procedure SIMPL

INPUT: A unification problem P_q with at least one rigid-rigid equation, called the *selected equation*:

$$\lambda_{A_1} \dots \lambda_{A_r} . (\underline{n} e_1^1 \dots e_{p_1}^1) =^? \lambda_{A_1} \dots \lambda_{A_s} . (\underline{m} e_1^2 \dots e_{p_2}^2) \wedge P'$$

where $s, r, p_1, p_2 \geq 0$ and $n, m > 0$.

OUTPUT: A terminal (success or a failure) status or an equivalent unification problem \overline{P}_q (without rigid-rigid equations) and containing at least one flexible-rigid equation.

WHILE there exists a rigid-rigid equation in P_q DO

If $r \neq s$ or $n \neq m$ then stop and report a failure status else let $p = p_1 = p_2$ (it must be the case since the terms have the same type) and replace the selected equation by the conjunction $\lambda_{A_1} \dots \lambda_{A_r}.e_1^1 =? \lambda_{A_1} \dots \lambda_{A_r}.e_1^2 \wedge \dots \wedge \lambda_{A_1} \dots \lambda_{A_r}.e_p^1 =? \lambda_{A_1} \dots \lambda_{A_r}.e_p^2$ in P_q and call the result \overline{P}_q .

DONE.

If there is a flexible-rigid equation in \overline{P}_q then return \overline{P}_q else stop and report a success status.

The same description is used for both β - and $\beta\eta$ -unification. In fact, in the λ -calculus with η -conversion, we always have $r = s$ because terms are always in η -long form.

3.2 The Procedure MATCH

The procedure MATCH takes a flexible-rigid equation as argument and returns a finite set of substitutions called Σ . We start describing the procedure MATCH for β -unification. In this case, a necessary condition to a flexible-rigid equation to be unifiable is that the heading of the rigid term must have at least the same number of abstractors than the heading of the flexible one. This is a consequence of the fact that substitutions never decrease the number of external abstractors in the flexible term. Hence, without loss of generality, we may assume that the input of the procedure MATCH is a flexible-rigid equation of the form:

$$\lambda_{A_1} \dots \lambda_{A_r}.(X e_1^1 \dots e_{p_1}^1) =? \lambda_{A_1} \dots \lambda_{A_{r+r'}}.(\underline{n} e_1^2 \dots e_{p_2}^2) \quad (1)$$

where $\tau(X) = B_1 \rightarrow \dots \rightarrow B_{p_1} \rightarrow A_{r+1} \rightarrow A_{r+r'} \rightarrow C$, where $p_1, p_2, r, r' \geq 0$, $n > 0$ and C is the type of the term $(\underline{n} e_1^2 \dots e_{p_2}^2)$. The type of X must have the above form so that we may be sure that both terms in equation (1) have the same type.

The procedure MATCH is based on two rules named *imitation* and *projection*. For a given flexible-rigid equation, the procedure MATCH calls the imitation rule and the projection rule and it returns a set Σ of incremental substitutions formed by the union of the substitutions generated by the imitation with the ones generated by the projection rules. As we will see in the description of the main procedure, if Σ is the empty set then the algorithm will stop and report a failure status. In the following we give the description of the imitation and projection rules.

The Imitation Rule. The *imitation* rule consists in substituting the head of the flexible term by another term with the same head as the rigid one. In this sense, it tries to imitate the head of the rigid term. Considering the equation (1), we can imitate the term $\lambda_{A_1} \dots \lambda_{A_{r+r'}}.(\underline{n} e_1^2 \dots e_{p_2}^2)$ by substituting the meta-variable X by a term with heading $\lambda_{C_1} \dots \lambda_{C_w}.\underline{t}$, for adequate types and an adequate w and where \underline{t} corresponds to the de Bruijn index \underline{n} . As variable

capture is forbidden in the λ -calculus, substitutions have some restrictions to work correctly.

For β -unification, Huet considers two cases for equation (1):

1. CASE $r' > 0$. Here, \underline{n} can be either a constant ($n > r+r'$) or a bound variable with $1 \leq n \leq r'$. In both cases, the term to be substituted for X has $p_1 + r'$ external abstractors so that the resulting term will have, after normalisation, the desired $r + r'$ external abstractors. If \underline{n} is a constant then we need to substitute X by a term with heading $\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \cdot \underline{k}$ where \underline{k} must be equal to \underline{n} after normalisation. When this substitution goes inside an abstractor, each of its free variables must be increased by one in order to avoid capture. Since this substitution needs to go inside r abstractors, these free variables need to be increased by r . After that, we need to apply p_1 steps of β -reduction to normalise the term. After each application of a β -reduction, its free variables are decreased by 1. Therefore, in order to guarantee that the normalised term has the same head of the right hand-side of the given equation, we need that $k = p_1 + n - r$. If \underline{n} is a bound variable with $1 \leq n \leq r'$ then it remains unchanged during this process. This suggests the substitution:

$$X / \lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \cdot (\underline{t} (X_1 \underline{p_1 + r'} \dots \underline{1}) \dots (X_{p_2} \underline{p_1 + r'} \dots \underline{1})) \quad (2)$$

where X_1, \dots, X_{p_2} are fresh meta-variables of the appropriate type and

$$t = \begin{cases} p_1 + n - r, & \text{if } \underline{n} \text{ is a constant;} \\ n, & \text{otherwise.} \end{cases}$$

2. CASE $r' = 0$. Now \underline{n} must be a constant (to avoid variable capture) and equation (1) assumes the form

$$\lambda_{A_1} \dots \lambda_{A_r} \cdot (X e_1^1 \dots e_{p_1}^1) =? \lambda_{A_1} \dots \lambda_{A_r} \cdot (\underline{n} e_1^2 \dots e_{p_2}^2) \quad (3)$$

A natural substitution obtained directly from the previous ones is:

$$X / \lambda_{B_1} \dots \lambda_{B_{p_1}} \cdot (\underline{p_1 + n - r} (X_1 \underline{p_1} \dots \underline{1}) \dots (X_{p_2} \underline{p_1} \dots \underline{1}))$$

where X_1, \dots, X_{p_2} are fresh meta-variables of the appropriate type. This substitution replaces all the p_1 arguments of X by terms whose heads are the meta-variables X_1, \dots, X_{p_2} (after p_1 steps of β -reduction) and we may investigate the existence of other substitutions which keep some of these arguments unchanged. This means that we intend to substitute X by terms with heading $\lambda_{B_1} \dots \lambda_{B_j} \cdot \underline{j + n - r}$, where $0 \leq j \leq p_1$. This is possible if there exists $0 \leq j \leq p_1$ such that:

- (a) $p_2 - p_1 + j \geq 0$, i.e., it is possible to complete the p_2 arguments of the rigid term, and;
- (b) $\tau(e_k^1) = \tau(e_{p_2 - p_1 + k}^2)$ for all $j < k \leq p_1$, i.e., the unchanged arguments of the flexible term have the same type of the corresponding arguments of the rigid one.

And so, for each $0 \leq j \leq p_1$ satisfying the conditions (a) and (b) we include the following substitutions in Σ :

$$X/\lambda_{B_1} \dots \lambda_{B_j} \cdot (\underline{j + \mathbf{n} - \mathbf{r}} (X_1 \underline{j} \dots \underline{1}) \dots (X_{p_2 - p_1 + j} \underline{j} \dots \underline{1}))$$

where $X_1, \dots, X_{p_2 - p_1 + j}$ are meta-variables of the appropriate types.

In $\beta\eta$ -unification, all meta-variables are substituted by η -long terms. Therefore λ -terms of the same type always have the same number of external abstractors and hence, without loss of generality, we can restrict the analysis to flexible-rigid equations of the form

$$\lambda_{A_1} \dots \lambda_{A_r} \cdot (X e_1^1 \dots e_{p_1}^1) \stackrel{?}{=} \lambda_{A_1} \dots \lambda_{A_r} \cdot (\underline{\mathbf{n}} e_1^2 \dots e_{p_2}^2)$$

where $\tau(X) = B_1 \rightarrow \dots \rightarrow B_{p_1} \rightarrow B$ (B atomic). The possible substitutions for X , will be exactly the η -long terms of this type whose head corresponds to the head of the rigid term. In general, higher order unification with η -conversion eliminates redundancies in the search for solutions (see Example 1). In this case, the substitution generated by the imitation rule is given by:

$$X/\lambda_{B_1} \dots \lambda_{B_{p_1}} \cdot (\underline{\mathbf{p}_1 + \mathbf{n} - \mathbf{r}} (X_1 \underline{\mathbf{p}_1} \dots \underline{1}) \dots (X_{p_2} \underline{\mathbf{p}_1} \dots \underline{1}))$$

where X_1, \dots, X_{p_2} are meta-variables with appropriate type and all sub-terms are in η -normal form. The imitation substitution in the $\beta\eta$ -unification is unique and all the substitutions generated in the β -unification are particular instances of the former. This fact can be formalised by the following theorem:

Theorem 1. *Let P be a unification problem containing the flexible-rigid equation as in (1). If $\sigma_1, \dots, \sigma_k$ ($k \geq 0$) are the substitutions generated by the imitation rule in the β -calculus and γ the substitution generated by the imitation rule in the $\beta\eta$ -calculus for the η -long form of the given equation, then there exist substitutions $\theta_1, \dots, \theta_k$ such that $\gamma\theta_i =_{\beta\eta} \sigma_i$, for all $1 \leq i \leq k$, i.e., the substitutions generated in the β -calculus are particular instances of the substitution generated in the $\beta\eta$ -calculus.*

Proof. Without loss of generality, we may assume that in the $\beta\eta$ -calculus, the given equation assumes the form

$$\lambda_{A_1} \dots \lambda_{A_{r+r'}} \cdot (X e_1^1 \dots e_{p_1}^1 \underline{\mathbf{r}'} \dots \underline{1}) \stackrel{?}{=} \lambda_{A_1} \dots \lambda_{A_{r+r'}} \cdot (\underline{\mathbf{n}} e_1^2 \dots e_{p_2}^2).$$

As usual, we assume that all sub-terms are in η -long form although some of them remain with the same name. The imitation generated in this case is given by

$$\gamma = \{X/\lambda_{B_1} \dots \lambda_{B_{p_1+r'}} \cdot (\underline{\mathbf{p}_1 + \mathbf{n} - \mathbf{r}} (Y_1 \underline{\mathbf{p}_1 + \mathbf{r}'} \dots \underline{1}) \dots (Y_{p_2} \underline{\mathbf{p}_1 + \mathbf{r}'} \dots \underline{1}))\}.$$

If $r' > 0$ then the sole imitation generated in the β -calculus is given by $\sigma = \{X/\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \cdot \underline{\mathbf{p}_1 + \mathbf{n} - \mathbf{r}} (X_1 (\underline{\mathbf{p}_1 + \mathbf{r}'} \dots \underline{1}) \dots X_{p_2} (\underline{\mathbf{p}_1 + \mathbf{r}'} \dots \underline{1}))\}$. Therefore, we have the result, if we consider θ to be the empty substitution.

If $r' = 0$ then $\tau(X) = B_1 \rightarrow \dots \rightarrow B_{p_1} \rightarrow B$ and the considered flexible-rigid equation has the form

$$\lambda_{A_1} \dots \lambda_{A_r} \cdot (X e_1^1 \dots e_{p_1}^1) \stackrel{?}{=} \lambda_{A_1} \dots \lambda_{A_r} \cdot (\underline{\mathbf{n}} e_1^2 \dots e_{p_2}^2).$$

In this case, the substitution γ is given by $\gamma = \{X/\lambda_{B_1} \dots \lambda_{B_{p_1}}.(\underline{p_1 + n - r}(Y_1 \underline{p_1} \dots \underline{1}) \dots (Y_{p_2} \underline{p_1} \dots \underline{1}))\}$ and the corresponding imitations in the β -calculus are given by

$$\sigma_j = \{X/\lambda_{B_1} \dots \lambda_{B_j}.(\underline{j + n - r}(X_1 \underline{j} \dots \underline{1}) \dots (X_{p_2 - p_1 + j} \underline{j} \dots \underline{1}))\}$$

for each $0 \leq j \leq p_1$ such that $p_2 - p_1 + j \geq 0$ and $\tau(e_k^1) = \tau(e_{p_2 - p_1 + k}^2)$ for all $j < k \leq p_1$. Now consider the substitution:

$$\theta_j = \{Y_1/\lambda_{B_1} \dots \lambda_{B_{p_1}}.(X_1 \underline{p_1} \dots \underline{p_1 - j + 1}), \dots, \\ Y_{p_2 - p_1 + j}/\lambda_{B_1} \dots \lambda_{B_{p_1}}.(X_{p_2 - p_1 + j} \underline{p_1} \dots \underline{p_1 - j + 1}), \\ Y_{p_2 - p_1 + j + 1}/\lambda_{B_1} \dots \lambda_{B_{p_1}}.\underline{p_1 - j}, \dots, Y_{p_2}/\lambda_{B_1} \dots \lambda_{B_{p_1}}.\underline{1}\}$$

which is well defined by the above type condition.

After a β -normalisation, we get

$$\gamma\theta_j = \{X/\lambda_{B_1} \dots \lambda_{B_{p_1}}.(\underline{p_1 + n - r}(X_1 \underline{p_1} \dots \underline{p_1 - j + 1}) \dots \\ (X_{p_2 - p_1 + j} \underline{p_1} \dots \underline{p_1 - j + 1}) \underline{p_1 - j} \dots \underline{1})\}$$

And finally, η -reducing, we get (up to the renaming of meta-variables):

$$\gamma\theta_j = \{X/\lambda_{B_1} \dots \lambda_{B_j}.(\underline{j + n - r}(X_1 \underline{j} \dots \underline{1}) \dots (X_{p_2 - p_1 + j} \underline{j} \dots \underline{1}))\} = \sigma_j$$

□

In this way, different substitutions in the β -calculus may correspond to the same ones in the $\beta\eta$ -calculus due to η -equivalence.

The Projection Rule. A *projection* can be used in case the head of the rigid term is a constant or a bound variable with $r' \leq n \leq r + r'$ (again, consider the structure of equation (1)). Note that in the last case an imitation does not apply due to variable capture and, the projection rule consists in “projecting” the head of the flexible term onto one of its arguments which eventually contains the index that corresponds to the variable \underline{n} .

In β -unification, there are two possible headings for X : $\lambda_{B_1} \dots \lambda_{B_j}.\underline{i}$ or $\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+j}}.\underline{i}$, for adequate de Bruijn index \underline{i} and subscript j . In fact, the first heading is used for each $1 \leq i \leq j \leq p_1$, such that there exists $m \geq 0$ with

$$\tau(e_i^1) = C_1 \rightarrow \dots \rightarrow C_m \rightarrow B_{j+1} \rightarrow \dots \rightarrow B_{p_1 + r'} \rightarrow C$$

In this case, X is projected over its i -th argument e_i^1 by including in Σ the following substitution: $X/\lambda_{B_1} \dots \lambda_{B_j}.(\underline{j - i + 1}(Y_1(\underline{j} \dots \underline{1}) \dots (Y_m \underline{j} \dots \underline{1})))$ where Y_1, \dots, Y_m are fresh meta-variables with appropriate type. Note that, in this case, we have, at most $\frac{p_1(p_1+1)}{2}$ possible distinct projections.

Projections with heading $\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+j}}.\underline{k}$ are generated whenever $r' > 0$ in equation (1), where $1 \leq j \leq r'$ and \underline{k} is a de Bruijn index bound by one of the $\lambda_{B_1}, \dots, \lambda_{B_{p_1}}$ abstractors, i.e., $j < k \leq p_1 + j$. In this case, for each e_i^1 such that there exists $m \geq 0$ where $\tau(e_i^1) = C_1 \rightarrow \dots \rightarrow C_m \rightarrow B_{p_1 + j + 1} \rightarrow \dots \rightarrow B_{p_1 + r'} \rightarrow C$, we add to Σ the substitution

$X/\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+j}} \cdot (\underline{p_1+j-i+1} \ Y_1(\underline{p_1+j} \dots \underline{1}) \dots (Y_m \ \underline{p_1+j} \dots \underline{1}))$, where Y_1, \dots, Y_m are fresh meta-variables with appropriate type. In this case, we have at most $p_1 \cdot r'$ possible distinct projections.

In $\beta\eta$ -unification, λ -terms of the same type always have the same number of external abstractors and the headings of the projections substitutions always have the form $\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \cdot \underline{i}$, where $r' < i \leq p_1 + r'$. This gives at most p_1 possible different projections, one for each argument of X . Analogously to the imitation case, all the projections generated in the $\beta\eta$ -calculus are more general than the corresponding projections in the β -calculus in the sense given by the following theorem.

Theorem 2. *Let P be a unification problem containing the flexible-rigid equation as in (1). If $\sigma_1, \dots, \sigma_k$ ($k \geq 0$) are the substitutions generated in one step of the projection rule onto the i -th argument of X ($1 \leq i \leq p_1$) in the β -calculus and γ the corresponding substitution generated in the $\beta\eta$ -calculus for the η -long form of the given equation, then there exist substitutions $\theta_1, \dots, \theta_k$ such that $\gamma\theta_i = \sigma_i$, for all $1 \leq i \leq k$.*

Proof. Let

$$\sigma_i = \{X/\lambda_{B_1} \dots \lambda_{B_j} \cdot (\underline{j-i+1} (X_1 \ \underline{j} \dots \underline{1}) \dots (X_m \ \underline{j} \dots \underline{1}))\}$$

be a substitution generated in the β -calculus. By the construction of the projection rule, there exists $m \geq 0$ such that $\tau(e_i^1) = C_1 \rightarrow \dots \rightarrow C_m \rightarrow B_{j+1} \rightarrow \dots \rightarrow B_{p_1} \rightarrow A_{r+1} \rightarrow A_{r+r'} \rightarrow C$. Without loss of generality, we may assume that in the $\beta\eta$ -calculus, the equation (1) assumes the form

$$\lambda_{A_1} \dots \lambda_{A_{r+r'}} \cdot (X \ e_1^1 \dots e_{p_1}^1 \ \underline{r'} \dots \underline{1}) \stackrel{?}{=} \lambda_{A_1} \dots \lambda_{A_{r+r'}} \cdot (\underline{n} \ e_1^2 \dots e_{p_2}^2).$$

The corresponding projection step, in the $\beta\eta$ -calculus, generates a substitution which has $p_1 + r'$ external abstractors (since terms are in η -long form) and whose number of arguments can be easily determined in terms of m :

$\gamma = \{X/\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \cdot \underline{p_1+r'-i+1} (Y_1 \ \underline{p_1+r'} \dots \underline{1}) \dots (Y_{m+r'+p_1-j} \ \underline{p_1+r'} \dots \underline{1})\}$ where $Y_1, \dots, Y_{m+r'+p_1-j}$ are meta-variables with appropriate type and sub-terms are in η -long form. Consider the substitution

$$\begin{aligned} \theta_j = & \{Y_1/\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \cdot (Z_1 \ \underline{p_1+r'} \dots \underline{p_1+r'-j+1}) \\ & \dots \\ & Y_m/\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \cdot (Z_m \ \underline{p_1+r'} \dots \underline{p_1+r'-j+1})\} \cup \\ & \{Y_{m+1}/\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \cdot \underline{p_1+r'-j}, \\ & Y_{m+2}/\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \cdot \underline{p_1+r'-j-1}, \\ & \dots \\ & Y_{m+r'+p_1-j}/\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \cdot \underline{1}\} \end{aligned}$$

The β -normal form of $\gamma\theta_j$ is

$$X/\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \cdot (\underline{p_1+r'-i+1} (Z_1 \ \underline{p_1+r'} \dots \underline{p_1+r'-j+1}) \dots (Z_m \ \underline{p_1+r'} \dots \underline{p_1+r'-j+1}) \ \underline{p_1+r'-j} \dots \underline{1})$$

And finally, η -reducing the above substitution, we have $\gamma\theta_j = \sigma_j$ up to renaming of meta-variables.

The analysis for projections with headings of the form

$$\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+j}} \cdot \underline{i}$$

is similar. □

In the following we give an algorithmic description of the procedure MATCH.

Procedure MATCH

INPUT: A flexible-rigid equation eq .

OUTPUT: A set Σ of incremental substitutions for the head of the flexible term.

1. Apply the imitation and/or the projection rules to eq non-deterministically and call Σ the set of generated substitutions.

3.3 The Main Procedure

The main procedure of Huet's algorithm non-deterministically and successively calls the procedures SIMPL and MATCH. These calls may generate equivalent unification problems or substitutions which can be organised into a unification tree (see Fig. 2).

Main Procedure

INPUT: A unification problem P_ϵ .

OUTPUT: A success status if the original problem is unifiable or a failure status if the original problem is not unifiable. The algorithm may not terminate in the latter case.

1. If $P_{i_1 \dots i_k}$ contains a rigid-rigid equation then apply SIMPL and go to the next step, else if it contains a flexible-rigid equation then rename $P_{i_1 \dots i_k}$ to $\overline{P}_{i_1 \dots i_k}$ and go to the next step, else go to step 4.
2. Let eq be a flexible-rigid equation in $\overline{P}_{i_1 \dots i_k}$. Apply MATCH to eq and call $\Sigma_{i_1 \dots i_k}$ the generated set of substitutions and go to step 3.
3. If $\Sigma_{i_1 \dots i_k}$ is the empty set then stop and report a failure status, else let $\Sigma_{i_1 \dots i_k} = \{\sigma_{i_1 \dots i_k 1}, \dots, \sigma_{i_1 \dots i_k r}\}$ where $r > 0$ and, for each substitution $\sigma_{i_1 \dots i_k j} \in \Sigma_{i_1 \dots i_k}$ call $P_{i_1 \dots i_k j} := P_{i_1 \dots i_k} \sigma_{i_1 \dots i_k j}$ the new unification problem and go to step 1.
4. Stop and report a success status. The corresponding solution assuming that the current node is at position $i_1 \dots i_k$ is given by the composition:

$$\sigma_{i_1} \sigma_{i_1 i_2} \dots \sigma_{i_1 i_2 \dots i_{k-1}} \sigma_{i_1 i_2 \dots i_k}$$

Example 1. Consider the unification problem $P = \{\lambda_A.X \ \underline{3} =? \ \lambda_A.\underline{2}(\underline{43})\}$ over the context $\Gamma = A \rightarrow B.A.A \rightarrow A.nil$ and $\Gamma \vdash X : A \rightarrow B$. Figures 3 and 4 show the matching tree for P considering the λ -calculus without respectively with η -conversion. The solutions are given by the compositions of the substitutions through a path whose leaf is a trivial problem (also called a success node). Note

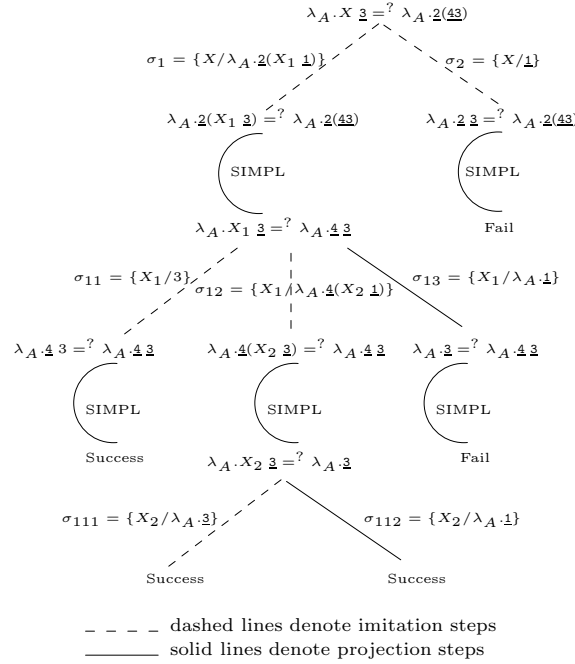


Fig. 3. Unification without η -conversion.

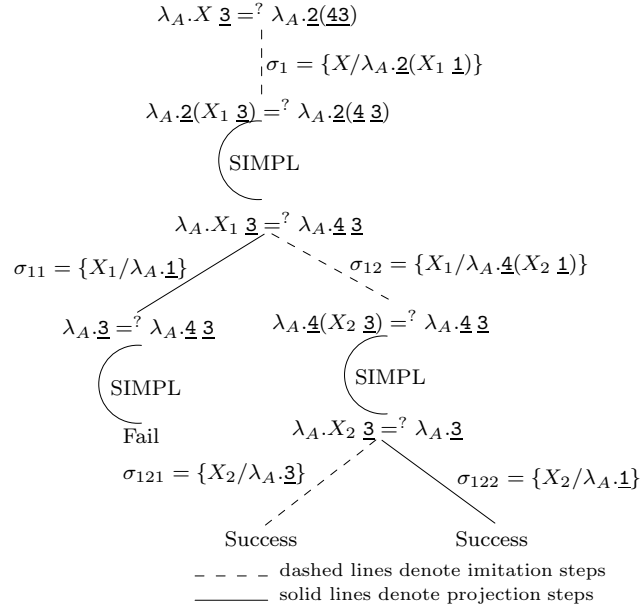


Fig. 4. Unification with η -conversion.

that, after composing, the terms need to be normalised. For instance, in Fig.4 one can compute the composition $\sigma_1\sigma_{12}\sigma_{122}$ by first applying the usual composition of substitutions:

$$\{X/\lambda_A.\underline{2}((\lambda_A.\underline{5}((\lambda_A.\underline{1}) \underline{1})) \underline{1}), X_1/\lambda_A.\underline{4}((\lambda_A.\underline{1}) \underline{1}), X_2/\lambda_A.\underline{1}\}$$

and then applying β -reduction:

$$\{X/\lambda_A.\underline{2}(\underline{41}), X_1/\lambda_A.(\underline{4} \underline{1}), X_2/\lambda_A.\underline{1}\}.$$

In the same way, one computes the substitution

$$\sigma_1\sigma_{12}\sigma_{121} = \{X/\lambda_A.\underline{2}(\underline{43}), X_1/\lambda_A.(\underline{4} \underline{3}), X_2/\lambda_A.\underline{3}\}.$$

The solutions to the original problem are given by the substitutions for the meta-variables that appear in it: $X/\lambda_A.\underline{2}(\underline{43})$ and $X/\lambda_A.\underline{2}(\underline{41})$.

4 Correspondence between $\lambda\sigma$ -HOU and Huet's HOU

In [DHK00], a generalisation of Huet's unification method, based on the $\lambda\sigma$ -style of explicit substitutions, is presented. Nevertheless, no formal comparison among these methods is provided. In this section, we compare these two methods proving that the SIMPL and MATCH procedures of Huet can be simulated in the $\lambda\sigma$ -calculus and that there exists a correspondence among the generated substitutions and graftings for unification problems which are in the image of the precooking. Although most of the modern computational systems based on the λ -calculus use η -conversion, we decided to analyse both β - and $\beta\eta$ -unification, as Huet did in his original paper [Hue75].

A unification problem in the $\lambda\sigma$ -calculus is written as a disjunction of existentially quantified conjunctions of the form $\bigvee_{j \in J} \exists \vec{w}_j \bigwedge_{i \in I_j} a_i =_{\lambda\sigma}^? b_i$ where a_i and b_i are $\lambda\sigma$ -terms of the same type. If $|J| = 1$ then the unification problem is called a *unification system*. Since we are interested in relating a unification algorithm in the pure λ -calculus and one in the $\lambda\sigma$ -calculus, it is important to know how to translate λ -unification problems into $\lambda\sigma$ -unification problems and vice-versa. The process of translating a unification problem P from the simply typed λ -calculus to the corresponding unification problem P_F in the typed $\lambda\sigma$ -calculus is done by the *precooking* translation defined as follows.

Definition 15 ([DHK00]). *Let $a \in \Lambda_{dB}(\mathcal{X})$ such that $\Gamma \vdash a : T$. To every meta-variable X of type U in the term a , we associate the type U and the context Γ in $\lambda\sigma$ -calculus. The precooking of a from $\Lambda_{dB}(\mathcal{X})$ to the set of $\lambda\sigma$ -terms, written $\Lambda_{\lambda\sigma}(\mathcal{X})$, is defined by $a_F = F(a, 0)$ where $F(a, n)$ is defined by:*

(a) $F((\lambda_B.a), n) = \lambda_B(F(a, n+1))$	(b) $F(\underline{k}, n) = \underline{1}[\uparrow^{k-1}]$
(c) $F((a b), n) = F(a, n) F(b, n)$	(d) $F(X, n) = X[\uparrow^n]$

Note that the precooking defined above is injective and hence its inverse is well defined. This remark will be important when unification solutions in the

language of the $\lambda\sigma$ -calculus need to be translated back to the language of the pure λ -calculus. In the next two subsections we present the $\lambda\sigma$ -version of the procedures **SIMPL** and **MATCH**, called **SIMPL** $_{\lambda\sigma}$ and **MATCH** $_{\lambda\sigma}$ which are built using adequate strategies over the inference rules of the $\lambda\sigma$ -HOU method. In this way we formalise the fact that HOU via explicit substitutions calculi is a generalisation of Huet's method.

The procedure **SIMPL $_{\lambda\sigma}$.** As in the pure λ -calculus, we give the same description for the procedure **SIMPL** $_{\lambda\sigma}$ for unification with and without **Eta**-conversion. The sole difference is that in the former case, the rule **Dec-App- λ** can be removed from the description because it never applies to **Eta**-long terms. The inference rules for unification in the $\lambda\sigma$ -calculus are given in Tables 3 and 4.

Dec-λ	$\frac{P \wedge \lambda_A.e_1 =_{\lambda\sigma}^? \lambda_A.e_2}{P \wedge e_1 =_{\lambda\sigma}^? e_2}$
Dec-App	$\frac{P \wedge (\underline{n} e_1^1 \dots e_p^1) =_{\lambda\sigma}^? (\underline{n} e_1^2 \dots e_p^2)}{P \wedge e_1^1 =_{\lambda\sigma}^? e_1^2 \wedge \dots \wedge e_p^1 =_{\lambda\sigma}^? e_p^2}$
Dec-Fail	$\frac{P \wedge (\underline{n} e_1^1 \dots e_{p_1}^1) =_{\lambda\sigma}^? (\underline{m} e_1^2 \dots e_{p_2}^2)}{Fail}, \text{ if } m \neq n.$
Dec-App-λ	$\frac{P \wedge \lambda_A.e =_{\lambda\sigma}^? (\underline{m} e_1^2 \dots e_{p_2}^2)}{Fail}$

Table 3. Unification Rules for the $\lambda\sigma$ -calculus (part I)

Procedure **SIMPL** $_{\lambda\sigma}$

INPUT: A unification problem P_q (in the language of the $\lambda\sigma$ -calculus) with at least one rigid-rigid equation.

OUTPUT: A terminal (failure or success) status or an equivalent unification problem \overline{P}_q without rigid-rigid equations and containing at least one flexible-rigid equation.

Assume that **Dec- λ** is applied eagerly.

WHILE there exists a rigid-rigid equation in P_q **DO**

1. Apply **Dec-App- λ** or **Dec-Fail**.
2. Apply **Dec-App**, and if the resulting unification problem contains a flexible-rigid equation, call it \overline{P}_q and give \overline{P}_q as result, else stop and report a success status.

DONE.

Exp	$\frac{P \wedge X[a_1 \dots a_p \cdot \uparrow^n] e_1 \dots e_l \stackrel{?}{=}_{\lambda\sigma} (\underline{m} \ b_1 \dots b_q)}{P \wedge X[a_1 \dots a_p \cdot \uparrow^n] e_1 \dots e_l \stackrel{?}{=}_{\lambda\sigma} (\underline{m} \ b_1 \dots b_q) \wedge \bigvee_{r \in R_p \cup R_i} \exists H_1 \dots \exists H_k : X \stackrel{?}{=}_{\lambda\sigma} (\underline{x} \ H_1 \dots H_k) \vee Q}$ <p>if X is not solved. where H_1, \dots, H_k are variables of the appropriate type not occurring in P with the contexts $\Gamma_{H_i} = \Gamma_X$, R_p is the subset of $\{1, \dots, p\}$ such that $(\underline{x} \ H_1 \dots H_k)$ has the right type, $R_i =$ if $m \geq n + 1$ then $\{m - n + p\}$ else \emptyset and $Q = \exists Y X \stackrel{?}{=}_{\lambda\sigma} \lambda.Y$ if X has a functional type and $Q = Fail$ otherwise.</p>
Exp2	$\frac{P \wedge X[a_1 \dots a_p \cdot \uparrow^n](e_1, \dots, e_l) \stackrel{?}{=}_{\lambda\sigma} \lambda b}{P \wedge X[a_1 \dots a_p \cdot \uparrow^n](e_1, \dots, e_l) \stackrel{?}{=}_{\lambda\sigma} \lambda b \wedge \bigvee_{r \in R_p} \exists H_1 \dots \exists H_k : X \stackrel{?}{=}_{\lambda\sigma} \underline{x}(H_1, \dots, H_k) \vee \exists Y X \stackrel{?}{=}_{\lambda\sigma} \lambda Y}$ <p>if X is not solved. where H_1, \dots, H_k are variables of the appropriate type not occurring in P with the contexts $\Gamma_{H_i} = \Gamma_X$, R_p is the subset of $\{1, \dots, p\}$ such that $\underline{x}(H_1, \dots, H_k)$ has the right type.</p>
Exp-λ	$\frac{P}{\exists Y : (A. \Gamma \vdash B), P \wedge X \stackrel{?}{=}_{\lambda\sigma} \lambda_A Y}$ <p>if $(X : \Gamma \vdash A \rightarrow B) \in TVar(P)$, $Y \notin TVar(P)$, and X is not a solved variable.</p>
Exp-App	$\frac{P \wedge X[a_1 \dots a_p \cdot \uparrow^n] \stackrel{?}{=}_{\lambda\sigma} (\underline{m} \ b_1 \dots b_q)}{P \wedge X[a_1 \dots a_p \cdot \uparrow^n] \stackrel{?}{=}_{\lambda\sigma} (\underline{m} \ b_1 \dots b_q) \wedge \bigvee_{r \in R_p \cup R_i} \exists H_1 \dots \exists H_k, X \stackrel{?}{=}_{\lambda\sigma} \underline{x}(H_1, \dots, H_k)}$ <p>if X has an atomic type and is not solved. where H_1, \dots, H_k are variables of appropriate types, not occurring in P, with the contexts $\Gamma_{H_i} = \Gamma_X$, R_p is the subset of $\{1, \dots, p\}$ such that $\underline{x}(H_1, \dots, H_k)$ has the right type, $R_i =$ if $m \geq n + 1$ then $\{m - n + p\}$ else \emptyset.</p>
Normalise1	$\frac{P \wedge e_1 \stackrel{?}{=}_{\lambda\sigma} e_2}{P \wedge e'_1 \stackrel{?}{=}_{\lambda\sigma} e'_2}$ <p>if e_1 or e_2 is not in long form. where e'_1 (resp. e'_2) is the long form of e_1 (resp. e_2) if e_1 (resp. e_2) is not a solved variable and e_1 (resp. e_2) otherwise.</p>
Normalise2	$\frac{P \wedge e_1 \stackrel{?}{=}_{\lambda\sigma} e_2}{P \wedge e'_1 \stackrel{?}{=}_{\lambda\sigma} e'_2}$ <p>if e_1 or e_2 is not in $\lambda\sigma$-normal form. where e'_1 (resp. e'_2) is the $\lambda\sigma$-normal form of e_1 (resp. e_2) if e_1 (resp. e_2) is not a solved variable and e_1 (resp. e_2) otherwise.</p>
Replace	$\frac{P \wedge X \stackrel{?}{=}_{\lambda\sigma} t}{\{X \mapsto t\}(P) \wedge X \stackrel{?}{=}_{\lambda\sigma} t}$ <p>if $X \in TVar(P)$, $X \notin TVar(t)$ and if t is a constant then $t \in TVar(P)$.</p>

Table 4. Unification Rules for the $\lambda\sigma$ -calculus (part II)

Proposition 2. *The application of the procedure $SIMPL_{\lambda\sigma}$ to any unification problem P (in the language of the $\lambda\sigma$ -calculus) always terminates.*

Proof. We need to prove that each rule used in the procedure $SIMPL_{\lambda\sigma}$ terminates. To do so, consider the function θ that measures the size of equations by $\theta(e_1 =_{\lambda\sigma}^? e_2) := |e_1| + |e_2|$ and $\theta(Fail) = 0$. Now consider the multi-set $\gamma(P)$ whose elements are $\theta(eq)$ for each equation $eq \in P$. Applications of the rule **Dec- λ** decrease the size of the terms in both sides of the selected equation and keep the other equations unchanged, hence $\gamma(P) \gg \gamma(\mathbf{Dec-}\lambda(P))$, where \gg is the usual order for multi-sets. The rules **Dec-App- λ** and **Dec-Fail** terminate since *Fail* is a terminal. The rule **Dec-App** generates a finite number of new equations, but each of them has size strictly less than the original equation and hence $\gamma(P) \gg \gamma(\mathbf{Dec-App}(P))$. Finally, each cycle in the **WHILE** starting from P generates a unification problem \bar{P} such that $\gamma(P) \gg \gamma(\bar{P})$ because at least one of the listed rules applies to P and there is no way to keep a rigid-rigid equation unchanged. After a finite number of steps, all the rigid-rigid equations will be eliminated and the procedure stops. The resulting unification problem is equivalent to the original one due to the correctness of the inference rules of Table 3 (cf. [DHK00]). \square

In order to establish a correspondence between the procedures $SIMPL$ and $SIMPL_{\lambda\sigma}$, it is worthwhile to notice that if P_F is the precooked image of the unification problem P then, by the definition of precooking, P_F does not contain any internal operator of the $\lambda\sigma$ -calculus other than the codification of de Bruijn indexes and updated meta-variables, i.e., $\underline{n} = \underline{1}[\uparrow^{n-1}]$ ($n > 0$) and $X[\uparrow^k]$ ($k \geq 0$). Note also that the headings of rigid terms are kept unchanged under the precooking translation, and hence, the precooked image of a rigid (resp. flexible) term is also a rigid (resp. flexible) term. In this direction we have the following results:

Lemma 1. *Let P be a unification problem in the typed $\lambda\sigma$ -calculus which is in the image of the precooking translation. Then applications of the rules **Dec- λ** , **Dec-App**, **Dec-Fail** and **Dec-App- λ** to P generate a new unification problem which is still in the image of the precooking translation.*

Proof. Straightforward from the definition of the rules. Note that no internal operator of the $\lambda\sigma$ -calculus is introduced by these rules. \square

Lemma 2. *Let P be a unification problem in the pure λ -calculus containing at least one rigid-rigid equation. If the procedure $SIMPL$ applied to P returns a failure status, then $SIMPL_{\lambda\sigma}$ applied to the precooked image of P stops and returns a failure status.*

Proof. If $SIMPL$ applied to P returns a failure status then there exists a rigid-rigid equation in P with different headings. Hence the precooked image of this equation will also fail after one application of **Dec-Fail** or **Dec-App- λ** . \square

Lemma 3. *Let P be a unification problem in the $\lambda\sigma$ -calculus which is in the image of the precooking translation and contains at least one rigid-rigid equation. If the procedure $SIMPL_{\lambda\sigma}$ applied to P stops and returns a failure status, then $SIMPL$ applied to the unification problem obtained by the inverse of the precooking applied to P stops and returns a failure status.*

Proof. If $SIMPL_{\lambda\sigma}$ applied to P stops and returns a failure status, then it comes from one application of the rules **Dec-Fail** or **Dec-App- λ** . Hence, there exists a rigid-rigid equation in P with different headings. As the inverse of the precooking does not modify headings of rigid terms and is well defined by Lemma 1, we conclude that the procedure $SIMPL$ will stop and return a failure status. \square

Lemma 4. *Let P be a unification problem in the pure λ -calculus containing at least one rigid-rigid equation. If the procedure $SIMPL$ applied to P returns a success status, then $SIMPL_{\lambda\sigma}$ applied to the precooked image of P returns a success status.*

Proof. If $SIMPL$ applied to P returns a success status then, after simplifying all rigid-rigid equations, the resulting unification problem is composed by a (possible empty) conjunction of flexible-flexible and trivial equations³. This means that P contains a finite number of rigid-rigid equations in which the heading of the left-hand side term is the same as that of the right-hand side term for each such equation and, the same condition holds to possible new rigid-rigid equations generated by the decomposition step. Since the precooking translation keeps unchanged the heading of a rigid term, we conclude that the rules **Dec-Fail** and **Dec-App- λ** do not apply neither to any of the rigid-rigid equations nor to the possible rigid-rigid equations generated by the rule **Dec-App**. Moreover, no flexible-rigid equation is generated by $SIMPL_{\lambda\sigma}$ otherwise it should also be generated by $SIMPL$. Therefore, $SIMPL_{\lambda\sigma}$ applied to P_F is a (possible empty) conjunction of flexible-flexible and trivial equations and hence $SIMPL_{\lambda\sigma}$ stops and returns a success status. \square

Lemma 5. *Let P be a unification problem in the $\lambda\sigma$ -calculus which is in the image of the precooking translation and which contains at least one rigid-rigid equation. If the procedure $SIMPL_{\lambda\sigma}$ applied to P stops and returns a success status, then $SIMPL$ applied to the unification problem obtained by the inverse of the precooking applied to P stops and returns a success status.*

Proof. If $SIMPL_{\lambda\sigma}$ reports a success status then, after simplifying all rigid-rigid equations using the rules **Dec- λ** and **Dec-App**, the resulting unification system is composed by a (possible empty) conjunction of flexible-flexible and trivial equations. This means that all possible rigid-rigid equations that were in P or that were generated by one application of **Dec-App** are such that the heading of the left-hand side term is equal to the heading of the right-hand side term. Since the inverse of the precooking keeps the heading of a rigid term and de Bruijn indexes unchanged and is well defined by Lemma 1, we can conclude that

³ Trivial equations can be eliminated automatically.

the headings of the rigid-rigid equations of the unification problem obtained by applying the inverse of the precooking to P are the same for the left-hand and right-hand side of each equation and hence the procedure SIMPL will return a success status. \square

Lemma 6. *Let P be a unification problem in the pure λ -calculus such that the result of applying SIMPL to P is the unification problem \overline{P} containing at least one flexible-rigid equation. Then the application of the procedure $\text{SIMPL}_{\lambda\sigma}$ to the precooked image P_F of P returns a unification problem containing at least one flexible-rigid equation.*

Proof. There are two possible origins for this flexible-rigid equation. Either it already exists in P or it was generated by the simplification of one rigid-rigid equation in P . In the former case, it will remain unchanged by the procedure SIMPL since only the rigid-rigid equations are modified. The procedure $\text{SIMPL}_{\lambda\sigma}$ also does not modify flexible-rigid equations, we can conclude that the precooked image of this flexible-rigid equation remains unchanged by application of $\text{SIMPL}_{\lambda\sigma}$. In the latter case, such equation is generated by the simplification process which creates a new equation from corresponding arguments, i.e., from an equation of the form $\lambda_{A_1} \dots \lambda_{A_r} . (\underline{n} e_1^1 \dots e_p^1) =? \lambda_{A_1} \dots \lambda_{A_r} . (\underline{n} e_1^2 \dots e_p^2)$ the simplification process generates the conjunction $\lambda_{A_1} \dots \lambda_{A_r} . e_1^1 =? \lambda_{A_1} \dots \lambda_{A_r} . e_1^2 \wedge \dots \wedge \lambda_{A_1} \dots \lambda_{A_r} . e_p^1 =? \lambda_{A_1} \dots \lambda_{A_r} . e_p^2$ which contains at least one flexible-rigid equation. Since the precooking translation keeps de Bruijn indexes unchanged, and only updates meta-variables, we can conclude that the conjunction $e_{1_F}^1 =? e_{1_F}^2 \wedge \dots \wedge e_{p_F}^1 =? e_{p_F}^2$ contains at least one flexible-rigid equation. \square

Lemma 7. *Let P be a unification problem in the $\lambda\sigma$ -calculus which is in the image of the precooking translation and such that an application of the procedure $\text{SIMPL}_{\lambda\sigma}$ to P returns a unification problem containing at least one flexible-rigid equation. Then the application of the procedure SIMPL to the unification problem obtained by the inverse of the precooking applied to P returns a simplified unification problem containing at least one flexible-rigid equation.*

Proof. Again, there are two possible origins for this flexible-rigid equation. Either it already exists in P or it was generated by one application of the rule **Dec-App**. In the former case, it will remain unchanged by the procedure $\text{SIMPL}_{\lambda\sigma}$ since it affects only rigid-rigid equations and the inverse of the precooking applied to P will also have a flexible-rigid equation which remains unchanged by application of the procedure SIMPL. In the latter case, such equation was generated from an equation of the form $\lambda_{A_1} \dots \lambda_{A_r} . (\underline{n} e_{1_F}^1 \dots e_{p_F}^1) =? \lambda_{A_1} \dots \lambda_{A_r} . (\underline{n} e_{1_F}^2 \dots e_{p_F}^2)$ which will be replaced by the conjunction $e_{1_F}^1 =? e_{1_F}^2 \wedge \dots \wedge e_{p_F}^1 =? e_{p_F}^2$. Hence at least one of these equations is flexible-rigid. Applying SIMPL to the inverse of the precooking applied to this equation (which is well defined by Lemma 1), that is, $\lambda_{A_1} \dots \lambda_{A_r} . (\underline{n} e_1^1 \dots e_p^1) =? \lambda_{A_1} \dots \lambda_{A_r} . (\underline{n} e_1^2 \dots e_p^2)$, allows us to conclude that one of the generated equations $\lambda_{A_1} \dots \lambda_{A_r} . e_1^1 =? \lambda_{A_1} \dots \lambda_{A_r} . e_1^2 \wedge \dots \wedge \lambda_{A_1} \dots \lambda_{A_r} . e_p^1 =? \lambda_{A_1} \dots \lambda_{A_r} . e_p^2$ is flexible-rigid. \square

The results of these lemmas can be jointed as:

Theorem 3 (Correspondence between SIMPL and SIMPL_{λσ}). *If P is a unification problem in the pure λ -calculus and P_F its precooked image, then:*

1. *SIMPL(P) fails \Leftrightarrow SIMPL_{λσ}(P_F) fails.*
2. *SIMPL(P) stops and reports a success status \Leftrightarrow SIMPL_{λσ}(P_F) stops and reports a success status;*
3. *SIMPL(P) returns a unification problem containing at least one flexible-rigid equation \Leftrightarrow SIMPL_{λσ}(P_F) returns a unification problem containing at least one flexible-rigid equation.*

Proof. Straightforward from lemmas 2-7. □

The procedure MATCH_{λσ}. In the unification method over the $\lambda\sigma$ -calculus [DHK00], the solutions are given in the form of solved equations which are defined as follows:

Definition 16 ([DHK00]). *A unification problem P is in $\lambda\sigma$ -solved form if all its meta-variables are of atomic type and it is a conjunction of nontrivial equations of the following forms:*

- **Solved:** $X =_{\lambda\sigma}^? a$ where the meta-variable X does not appear anywhere else in P and a is in **Eta**-long form⁴. Such an equation is said to be solved in P and the variable X is also said to be solved.
- **Flexible-flexible:** $X[a_1 \cdots a_p \uparrow^n] =_{\lambda\sigma}^? Y[b_1 \cdots b_q \uparrow^m]$, where $X[a_1 \cdots a_p \uparrow^n]$ and $Y[b_1 \cdots b_q \uparrow^m]$ are **Eta**-long terms and the equation is not solved.

In our description, as a consequence of the fact that the solutions in the $\lambda\sigma$ -calculus have the form of equations and hence do not belong to another syntactical structure as in the pure λ -calculus, the procedure MATCH_{λσ} takes a unification system as argument instead of just an equation as in Huet’s algorithm. We start by considering unification problems in the $\lambda\sigma$ -calculus with **Eta**-conversion. The following description is done according to the unification tree notation (see Fig. 5).

Procedure MATCH_{λσ} (with Eta)

INPUT: A unification system P_q with at least one flexible-rigid equation.

OUTPUT: A disjunction of equivalent unification systems, written $P_{q1} \vee \dots \vee P_{qk}$.

Assume that the rule **Dec-λ** is applied eagerly.

1. Apply **Exp-λ** and **Replace** as much as possible to the selected equation and call P'_q the resulting unification system.

⁴ In the $\lambda\sigma$ -calculus without **Eta**-conversion, consider the $\lambda\sigma$ -normal form instead of the **Eta**-long form.

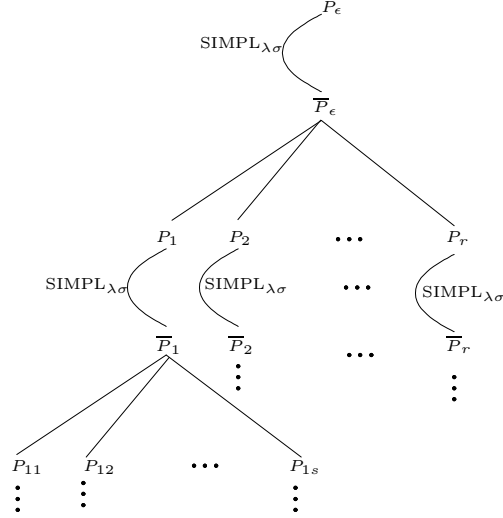


Fig. 5. The $\lambda\sigma$ -unification tree.

2. Apply **Exp-App** and **Replace** and **Normalise** to P'_q and call $P_{q1} \vee \dots \vee P_{qk}$ the resulting unification problem.

The following results allow us to establish a formal correspondence between the procedures **MATCH** and **MATCH** $_{\lambda\sigma}$ in the λ -calculus with η -conversion and in the $\lambda\sigma$ -calculus with **Eta**-conversion, respectively. In this comparison, we want to relate substitutions in the pure λ -calculus and the generated equations by the **Exp-App** rule in the $\lambda\sigma$ -calculus. This is formally done according to the following definition.

Definition 17. Let X/a be a substitution generated in the pure λ -calculus by Huet's algorithm. We say that the equation $Y =_{\xi}^? b$ corresponds (or is associated) to the substitution X/a if X and Y are two meta-variables of the same type and the terms a and b have the same headings, where $\xi \in \{\lambda\sigma, \lambda s_e\}$ ⁵.

Note that the substitutions generated in Huet's algorithm always replace meta-variables by rigid terms and hence the precooking does not affect the headings of these terms. Therefore we can say that the λ -term a has the same heading of the $\lambda\sigma$ -term b .

Lemma 8. Let $(X[a_1 \dots a_p \uparrow^n] e_1 \dots e_k) =_{\lambda\sigma}^? b$ be a flexible-rigid equation in the $\lambda\sigma$ -calculus such that the type of X is given by $A_1 \rightarrow \dots \rightarrow A_k \rightarrow B$ (with $k \geq 0$ and B atomic). Then the $\lambda\sigma$ -normal form of this equation, after some atomisation steps generated by successive applications of the strategy **Exp- λ** and

⁵ Note that the codification $\underline{1}[\uparrow^{n-1}]$ and the de Bruijn index \underline{n} are considered the same in this definition.

Replace, produces a conjunction of the form $X_k[e_k \cdots e_1.a_1.a_2 \cdots a_p.\uparrow^n] =_{\lambda\sigma}^? b_k \wedge X =_{\lambda\sigma}^? \lambda_{A_1} \cdots \lambda_{A_k}.X_k \wedge Q$, after a normalisation step, where X_k is a new meta-variable with atomic type B , b_k is a rigid term in $\lambda\sigma$ -normal form and Q is a conjunction of equations.

Proof. The proof is straightforward using the given strategy. Nevertheless, we detail the initial steps:

$$\begin{aligned}
& X[a_1.a_2 \cdots a_p.\uparrow^n] e_1 \dots e_k =_{\lambda\sigma}^? b \\
& \rightarrow \mathbf{Exp-\lambda} \\
& X[a_1.a_2 \cdots a_p.\uparrow^n] e_1 \dots e_k =_{\lambda\sigma}^? b \wedge X =_{\lambda\sigma}^? \lambda_{A_1}.X_1 \\
& \rightarrow \mathbf{Replace} \\
& (\lambda_{A_1}.X_1)[a_1.a_2 \cdots a_p.\uparrow^n] e_1 \dots e_k =_{\lambda\sigma}^? b_1 \wedge X =_{\lambda\sigma}^? \lambda_{A_1}.X_1 \\
& \rightarrow \mathbf{Exp-\lambda} \\
& \lambda_{A_1}.X_1[e_1.a_1.a_2 \cdots a_p.\uparrow^n] e_2 \dots e_k =_{\lambda\sigma}^? b_1 \wedge X =_{\lambda\sigma}^? \lambda_{A_1}.X_1 \wedge Q \\
& \rightarrow \mathbf{Replace} \\
& \lambda_{A_1}\lambda_{A_2}.X_2[e_1.a_1.a_2 \cdots a_p.\uparrow^n] e_2 \dots e_k =_{\lambda\sigma}^? b_2 \wedge X =_{\lambda\sigma}^? \lambda_{A_1}\lambda_{A_2}.X_2 \wedge Q \\
& \rightarrow \mathbf{Exp-\lambda} \\
& \vdots \\
& \rightarrow \mathbf{Normalise} \\
& X_k[e_k \cdots e_1.a_1.a_2 \cdots a_p.\uparrow^n] =_{\lambda\sigma}^? b_k \wedge X =_{\lambda\sigma}^? \lambda_{A_1} \cdots \lambda_{A_p}.X_k \wedge Q
\end{aligned}$$

where X_i are new meta-variables of type $A_{i+1} \rightarrow \dots \rightarrow A_k \rightarrow B$, for $i = 1, \dots, k-1$ and X_k has the atomic type B . \square

Note that, the number of arguments of X in the previous lemma coincides with the number of arguments of its type. This is not a severe restriction because it always happens for equations in **Eta**-long form.

Proposition 3 (Correspondence from MATCH to MATCH $_{\lambda\sigma}$). *Let $\lambda_{A_1} \cdots \lambda_{A_r}.(X e_1^1 \cdots e_{p_1}^1) =_{\lambda\sigma}^? \lambda_{A_1} \cdots \lambda_{A_r}.(\underline{n} e_1^2 \cdots e_{p_2}^2)$ be a flexible-rigid equation in η -long form in the pure λ -calculus where $p_1, p_2, r \geq 0$ and $\tau(X) = B_1 \rightarrow \dots \rightarrow B_{p_1} \rightarrow B$ with B atomic. Then, for each substitution generated by the procedure MATCH, when applied to this equation, there exists a corresponding equation⁶ in the $\lambda\sigma$ -calculus generated by the procedure MATCH $_{\lambda\sigma}$ to the precooked version of the given equation.*

Proof. By the description of the procedure MATCH done in section 3, in this case there exists exactly one possible imitation, if \underline{n} is a constant ($n > r$), whose substitution is given by:

$$X/\lambda_{B_1} \cdots \lambda_{B_{p_1}}.(\underline{p_1 + n - r} (Z_1 \underline{p_1} \cdots \underline{1}) \cdots (Z_{p_2} \underline{p_1} \cdots \underline{1})).$$

To generate the corresponding equation in the $\lambda\sigma$ -HOU algorithm consider the precooked version of the original equation: $\lambda_{A_1} \cdots \lambda_{A_r}.(X[\uparrow^r] e_{1_F}^1 \cdots e_{p_{1_F}}^1) =_{\lambda\sigma}^? \lambda_{A_1} \cdots \lambda_{A_r}.(\underline{n} e_{1_F}^2 \cdots e_{p_{2_F}}^2)$ and then, applying the procedure MATCH $_{\lambda\sigma}$ to it we have:

⁶ According to Definition 17.

1. Applying **Dec-λ**: $X[\uparrow^r] e_{1_F}^1 \dots e_{p_{1_F}}^1 =_{\lambda\sigma}^? \underline{n} e_{1_F}^2 \dots e_{p_{2_F}}^2$
2. Applying the strategy given in Lemma 8:
 $X_{p_1}[e_{p_{1_F}}^1 \dots e_{1_F}^1 \cdot \uparrow^r] =_{\lambda\sigma}^? \underline{n} e_{1_F}^2 \dots e_{p_{2_F}}^2 \wedge X =_{\lambda\sigma}^? \lambda_{B_1} \dots \lambda_{B_{p_1}} \cdot X_{p_1} \wedge Q$
3. Finally, applying **Exp-App**: $X_{p_1}[e_{p_{1_F}}^1 \dots e_{1_F}^1 \cdot \uparrow^r] =_{\lambda\sigma}^? \underline{n} e_{1_F}^2 \dots e_{p_{2_F}}^2 \wedge X =_{\lambda\sigma}^? \lambda_{B_1} \dots \lambda_{B_{p_1}} \cdot X_{p_1} \wedge Q \wedge X_{p_1} =_{\lambda\sigma}^? \underline{p_1 + n - r} H_1 \dots H_{p_2}$.

An application of **Replace** gives the solved equation

$$X =_{\lambda\sigma}^? \lambda_{B_1} \dots \lambda_{B_{p_1}} \cdot (\underline{p_1 + n - r} H_1 \dots H_{p_2})$$

which corresponds to the given substitution.

If \underline{n} is a bound variable, i.e., $1 \leq n \leq r$, then suppose that the following projection is generated by Huet's algorithm:

$$X/\lambda_{B_1} \dots \lambda_{B_{p_1}} \cdot (\underline{p_1 - i + 1} (Z_1 \underline{p_1} \dots \underline{1}) \dots (Z_s \underline{p_1} \dots \underline{1})).$$

Note that this projection is generated only if the target type of the i -th argument of X is B , i.e., e_i^1 must have the same target type of X . To generate this substitution in the $\lambda\sigma$ -HOU algorithm consider again the unification system

$$X_{p_1}[e_{p_{1_F}}^1 \dots e_{1_F}^1 \cdot \uparrow^r] =_{\lambda\sigma}^? \underline{n} e_{1_F}^2 \dots e_{p_{2_F}}^2 \wedge X =_{\lambda\sigma}^? \lambda_{B_1} \dots \lambda_{B_{p_1}} \cdot X_{p_1} \wedge Q.$$

As the precooking translation is a type preserving function, we can conclude that the target type of $e_{i_F}^1$ is also B and hence the equation

$$X_{p_1} =_{\lambda\sigma}^? \underline{p_1 - i + 1} H_1 \dots H_s$$

is generated by the rule **Exp-App**. After an application of **Replace**, we get the equation $X =_{\lambda\sigma}^? \lambda_{B_1} \dots \lambda_{B_{p_1}} \cdot (\underline{p_1 - i + 1} H_1 \dots H_s)$ which corresponds to the given projection substitution. \square

Proposition 4 (Correspondence from $\text{MATCH}_{\lambda\sigma}$ to MATCH). *For each new generated equation by the rule **Exp-App**, when applied to a flexible-rigid equation which is in the image of the precooking translation, there exists a corresponding substitution in the pure λ -calculus in the following sense: for each element in R_p there exists a corresponding substitution in the pure λ -calculus and, if $R_i \neq \emptyset$ then there exists an imitation in the pure λ -calculus for the inverse of the precooking translation applied to this equation.*

Proof. Without loss of generality, we may assume that the considered flexible-rigid equation has the form $\lambda_{A_1} \dots \lambda_{A_r} \cdot (X[\uparrow^r] e_{1_F}^1 \dots e_{p_{1_F}}^1) =_{\lambda\sigma}^? \lambda_{A_1} \dots \lambda_{A_r} \cdot b_F$, where $p_1, r \geq 0$ and $\tau(X) = B_1 \rightarrow \dots \rightarrow B_{p_1} \rightarrow B$ with B atomic. After applying the strategy sketched in Lemma 8, we get $X_{p_1}[e_{p_{1_F}}^1 \dots e_{1_F}^1 \cdot \uparrow^r] =_{\lambda\sigma}^? b_F \wedge X =_{\lambda\sigma}^? \lambda_{B_1} \dots \lambda_{B_{p_1}} \cdot X_{p_1} \wedge Q$, where X_{p_1} has the atomic type B and Q is a conjunction of equations. Now, we are ready to apply the **Exp-App** rule. Suppose that $R_p \neq \emptyset$, that is, there exists $\underline{i} \in R_p \subseteq \{\underline{1}, \dots, \underline{p}\}$ and the equation $X_{p_1} =_{\lambda\sigma}^? \underline{i} H_1 \dots H_s$ is generated. This means that the i -th element of the list $e_{p_{1_F}}^1 \dots e_{1_F}^1$ has target type B , i.e., $\tau(e_{p_1-i+1}^1) = C_1 \rightarrow \dots \rightarrow C_s \rightarrow B$ ($s \geq 0$).

But this is exactly the required condition by Huet's algorithm to generate the substitution

$$X/\lambda_{B_1} \dots \lambda_{B_{p_1}}.(\underline{\mathbf{1}}(Z_1 \underline{\mathbf{p}}_1 \dots \underline{\mathbf{1}}) \dots (Z_s \underline{\mathbf{p}}_1 \dots \underline{\mathbf{1}}))$$

which corresponds to the above equation generated in the $\lambda\sigma$ -HOU algorithm.

To analyse the case $R_i \neq \emptyset$, suppose without loss of generality, that b_F is of the form $(\underline{\mathbf{n}} e_1^2 \dots e_{p_2}^2)$, where $\tau(\underline{\mathbf{n}}) = C_1 \rightarrow \dots \rightarrow C_{p_2} \rightarrow B$. If $n > r$ then the rule **Exp-App** will generate the equation $X_{p_1} =_{\lambda\sigma}^? \underline{\mathbf{p}}_1 + \underline{\mathbf{n}} - \underline{\mathbf{r}} H_1 \dots H_s$. The condition $n > r$ means that $\underline{\mathbf{n}}$ is a constant and hence, Huet's algorithm will generate the corresponding substitution

$$X/\lambda_{B_1} \dots \lambda_{B_{p_1}}.(\underline{\mathbf{p}}_1 + \underline{\mathbf{n}} - \underline{\mathbf{r}}(Z_1 \underline{\mathbf{p}}_1 \dots \underline{\mathbf{1}}) \dots (Z_{p_2} \underline{\mathbf{p}}_1 \dots \underline{\mathbf{1}})). \quad \square$$

Propositions 3 and 4 justify the name $\text{MATCH}_{\lambda\sigma}$ because our procedure simulates in a faithful way the procedure **MATCH** for unification problems which are in the image of the pre-cooking translation. Now, the following theorem holds:

Theorem 4 (Correspondence between **MATCH and $\text{MATCH}_{\lambda\sigma}$).** *Let eq be a flexible-rigid equation in η -long form in the pure λ -calculus and eq_F its pre-cooked image. Then, **MATCH** applied to eq generates a substitution σ if and only if $\text{MATCH}_{\lambda\sigma}$ applied to eq_F generates a substitution equivalent to σ .*

Proof. This is a corollary of Propositions 3 and 4. \square

Now we make a similar analysis considering the λ -calculus without the η -conversion and the $\lambda\sigma$ -calculus without the **Eta**-conversion. In the following we give the algorithmic description of the procedure $\text{MATCH}_{\lambda\sigma}$ without **Eta**-conversion.

Procedure $\text{MATCH}_{\lambda\sigma}$ (without **Eta**)

INPUT: A unification system P_q with at least one flexible-rigid equation.

OUTPUT: A disjunction of equivalent unification systems, written $P_{q_1} \vee \dots \vee P_{q_k}$.

Assume that the rule **Dec- λ** is applied eagerly.

1. Apply **Exp2** and **Replace** as much as possible to the selected equation and then **Normalise**, if possible.
2. Apply **Exp** and **Replace** as much as possible to the selected equation and then **Normalise**, if possible.
3. Call $P_{q_1} \vee \dots \vee P_{q_k}$ the generated unification problem.

The following results allow us to formalise the relation between the procedure **MATCH** in the λ -calculus without η -conversion and the procedure $\text{MATCH}_{\lambda\sigma}$ in the $\lambda\sigma$ -calculus without **Eta**-conversion.

Lemma 9. *Let $\lambda_{A_1} \dots \lambda_{A_r}.(X[a_1 \dots a_p. \uparrow^n e_1^1 \dots e_{p_1}^1]) =_{\lambda\sigma}^? \lambda_{A_1} \dots \lambda_{A_{r+r'}}.b$ with $p_1, p, n, r, r' \geq 0$, be a flexible-rigid equation in the $\lambda\sigma$ -calculus without **Eta**-conversion where $\tau(X) = B_1 \rightarrow \dots \rightarrow B_{p_1} \rightarrow A_{r+1} \rightarrow \dots \rightarrow A_{r+r'} \rightarrow C$. After an application of the procedure $\text{MATCH}_{\lambda\sigma}$ we get an equivalent unification*

system of the form:

$X_{p_1+r'}[\underline{1}, \dots, \underline{r}].e_{p_1}^1[\uparrow^{r'}].\dots.e_1^1[\uparrow^{r'}].a_1[\uparrow^{r'}].a_2[\uparrow^{r'}].\dots.a_p[\uparrow^{r'}].\uparrow^{n+r'} =_{\lambda\sigma}^? b_{p_1+r'} \wedge X =_{\lambda\sigma}^? \lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}}.X_{p_1+r'} \wedge Q$ where $X_{p_1+r'}$ is a new meta-variable of type C , $b_{p_1+r'}$ is a $\lambda\sigma$ -normalised rigid term and Q is a conjunction of equations.

Proof. Straightforward considering the strategy given in the above description. This proof is analogous to the one sketched for Lemma 8. \square

Proposition 5 (Correspondence from MATCH to MATCH $_{\lambda\sigma}$ (without η)). *Let*

$$\lambda_{A_1} \dots \lambda_{A_r}.(X e_1^1 \dots e_{p_1}^1) =^? \lambda_{A_1} \dots \lambda_{A_{r+r'}}.(\underline{m} e_1^2 \dots e_{p_2}^2)$$

with $p_1, p_2, r, r' \geq 0$ and $m > 0$, be a flexible-rigid equation in the simply typed λ -calculus. For each substitution generated in the λ -calculus without η -conversion there exists a corresponding equation generated by the procedure MATCH $_{\lambda\sigma}$ in the $\lambda\sigma$ -calculus without **Eta**-conversion.

Proof. The analysis is divided into the imitation and projection cases.

IMITATION: The generated imitation substitutions are according to the following cases:

$r' > 0$: In this case, the head \underline{m} of the rigid term can be either a constant ($m > r + r'$) or a bound variable with $1 \leq m \leq r'$. In the former case, the generated substitution is given by

$$X/\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}}.(\underline{p_1 + m - r} (X_1 \underline{p_1 + r'} \dots \underline{1}) \dots (X_{p_2} \underline{p_1 + r'} \dots \underline{1}))$$

and, if $1 \leq m \leq r'$, the generated substitution is given by

$$X/\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}}.(\underline{m} (X_1 \underline{p_1 + r'} \dots \underline{1}) \dots (X_{p_2} \underline{p_1 + r'} \dots \underline{1})).$$

To generate the equation corresponding to the former substitution in the $\lambda\sigma$ -HOU algorithm, i.e., when \underline{m} is a constant, consider the precooked version of the original equation:

$$\lambda_{A_1} \dots \lambda_{A_r}.(X[\uparrow^r] e_{1_F}^1 \dots e_{p_1_F}^1) =_{\lambda\sigma}^? \lambda_{A_1} \dots \lambda_{A_{r+r'}}.(\underline{m} e_{1_F}^2 \dots e_{p_2_F}^2)$$

and then apply the procedure MATCH $_{\lambda\sigma}$ to it. According to Lemma 9, there exists an strategy such that the resulting unification system has the form:

$$X_{p_1+r'}[\underline{1}, \dots, \underline{r}].e_{p_1_F}^1[\uparrow^{r'}].\dots.e_{1_F}^1[\uparrow^{r'}].\uparrow^{r+r'} =_{\lambda\sigma}^? (\underline{m} e_{1_F}^2 \dots e_{p_2_F}^2) \wedge X =_{\lambda\sigma}^? \lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}}.X_{p_1+r'} \wedge Q$$

where $X_{p_1+r'}$ is a new meta-variable of type C . Note that the terms $e_{1_F}^2, \dots, e_{p_2_F}^2$ may change if they have an occurrence of X inside; nevertheless, we leave them with the same name because we are interested only in the external structure of

the term. Since $m > r + r'$, the **Exp** rule generates the equation $X_{p_1+r'} =_{\lambda\sigma}^? (\underline{p_1 + m - r} Y_1 \dots Y_{p_2})$ and, after an application of **Replace** and **Normalise**, we get an unification system containing the equation

$$X =_{\lambda\sigma}^? \lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} (\underline{p_1 + m - r} Y_1 \dots Y_{p_2})$$

which corresponds to the given imitation substitution.

If the other substitution is generated, i.e., if \underline{m} is a bound variable with $1 \leq m \leq r'$, then we can follow the strategy given by Lemma 9 and get the unification system

$$\begin{aligned} X_{p_1+r'} [\underline{1} \dots \underline{r'} \cdot e_{p_{1F}}^1 [\uparrow^{r'}] \dots e_{p_{1F}}^1 [\uparrow^{r'}] \cdot \uparrow^{r+r'}] &=_{\lambda\sigma}^? (\underline{m} e_{1F}^2 \dots e_{p_{2F}}^2) \wedge \\ X &=_{\lambda\sigma}^? \lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} X_{p_1+r'} \wedge Q. \end{aligned}$$

Since the precooking is a type preserving function, we have that the m -th element of the list $\underline{1} \dots \underline{r'} \cdot e_{p_{1F}}^1 [\uparrow^{r'}] \dots e_{p_{1F}}^1 [\uparrow^{r'}] \cdot \uparrow^{r+r'}$ has the same type of \underline{m} and the equation $X_{p_1+r'} =_{\lambda\sigma}^? (\underline{m} Y_1 \dots Y_{p_2})$ is generated, which corresponds to the given substitution. Note that, due to type conditions $\underline{m} \in R_p \cap \{1, \dots, r'\}$. At this point we have an interesting detail. There are some imitation substitutions which are simulated in the $\lambda\sigma$ -HOU algorithm by elements in R_p which are supposed to be projections.

$r' = 0$: In this case, we have that $m > r$, that is, the head of the rigid term is a constant, and for each $0 \leq j \leq p_1$ such that $p_2 - p_1 + j \geq 0$ and $\tau(e_i^1) = \tau(e_{p_2-p_1+i}^2)$, for all $j < i \leq p_1$, Huet's algorithm generates the substitution:

$$X / \lambda_{B_1} \dots \lambda_{B_j} (\underline{j + m - r} (X_1 \underline{j} \dots \underline{1}) \dots (X_{p_2-p_1+j} \underline{j} \dots \underline{1}))$$

which gives at most $\min(p_1, p_2) + 1$ possible solutions. To generate the corresponding substitution in the $\lambda\sigma$ -HOU algorithm, consider the precooked version of the original equation:

$$\lambda_{A_1} \dots \lambda_{A_r} (X[\uparrow^r] e_{1F}^1 \dots e_{p_{1F}}^1) =_{\lambda\sigma}^? \lambda_{A_1} \dots \lambda_{A_r} (\underline{m} e_{1F}^2 \dots e_{p_{2F}}^2).$$

The application of the procedure $\text{MATCH}_{\lambda\sigma}$ to this equation can be described according to the following steps:

- **Dec- λ** : $X[\uparrow^r] e_{1F}^1 \dots e_{p_{1F}}^1 =_{\lambda\sigma}^? \underline{m} e_{1F}^2 \dots e_{p_{2F}}^2$.
- **Exp**: $(X[\uparrow^r] e_{1F}^1 \dots e_{p_{1F}}^1 =_{\lambda\sigma}^? \underline{m} e_{1F}^2 \dots e_{p_{2F}}^2) \wedge (X =_{\lambda\sigma}^? \underline{m - r} Y_1 \dots Y_{p_2-p_1} \vee X =_{\lambda\sigma}^? \lambda_{B_1} X_1)$, and the equation $X =_{\lambda\sigma}^? \underline{m - r} Y_1 \dots Y_{p_2-p_1}$ corresponds to the case $j = 0$.

The other equation, i.e., $X =_{\lambda\sigma}^? \lambda_{B_1} X_1$, is an atomisation step and, after an application of **Replace** and **Normalise** we get the following unification system: $X_1 [e_{1F}^1 \cdot \uparrow^r] e_{2F}^2 \dots e_{p_{1F}}^1 =_{\lambda\sigma}^? \underline{m} e_{1F}^2 \dots e_{p_{2F}}^2 \wedge X =_{\lambda\sigma}^? \lambda_{B_1} X_1$. The imitation case for $j = 1$ is given by the solution in R_i which corresponds to the equation $X_1 =_{\lambda\sigma}^? \underline{m - r + 1} Z_1 \dots Z_{p_2-p_1+1}$. After an application of **Replace** we get the desired corresponding equation $X =_{\lambda\sigma}^? \lambda_{B_1} \underline{m - r + 1} Z_1 \dots Z_{p_2-p_1+1}$. Repeating this strategy, we will have at most $\min(p_1, p_2) + 1$ as in Huet's algorithm.

PROJECTION: Suppose that Huet's algorithm generates a projection substitution for the flexible-rigid equation:

$$\lambda_{A_1} \dots \lambda_{A_r} \cdot (X \ e_1^1 \dots e_{p_1}^1) =^? \lambda_{A_1} \dots \lambda_{A_{r+r'}} \cdot (\underline{m} \ e_1^2 \dots e_{p_2}^2).$$

The heading of this substitution, according to the description given in section 3.2, is either of the form

$$\lambda_{B_1} \dots \lambda_{B_j} \cdot \underline{i} \text{ with } 1 \leq j \leq p_1$$

or

$$\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+j}} \cdot \underline{i} \text{ with } 1 \leq j \leq r'$$

where \underline{i} is an appropriate de Bruijn index. In the former case we have that $X/\lambda_{B_1} \dots \lambda_{B_j} \cdot (\underline{j-i+1}(Z_1 \ \underline{j} \dots \underline{1}) \dots (Z_s \ \underline{j} \dots \underline{1}))$, and there exists $s \geq 0$ such that $\tau(e_i^1) = C_1 \rightarrow \dots \rightarrow C_s \rightarrow B_{j+1} \rightarrow \dots \rightarrow B_{p_1} \rightarrow A_{r+1} \rightarrow \dots \rightarrow A_{r+r'} \rightarrow C$. To simulate such a projection in the $\lambda\sigma$ -HOU algorithm consider the strategy given by the procedure $\text{MATCH}_{\lambda\sigma}$ until we get the unification system:

$X_j[e_{j_F}^1 \dots e_{1_F}^1 \cdot \uparrow^r] e_{j+1_F}^1 \dots e_{p_1_F}^1 =^?_{\lambda\sigma} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \cdot (\underline{m} \ e_{1_F}^2 \dots e_{p_2_F}^2) \wedge X =^?_{\lambda\sigma} \lambda_{B_1} \dots \lambda_{B_j} \cdot X_j \wedge Q$ where Q is a conjunction of equations. Applying the rule **Exp2** we get $X_j =^?_{\lambda\sigma} (\underline{j-i+1} \ H_1 \dots H_s)$ since the types are preserved by the precooking and, after an application of **Replace**, we get the corresponding equation.

When the heading has the form $\lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+j}} \cdot \underline{i}$, the analysis is similar. \square

Proposition 6 (Correspondence from $\text{MATCH}_{\lambda\sigma}$ to MATCH (without η)). *For each index in $R_p \cup R_i$ generated by the rules **Exp** or **Exp2** to a flexible-rigid equation which is in the image of the precooking translation according to the strategy given by the procedure $\text{MATCH}_{\lambda\sigma}$, there exists a corresponding substitution in the pure λ -calculus for the image of this equation by the inverse of the precooking translation.*

Proof. Without loss of generality we may assume that the flexible-rigid equation under consideration is of the form: $\lambda_{A_1} \dots \lambda_{A_r} \cdot (X[\uparrow^r] \ e_{1_F}^1 \dots e_{p_1_F}^1) =^?_{\lambda\sigma} \lambda_{A_1} \dots \lambda_{A_{r+r'}} \cdot (\underline{m} \ e_{1_F}^2 \dots e_{p_2_F}^2)$ where $p_1, p_2, r, r' \geq 0$ and $m > 0$, $\tau(X) = B_1 \rightarrow \dots \rightarrow B_{p_1} \rightarrow A_{r+1} \rightarrow A_{r+r'} \rightarrow C$. An application of the procedure $\text{MATCH}_{\lambda\sigma}$ to this equation can be described as follows considering two cases:

– $r' > 0$: The initial steps are given by:

1. **Dec- λ :** $X[\uparrow^r] \ e_{1_F}^1 \dots e_{p_1_F}^1 =^?_{\lambda\sigma} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \cdot (\underline{m} \ e_{1_F}^2 \dots e_{p_2_F}^2).$
2. **Exp2:** $X[\uparrow^r] \ e_{1_F}^1 \dots e_{p_1_F}^1 =^?_{\lambda\sigma} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \cdot (\underline{m} \ e_{1_F}^2 \dots e_{p_2_F}^2) \wedge X =^?_{\lambda\sigma} \lambda_{B_1} \cdot X_1.$
3. **Replace and Normalise:**
 $X_1[e_{1_F}^1 \cdot \uparrow^r] \ e_{2_F}^1 \dots e_{p_1_F}^1 =^?_{\lambda\sigma} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \cdot (\underline{m} \ e_{1_F}^2 \dots e_{p_2_F}^2) \wedge X =^?_{\lambda\sigma} \lambda_{B_1} \cdot X_1.$

At this point **Exp2** may generate either another atomisation step, i.e., $X_1 = \overset{?}{\lambda_\sigma} \lambda_{B_2}.X_2$, or an equation of the form $X_1 = \overset{?}{\lambda_\sigma} (\underline{1} H_1 \dots H_s)$ where $1 \in R_p$. The latter equation will not fail after an application of **Replace** and **Normalise** if the generated unification system, which is given by:

$$e_1^1 H_1 [e_{1_F}^1 \cdot \uparrow^r] \dots H_s [e_{1_F}^1 \cdot \uparrow^r] e_{2_F}^1 \dots e_{p_1_F}^1 = \overset{?}{\lambda_\sigma} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} (\underline{m} e_{1_F}^2 \dots e_{p_2_F}^2) \wedge X = \overset{?}{\lambda_\sigma} \lambda_{B_1}.X_1 \text{ is such that the heading of } e_1^1 \text{ is equal to } \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} \underline{m},$$

if e_1^1 is a rigid term, or $\lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}}.Y$, otherwise. In both cases, there exists an $s \geq 0$ where $\tau(e_1^1) = C_1 \rightarrow \dots \rightarrow C_s \rightarrow B_2 \rightarrow \dots \rightarrow B_{p_1} \rightarrow A_{r+1} \rightarrow \dots \rightarrow A_{r+r'} \rightarrow C$. But this is exactly the required condition by Huet's algorithm to generate the projection $X/\lambda_{B_1}(\underline{1}(Z_1 \underline{1}) \dots (Z_s \underline{1}))$. Now take the atomisation step $X_1 = \overset{?}{\lambda_\sigma} \lambda_{B_2}.X_2$. Applying the above strategy we get: $X_2 [e_{1_F}^1 \cdot \uparrow^r] e_{3_F}^1 \dots e_{p_1_F}^1 = \overset{?}{\lambda_\sigma} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} (\underline{m} e_{1_F}^2 \dots e_{p_2_F}^2) \wedge$

$X = \overset{?}{\lambda_\sigma} \lambda_{B_1} \lambda_{B_2}.X_2 \wedge X_1 = \overset{?}{\lambda_\sigma} \lambda_{B_2}.X_2$. Repeating the same analysis allows us to conclude that an application of the rule **Exp2** will generate an element in R_p only if there exists $s \geq 0$ such that $\tau(e_i^1) = C_1 \rightarrow \dots \rightarrow C_s \rightarrow B_3 \rightarrow \dots \rightarrow B_{p_1} \rightarrow A_{r+1} \rightarrow \dots \rightarrow A_{r+r'} \rightarrow C$ for $i = 1, 2$, and in this case Huet's algorithm will also generate the projection substitution $X/\lambda_{B_1} \lambda_{B_2} (\underline{3} - \underline{1}(Z_1 \underline{2} \underline{1}) \dots (Z_s \underline{2} \underline{1}))$ since the type conditions are preserved by the back translation of the precooking.

In general, using the above strategy, we get a unification system of the form: $X_j [e_{j_F}^1 \cdot \dots \cdot e_{1_F}^1 \cdot \uparrow^r] e_{j+1_F}^1 \dots e_{p_1_F}^1 = \overset{?}{\lambda_\sigma} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} (\underline{m} e_{1_F}^2 \dots e_{p_2_F}^2) \wedge X = \overset{?}{\lambda_\sigma} \lambda_{B_1} \dots \lambda_{B_j}.X_j \wedge Q$ where $0 \leq j \leq p_1$ and Q is a conjunction of equations. The rule **Exp2** will generate an element in $\underline{i} \in R_p \subseteq \{\underline{1}, \dots, \underline{j}\}$ only if there exists $s \geq 0$ such that $\tau(e_i^1) = C_1 \rightarrow \dots \rightarrow C_s \rightarrow B_{j+1} \rightarrow \dots \rightarrow B_{p_1} \rightarrow A_{r+1} \rightarrow \dots \rightarrow A_{r+r'} \rightarrow C$ which is the necessary condition for Huet's algorithm to generate the projection

$$X/\lambda_{B_1} \dots \lambda_{B_j} (\underline{j} - \underline{i} + \underline{1}(Z_1 \underline{j} \dots \underline{1}) \dots (Z_s \underline{j} \dots \underline{1}))$$

After a finite number of steps we get a unification system of the form: $X_{p_1} [e_{p_1_F}^1 \cdot \dots \cdot e_{1_F}^1 \cdot \uparrow^r] = \overset{?}{\lambda_\sigma} \lambda_{A_{r+1}} \dots \lambda_{A_{r+r'}} (\underline{m} e_{1_F}^2 \dots e_{p_2_F}^2) \wedge X = \overset{?}{\lambda_\sigma} \lambda_{B_1} \lambda_{B_{p_1}}.X_{p_1} \wedge Q^7$. Until this point we have at most $\frac{p_1(p_1+1)}{2}$ possible distinct projections generated in the $\lambda\sigma$ -HOU algorithm which corresponds to the projections substitutions with headings $\lambda_{B_1} \dots \lambda_{B_j} \underline{i}$ of Huet's algorithm. Certainly X_{p_1} has functional type because the right-hand side of the current equation is an abstraction and, hence another atomisation step of the form $X_{p_1} = \overset{?}{\lambda_\sigma} \lambda_{A_{r+1}}.X_{p_1+1}$ is generated by the rule **Exp2**. After an application of **Replace** and **Normalise**, we get:

$$X_{p_1+1} [\underline{1}.e_{p_1_F}^1 [\uparrow] \cdot \dots \cdot e_{1_F}^1 [\uparrow] \cdot \uparrow^{r+1}] = \overset{?}{\lambda_\sigma} \lambda_{A_{r+2}} \dots \lambda_{A_{r+r'}} (\underline{m} e_{1_F}^2 \dots e_{p_2_F}^2) \wedge X = \overset{?}{\lambda_\sigma} \lambda_{B_1} \lambda_{B_{p_1+1}}.X_{p_1+1} \wedge Q. \text{ In this case, } R_p \subseteq \{2, \dots, p_1 + 1\} \text{ because the "projection" over the first argument fail since the right-hand side of the equation is an abstraction. In this case we have at most } p_1 \text{ distinct possible}$$

⁷ Of course, the conjunctions in Q are changing, but we decided keep the name Q for clarity.

projections. One such “projection” $\underline{i} \in R_p \subseteq \{2, \dots, p_1 + 1\}$ can be generated only if there exists an $s \geq 0$ such that $\tau(e_{p_1-i+2}^1) = C_1 \rightarrow \dots \rightarrow C_s \rightarrow B_{p_1} \rightarrow A_{r+1} \rightarrow \dots \rightarrow A_{r+r'} \rightarrow C$. A similar situation happens during the following applications of this strategy until we get the unification system: $X_{p_1+r'}[\underline{1} \dots \underline{r}].e_{p_1F}^1[\uparrow^{r'}] \dots .e_{1F}^1[\uparrow^{r'}].\uparrow^{r+r'} =_{\lambda\sigma}^? \underline{m} e_{1F}^2 \dots e_{p_2F}^2 \wedge X =_{\lambda\sigma}^? \lambda_{B_1} \lambda_{B_{p_1+r'}} . X_{p_1+r'} \wedge Q$, and only at this point, we may apply the rule **Exp** which requires a similar type condition to generate at most p_1 different equations of the form $X =_{\lambda\sigma}^? \lambda_{B_1} \dots \lambda_{B_{p_1}} \lambda_{A_{r+1}} \dots \lambda_{A_{r+j}} . (\underline{p_1 + j - i + 1} Z_1 \dots Z_s)$. In this way, each equation of the form $X =_{\lambda\sigma}^? \lambda_{B_1} \dots \lambda_{B_j} . (\underline{j - i + 1} H_1 \dots H_s)$ corresponds to the substitution $X/\lambda_{B_1} \dots \lambda_{B_j} . (\underline{j - i + 1} Z_1 \dots Z_s)$. In total, we have at most $p_1 \cdot r'$ possible distinct “projections” as in Huet’s algorithm. If $m > r + r'$ then \underline{m} is a constant and hence $\underline{p_1 + m - r} \in R_i$. In this case, Huet’s algorithm always generates the substitution:

$$X/\lambda_{B_1} \dots \lambda_{B_{p_1+r'}} . (\underline{p_1 + m - r} (Z_1 \underline{p_1 + r'} \dots \underline{1}) \dots (Z_{p_2} \underline{p_1 + r'} \dots \underline{1})).$$

If $1 \leq m \leq r'$ then the equation $X_{p_1+r'} =_{\lambda\sigma}^? (\underline{m} H_1 \dots H_{p_2})$ is generated by the rule **Exp** and, hence we have $X =_{\lambda\sigma}^? \lambda_{B_1} \dots \lambda_{B_{p_1+r'}} . (\underline{m} H_1 \dots H_{p_2})$. This equation corresponds to the imitation substitution:

$$X/\lambda_{B_1} \dots \lambda_{B_{p_1+r'}} . (\underline{m} (Z_1 \underline{p_1 + r'} \dots \underline{1}) \dots (Z_{p_2} \underline{p_1 + r'} \dots \underline{1}))$$

generated by Huet’s algorithm.

– $r' = 0$: In this case, the original unification problem has the form

$$\lambda_{A_1} \dots \lambda_{A_r} . (X[\uparrow^r] e_{1F}^1 \dots e_{p_1F}^1) =_{\lambda\sigma}^? \lambda_{A_1} \dots \lambda_{A_r} . (\underline{m} e_{1F}^2 \dots e_{p_2F}^2)$$

where $p_1, p_2, r \geq 0$, $m > 0$ and $\tau(X) = B_1 \rightarrow \dots \rightarrow B_{p_1} \rightarrow A_{r+1} \rightarrow \dots \rightarrow A_{r+r'} \rightarrow C$.

If $m > r$ then $R_i \neq \emptyset$ and the equation $X =_{\lambda\sigma}^? (\underline{m - r} H_1 \dots H_{p_2-p_1})$ is generated. The replacement of X by the right side of this equation will not fail in the next application of $\text{SIMPL}_{\lambda\sigma}$, only if $p_2 - p_1 \geq 0$, and in this case, this equation corresponds to the substitution $X/(\underline{m - r} Z_1 \dots Z_{p_2-p_1})$. Since X has functional type, an atomisation step $X =_{\lambda\sigma}^? \lambda_{B_1} . X_1$ is generated. After an application of **Replace** and **Normalise** we get a unification problem with the equation $X_1[e_{1F}^1 \dots \uparrow^r] e_{2F}^2 \dots e_{p_1F}^1 =_{\lambda\sigma}^? \underline{m} e_1^2 \dots e_{p_2}^2$. Again, we have that $R_i \neq \emptyset$ and the equation $X_1 =_{\lambda\sigma}^? (\underline{1 + m - r} H_1 \dots H_{p_2-p_1+1})$ is generated and, the replacement of X_1 will not fail in the next application of $\text{SIMPL}_{\lambda\sigma}$ only if $p_2 - p_1 + 1 \geq 0$ and $\tau(e_i^1) = \tau(e_{p_2-p_1+i}^2) \forall 1 < i \leq p_1$. Hence, we get $X =_{\lambda\sigma}^? \lambda_{B_1} . (\underline{1 + m - r} H_1 \dots H_{p_2-p_1+1})$ which corresponds to the imitation substitution $X/\lambda_{B_1} . (\underline{1 + m - r} (Z_1 \underline{1}) \dots (Z_{p_2-p_1+1} \underline{1}))$ of Huet. According to the type of e_{1F}^1 , the equation $X_1 =_{\lambda\sigma}^? (\underline{1} H_1 \dots H_s)$ is generated, and after a replacement we have $X =_{\lambda\sigma}^? \lambda_{B_1} (\underline{1} H_1 \dots H_s)$, where $1 \in R_p$. Since the precooking, and hence its inverse, is a type preserving function, we conclude that the corresponding substitution $X/\lambda_{B_1} . (\underline{1} (Z_1 \underline{1}) \dots (Z_s \underline{1}))$ is generated by Huet’s algorithm. In general, after each atomisation step, we have a unification system containing the equation

$$X_j[e_{jF}^1 \dots e_{1F}^1 \uparrow^r] e_{j+1F}^2 \dots e_{p_1F}^1 =_{\lambda\sigma}^? \underline{m} e_1^2 \dots e_{p_2}^2$$

and the element in R_i generates the equation

$$X_j =_{\lambda\sigma}^? (\underline{j + m - r} H_1 \dots H_{p_2 - p_1 + j})$$

which will not fail in the next application of $\text{SIMPL}_{\lambda\sigma}$ only if $p_2 - p_1 + j \geq 0$ and $\tau(e_i^1) = \tau(e_{p_2 - p_1 + i}^2)$, $\forall j < i \leq p_1$. In this case, the generated equation $X =_{\lambda\sigma}^? \lambda_{B_1} \dots \lambda_{B_i} (\underline{j + m - r} H_1 \dots H_{p_2 - p_1 + j})$ corresponds to the substitution $X/\lambda_{B_1} \dots \lambda_{B_j} (\underline{j + m - r} (Z_1 \underline{j} \dots \underline{1})) \dots (Z_{p_2 - p_1 + j} \underline{j} \dots \underline{1})$, where $1 \leq j \leq p_1$. Hence, we have at most $\min(p_1, p_2) + 1$ possible distinct imitations. The possible projections in this case are analysed in the same way as in the previous case for the headings of the form $\lambda_{B_1} \dots \lambda_{B_j} \underline{i}$ with $1 \leq i \leq j \leq p_1$.

□

In the $\lambda\sigma$ -calculus, our approach according to Huet's algorithm can be seen as successive calls to the procedures $\text{SIMPL}_{\lambda\sigma}$ and $\text{MATCH}_{\lambda\sigma}$ according to the following description. In this description, we can take into account the improvements described in [Bor95]. To do so, we need to consider a normalisation rule which calculates only weak head normal forms and weak **Eta**-long forms of the terms instead of **Eta**-long forms.

Main Procedure

INPUT: A unification system P_ϵ .

OUTPUT: A success or a failure status and in the former case the solutions are the solved equations whose left-hand side corresponds to the meta-variables of the initial problem. If the initial problem is non-unifiable the algorithm may not terminate.

1. If P_q contains a rigid-rigid equation, then apply $\text{SIMPL}_{\lambda\sigma}$ and go to the next step, else if P_q contains a non-solved flexible-rigid equation then rename it to \overline{P}_q and go to the next step.
2. Apply $\text{MATCH}_{\lambda\sigma}$ to \overline{P}_q and let $P_{q1} \vee \dots \vee P_{qr}$ be the resulting unification problem.
3. If the current unification problem contains a unification system not in solved form then select it and go to step 1, else stop and report a success status.

In the appendix, we solve the same unification problem of Example 1 using our approach to the $\lambda\sigma$ -HOU algorithm. We have so far established a comparison for the two procedures that forms the kernel of Huet's algorithm in the pure λ -calculus and in the $\lambda\sigma$ -calculus of explicit substitutions. This comparison allows us to better understand both Huet's algorithm and $\lambda\sigma$ -HOU algorithm. It is not surprising that the procedures SIMPL and MATCH can be simulated in a calculus of explicit substitutions, but the way back, that is, how graftings (the first-order substitutions generated by the $\lambda\sigma$ -HOU algorithm) in the $\lambda\sigma$ -calculus can be seen in the pure λ -calculus, gives good insights about how the $\lambda\sigma$ -calculus

generates the corresponding projection and imitation substitutions for unification problems that can be back translated to the λ -calculus. An interesting fact that carries out from our analysis is that, in the $\lambda\sigma$ -calculus without the **Eta**-conversion, the equations corresponding to some imitation substitutions in the pure λ -calculus are generated by elements in the set R_p which is supposed to generate projections. These comparisons, however, cannot be extended to general unification problems in the $\lambda\sigma$ -calculus since $\lambda\sigma$ -unification problems containing operators of the extended language which do not have a counterpart in the pure λ -calculus. That is the reason why the corresponding matching trees of Example 3 and the one in the appendix do not need to be equal. Nevertheless the final results of the original unification problem, of course, must be the same.

5 λs_e -HOU versus Huet's algorithm

In this section, we use particular strategies of the rules presented in [ARK01] to conclude that the algorithm presented by Ayala-Rincón and Kamareddine is a generalisation of Huet's one.

Definition 18 ([ARK03]). Take $a \in \Lambda_{dB}(\mathcal{X})$ where $\Gamma \vdash a : T$. We give to every meta-variable X of type A in a , the same type and context Γ in the λs_e -calculus. The precooking of a from Λ_{dB} to the λs_e -calculus is defined by $a_{pc} = PC(a, 0)$ where $PC(a, n)$ is defined by:

$$\begin{aligned} (a) \quad PC(\lambda_B.a, n) &= \lambda_B.PC(a, n+1) & (b) \quad PC((a \ b), n) &= PC(a, n) \ PC(b, n) \\ (c) \quad PC(\underline{k}, n) &= \underline{k} & (d) \quad PC(X, n) &= \begin{cases} X, & \text{if } n = 0 \\ \varphi_0^{n+1} X, & \text{otherwise.} \end{cases} \end{aligned}$$

Definition 19 ([ARK01]). Let t be a λs_e -normal term whose root operator is either σ or φ and let X be its leftmost innermost meta-variable. Denote by $\psi_{i_k}^{j_k}$ the k -th operator following the sequence of operators σ and φ , considering only left arguments of the σ operators, in the innermost outermost ordering. Additionally, if $\psi_{i_k}^{j_k}$ corresponds to an operator φ then j_k and i_k denote its superscripts and subscripts, respectively, and if $\psi_{i_k}^{j_k}$ corresponds to an operator σ then $j_k = 0$ and i_k denote its superscript. Let a_k denote the corresponding right argument of the k -th operator if $\psi_{i_k}^{j_k} = \sigma^{i_k}$ and the empty argument if $\psi_{i_k}^{j_k} = \varphi_{i_k}^{j_k}$. The skeleton of t , written as $sk(t)$, is $\psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p)$.

Example 2. The skeleton of the λs_e -term $\varphi_{i_5}^{j_5}((\varphi_{i_3}^{j_3}(\varphi_{i_2}^{j_2}(X \sigma^{i_1} a)))\sigma^{i_4} b)$ is given by $\varphi_{i_5}^{j_5} \sigma^{i_4} \varphi_{i_3}^{j_3} \varphi_{i_2}^{j_2} \sigma^{i_1}(X, a, b)$.

The following result is worthwhile to understand the structure of a λs_e -normal form for flexible terms:

Lemma 10 ([ARK01]). Let t be a λs_e -normal term whose root operator is either σ or φ and let $sk(t) = \psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p)$. Successive subscripts i_k and i_{k+1} satisfy the following:

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

According to this lemma, the sequence i_1, \dots, i_p of subscripts is non-increasing. Hence, λ_{s_e} -flexible terms in normal form can be determined up to simple arithmetic constraints.

The unification rules in the λ_{s_e} -calculus are given in Tables 5 and 6.

Dec-λ	$\frac{P \wedge \lambda_A.e_1 \stackrel{?}{=} \lambda_A.e_2}{P \wedge e_1 \stackrel{?}{=} e_2}$
Dec-App	$\frac{P \wedge \underline{n}(e_1^1, \dots, e_p^1) \stackrel{?}{=} \underline{n}(e_1^2, \dots, e_p^2)}{P \wedge e_1^1 \stackrel{?}{=} e_1^2 \wedge \dots \wedge e_p^1 \stackrel{?}{=} e_p^2}$
App-Fail	$\frac{P \wedge \underline{n}(e_1^1, \dots, e_{p_1}^1) \stackrel{?}{=} \underline{m}(e_1^2, \dots, e_{p_2}^2)}{Fail}, \text{ if } m \neq n.$
Dec-App-λ	$\frac{P \wedge \lambda_A.e \stackrel{?}{=}_{\lambda\sigma} (\underline{m} e_1^2 \dots e_{p_2}^2)}{Fail}$

Table 5. Unification Rules for the λ_{s_e} -calculus (part I)

5.1 A note on the λ_{s_e} -HOU rules.

The rules presented in Tables 5 and 6 are slightly different from the ones presented in [ARK01]. In particular, we added the rule **Dec-App- λ** for unification without **Eta**-conversion and we omitted the rule **Dec- φ** . In fact, the rule **Dec- φ** is not necessary in the λ_{s_e} -HOU algorithm (despite its correctness and completeness) because it is not useful neither to foresee a failure status nor to decrease the number of unification steps since it is supposed to be applied during the application of a normalisation step.

The following result formalises the fact that the rule **Dec- φ** is not essential in the λ_{s_e} -unification algorithm.

Lemma 11 ([ARK01]). *Solved problems are normalised for the λ_{s_e} -unification rules and, conversely, if a system is a conjunction of equations that cannot be reduced by applying the λ_{s_e} -unification rules then it is solved.*

Proof. The proof is the same as that of Lemma 5.10 (page 515) in [ARK01], except that the last paragraph of this proof should be omitted because, by hypothesis we are considering a normalised problem and hence, there is no equation with φ as main operator on both sides of a flexible-rigid or a rigid-rigid equation in λ_{s_e} -normal form. \square

Exp-λ	$\frac{P}{\exists Y : (A.\Gamma \vdash B), P \wedge X \stackrel{?}{=}_{\lambda\sigma} \lambda_A Y}$ if $(X : \Gamma \vdash A \rightarrow B) \in \mathcal{TVar}(P)$, $Y \notin \mathcal{TVar}(P)$, and X is not a solved variable.
Exp-App	$\frac{P \wedge \psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p) \stackrel{?}{=}_{\lambda s_e} \underline{\mathbb{M}}(b_1, \dots, b_q)}{P \wedge \psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p) \stackrel{?}{=}_{\lambda\sigma} \underline{\mathbb{M}}(b_1, \dots, b_q) \wedge \bigvee_{r \in R_p \cup R_i} \exists H_1 \dots \exists H_k, X \stackrel{?}{=}_{\lambda s_e} \underline{\mathbb{I}}(H_1, \dots, H_k)}$ if $\psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p)$ is the skeleton of a λs_e normal term, and X has an atomic type and is not solved. where H_1, \dots, H_k are meta-variables of appropriate types, not occurring in P , with the contexts $\Gamma_{H_i} = \Gamma_X$, R_p is the subset of $\{i_1, \dots, i_p\}$ of superscripts of the σ operator such that $\underline{\mathbb{I}}(H_1, \dots, H_k)$ has the right type, $R_i = \bigcup_{k=0}^p$ if $i_k \geq m + p - k - \sum_{l=k+1}^p j_l > i_{k+1}$ then $\{m + p - k - \sum_{l=k+1}^p j_l\}$ else \emptyset , where $i_0 = \infty$ and $i_{p+1} = 0$.
Replace	$\frac{P \wedge X \stackrel{?}{=}_{\lambda s_e} t}{\{X/t\}(P) \wedge X \stackrel{?}{=}_{\lambda s_e} t}$ if $X \in \mathcal{TVar}(P)$, $X \notin \mathcal{TVar}(t)$ and if $t \in \mathcal{X} \Rightarrow t \in \mathcal{TVar}(P)$.
Normalise	$\frac{P \wedge e_1 \stackrel{?}{=}_{\lambda s_e} e_2}{P \wedge e'_1 \stackrel{?}{=}_{\lambda\sigma} e'_2}$ if e_1 or e_2 is not in long form. where e'_1 (resp. e'_2) is the long form of e_1 (resp. e_2) if e_1 (resp. e_2) is not a solved variable and e_1 (resp. e_2) otherwise.

Table 6. Unification Rules for the λs_e -calculus (part II)

Theorem 5 ([ARK01]). *The λ_{s_e} -unification rules are correct and complete.*

Proof. Here we detail the proof which is only sketched in [ARK01]. We split the proof in two parts:

1. Correctness:

Dec- λ : If σ is a unifier of $a =_{\lambda_{s_e}}^? b$ then σ is also a unifier of $\lambda_A.a =_{\lambda_{s_e}}^? \lambda_A.b$ because no substitution applies on the root of the terms $\lambda_A.a$ or $\lambda_A.b$ and hence $(\lambda_A.a)\sigma =_{\lambda_{s_e}} (\lambda_A.b)\sigma \Rightarrow \lambda_A.(a\sigma) =_{\lambda_{s_e}} \lambda_A.(b\sigma)$, which is unifiable by hypothesis.

Dec-App: Let σ be a unifier of the conjunction of equations

$$a_1 =_{\lambda_{s_e}}^? b_1 \wedge \dots \wedge a_n =_{\lambda_{s_e}}^? b_n$$

Applying σ to the equation $\underline{\mathbf{k}}a_1 \dots a_n =_{\lambda_{s_e}}^? \underline{\mathbf{k}}b_1 \dots b_n$, we get $\underline{\mathbf{k}}a_1\sigma \dots a_n\sigma =_{\lambda_{s_e}}^? \underline{\mathbf{k}}b_1\sigma \dots b_n\sigma$ which is unifiable.

App-Fail: Straightforward since *Fail* has no unifier.

Exp- λ : Let σ be a unifier of $P \wedge X =_{\lambda_{s_e}}^? \lambda_A.Y$ where X is a meta-variable of functional type that occurs in P . Then σ is also a unifier of smaller (and simpler) unification problem P .

Exp-App: Let σ be a unifier of $P \wedge X =_{\lambda_{s_e}}^? \underline{\mathbf{r}}b_1 \dots b_q$, where X is a meta-variable of atomic type that occurs in P . Then certainly σ is also a unifier of P since it is a simpler problem, that is, it is the same original problem without one equation.

Replace: Let σ be a unifier of $P\{X/a\} \wedge X =_{\lambda_{s_e}}^? a$. Then σ also unifies $P \wedge X =_{\lambda_{s_e}}^? a$ because they are the same unification problem but $P\{X/a\}$ does not have occurrences of X anymore.

Normalise: Let σ be a unifier of $P \wedge a' =_{\lambda_{s_e}}^? b'$, where a' and b' are the normal forms of a and b respectively. Certainly, all the meta-variables that appear in a' (resp. b') also appear in a (resp. b). After applying σ to $a =_{\lambda_{s_e}}^? b$ the β -redexes remain unchanged and, after a normalisation we get $a'\sigma =_{\lambda_{s_e}}^? b'$ which is unifiable by hypothesis.

2. Completeness:

Dec- λ : Let σ be a unifier of $\lambda_A.a =_{\lambda_{s_e}}^? \lambda_A.b$. Since σ does not apply on the root of the terms, we have that $(\lambda_A.a)\sigma =_{\lambda_{s_e}}^? (\lambda_A.b)\sigma$ must be the same as $\lambda_A.(a\sigma) =_{\lambda_{s_e}}^? \lambda_A.(b\sigma)$ and therefore, $a\sigma =_{\lambda_{s_e}}^? b\sigma$.

Dec-App: Let σ be a unifier of $(\underline{\mathbf{k}}a_1 \dots a_n) =_{\lambda_{s_e}}^? (\underline{\mathbf{k}}b_1 \dots b_n)$. Then since the typed λ_{s_e} -calculus is weakly terminating and confluent we have that $(\underline{\mathbf{k}}a_1 \dots a_n)\sigma =_{\lambda_{s_e}}^? (\underline{\mathbf{k}}b_1 \dots b_n)\sigma \Leftrightarrow \underline{\mathbf{k}}a_1\sigma \dots a_n\sigma =_{\lambda_{s_e}}^? \underline{\mathbf{k}}b_1\sigma \dots b_n\sigma$, and then σ unifies $a_i =_{\lambda_{s_e}}^? b_i, \forall 1 \leq i \leq n$.

App-Fail: Straightforward since a rigid-rigid equation with different heads is not unifiable.

Exp- λ : If σ is a unifier of P in which the meta-variable X of functional type, say $A \rightarrow B$, occur and $X\sigma = a$ then we can assume that a is of the form $\lambda_A.a'$ where a' has type B . Let us define σ' such that $\forall X \in \text{Dom}(\sigma), X\sigma = X\sigma'$ and $Y\sigma' = a'$. Then σ' is a unifier of $P \wedge X =_{\lambda_{s_e}}^? \lambda_A.Y$, which shows that σ

unifies $P \wedge X =_{\lambda_{s_e}}^? \lambda_A.Y$.

Exp-App: Done in [ARK01].

Normalise: Straightforward since grafting and reduction commute.

Replace: The proof is as in the first order case. \square

In the following, we give a description of the λ_{s_e} -HOU algorithm according to Huet's approach.

Procedure $\text{SIMPL}_{\lambda_{s_e}}$

INPUT: A unification problem P_q with at least one rigid-rigid equation.

OUTPUT: A terminal (failure or success) status or an equivalent unification problem \overline{P}_q without rigid-rigid equations and containing at least one flexible-rigid equation.

Assume that **Dec- λ** is applied eagerly.

WHILE there exists a rigid-rigid equation in P_q DO:

1. Apply **Dec-App- λ** or **App-Fail**.
2. Apply **Dec-App** and, if the resulting unification problem contains a flexible-rigid equation, call it \overline{P}_q and give \overline{P}_q as result, else stop and report a success status.

DONE.

Procedure $\text{MATCH}_{\lambda_{s_e}}$

INPUT: A unification system P_q with at least one flexible-rigid equation.

OUTPUT: A disjunction of equivalent unification systems, written $P_{q1} \vee \dots \vee P_{qr}$.

Assume that **Dec- λ** is applied eagerly.

1. Apply **Exp- λ** and **Replace** as much as possible to the selected flexible-rigid equation and call P'_q the resulting unification system.
2. Apply **Exp-App** and **Replace** and **Normalise** to P'_q and call $P_{q1} \vee \dots \vee P_{qr}$ the resulting unification problem.

The Main Procedure

INPUT: A unification system P_ϵ .

OUTPUT: A success or a failure status and in the former case the solutions are the solved equations whose left-hand side corresponds the meta-variables of the initial problem. If the initial problem is non-unifiable the algorithm may not terminate.

1. If P_q contains a rigid-rigid equation then apply $\text{SIMPL}_{\lambda_{s_e}}$ to it, else if P_q contains a non-solved flexible-rigid equation then rename it to \overline{P}_q and go to the next step.
2. Apply $\text{MATCH}_{\lambda_{s_e}}$ to \overline{P}_q and let $P_{q1} \vee \dots \vee P_{qr}$ be the resulting unification problem.
3. If the current unification problem contains a unification system not in solved form then select it and go to step 1, else stop and report a success status.

6 Comparing the $\lambda\sigma$ - and the λs_e -styles of unification

In this section we give a formal comparison between the $\lambda\sigma$ - and the λs_e -HOU algorithms. This is done by operators that translate $\lambda\sigma$ -terms into λs_e -terms and vice-versa. Note that the rules used to simplify rigid-rigid equations are essentially the same in both calculi: **Dec- λ** , **Dec-App** and **Dec-Fail** (which is called **App-Fail** in the λs_e -calculus) which allow us to conclude that $\text{SIMPL}_{\lambda\sigma}$ and $\text{SIMPL}_{\lambda s_e}$ do the same job for rigid-rigid equations.

The comparison between $\text{MATCH}_{\lambda\sigma}$ and $\text{MATCH}_{\lambda s_e}$ consists basically in showing that the equations generated by the former correspond, in the sense of Definition 17, to the equations generated by the latter and vice-versa. This definition can be specialised to the explicit substitutions calculi treated here as follows:

Definition 20. *Let $X =?_{\lambda\sigma} a$ and $X =?_{\lambda s_e} a'$ be two flexible-rigid equations in the $\lambda\sigma$ - and λs_e -calculus respectively. These equations are said to be corresponding (or associated) if a and a' have the same heading.*

Equations as in the previous definition are generated by the rule **Exp-App** of both calculi, that we will denote so far by **Exp-App** $_{\lambda\sigma}$ and **Exp-App** $_{\lambda s_e}$.

We start by defining an operator that translates λs_e -terms into $\lambda\sigma$ -terms.

Definition 21. *The operator $T : \Lambda_{\lambda s_e} \rightarrow \Lambda_{\lambda\sigma}$ is defined inductively as:*

- (a) $T(X) = X$ (b) $T(\underline{n}) = \underline{1}[\uparrow^{n-1}]$ (c) $T(a b) = T(a) T(b)$ (d) $T(\lambda.a) = \lambda.T(a)$
- (e) $T(a\sigma^i b) = T(a)[\underline{1}, \underline{2}, \dots, \underline{i-1}.T(b)[\uparrow^{i-1}], \uparrow^{i-1}]$, where $i \geq 1$.
- (f) $T(\varphi_k^i(a)) = T(a)[\underline{1}, \underline{2}, \dots, \underline{k}, \uparrow^{k+i-1}]$, where $k \geq 0$ and $i \geq 1$.

If $r = 0$ in the list $\underline{1}, \dots, \underline{r}$, then it is to be interpreted as the empty list. In addition, $\uparrow^0 = id$.

The next proposition shows that the operator T translates a λs_e -term into a $\lambda\sigma$ -term preserving the context and the type of the term as well as the corresponding sub-terms.

Proposition 7 (Preservation of types by T). *Let Γ be a context, A a type and a a term in the language of the λs_e -calculus such that $\Gamma \vdash a : A$. Then $\Gamma \vdash T(a) : A$.*

Proof. Let $a \in \Lambda_{s_e}$ such that $\Gamma \vdash a : A$, and note that a context is represented by a list of types in both the $\lambda\sigma$ and the λs_e -calculus. The proof is by structural induction on a and in this proof as well as in the proof of Proposition 8 we use decorated terms like a_A^Γ meaning that the term a has context Γ and type A .

- If $a_A^\Gamma = X_A^\Gamma$ then $T(X_A^\Gamma) = X_A^\Gamma = T(a)_A^\Gamma$.
- If $a_A^\Gamma = \underline{n}_A^\Gamma$ then the n -th element of Γ must be the type A , therefore $T(\underline{n}_A^\Gamma) = \underline{n}_A^\Gamma = T(a)_A^\Gamma$.
- If $a_A^\Gamma = (b c)_A^\Gamma$ then, by the type rules of the λs_e -calculus, there exists a type B such that $b_{B \rightarrow A}^\Gamma$, and hence $(b_{B \rightarrow A}^\Gamma c_B^\Gamma)_A^\Gamma$. Therefore, $T(a_A^\Gamma) = T(b_{B \rightarrow A}^\Gamma c_B^\Gamma) \stackrel{def}{=} T(b_{A \rightarrow B}^\Gamma) T(c_A^\Gamma) \stackrel{HI}{=} T(b)_{A \rightarrow B}^\Gamma T(c)_A^\Gamma = (T(b) T(c))_A^\Gamma = T(a)_A^\Gamma$.

- If $a_A^\Gamma = (\lambda_B.b)_A^\Gamma$ then $b_C^{A,\Gamma}$ and $T(\lambda_A.a_B^\Gamma) = \lambda_A.T(a)_B^\Gamma$
- $T((a_A^{\Gamma_{<i}.B.\Gamma_{\geq i}} \sigma^i b_B^{\Gamma_{\geq i}})_A^\Gamma) =$
 $(T(a)_A^{\Gamma_{<i}.B.\Gamma_{\geq i}} [(\underline{1.2} \dots \underline{i-1}.T(b)_B^{\Gamma_{\geq i}} [\uparrow^{i-1}]. \uparrow^{i-1})_{\Gamma_{<i}.B.\Gamma_{\geq i}}])_A^\Gamma$, where $i \geq 1$.
- $T((\varphi_k^i (a_A^{\Gamma_{\leq k}.\Gamma_{\geq k+i}}))_A^\Gamma) = T(a)_A^{\Gamma_{\leq k}.\Gamma_{\geq k+i}} [(\underline{1.2} \dots \underline{k-1.k} \uparrow^{k+i-1})_{\Gamma_{\leq k}.\Gamma_{\geq k+i}}]_A^\Gamma$,
 where $k \geq 0$ and $i \geq 1$.

□

The following lemmas are technical results used to prove the correspondence between the procedures $\text{MATCH}_{\lambda s_e}$ and $\text{MATCH}_{\lambda \sigma}$.

Lemma 12. *Let $\varphi_{i_p}^{j_p} \dots \varphi_{i_1}^{j_1}(X)$ be the skeleton of a flexible term in λs_e -normal form. Then, the $\lambda \sigma$ -normal form of $T(\varphi_{i_p}^{j_p} \dots \varphi_{i_1}^{j_1}(X))$ is given by*

$$X[\underline{1.2} \dots \underline{i_p-1.i_p.i_p+1+(j_p-1)} \dots \underline{i_{p-1}-1+(j_{p-1}-1).i_{p-1}+(j_{p-1}).i_{p-1}+1+(j_p+j_{p-1}-2)} \dots \underline{i_k-1-(p-k)+\sum_{l=k+1}^p j_l} \dots \underline{i_k-(p-k)+\sum_{l=k+1}^p j_l.i_k+1-(p-k+1)+\sum_{l=k}^p j_l} \dots \underline{i_1-1-(p-1)+\sum_{l=2}^p j_l.i_1-(p-1)+\sum_{l=2}^p j_l} \uparrow^{i_1-p+\sum_{l=1}^p j_l}]$$

Proof. By induction on p . If $p = 1$ then $X[\underline{1.2} \dots \underline{i_1-1.i_1} \uparrow^{i_1+j_1-1}]$ is obtained by the definition of T . Suppose that the result holds for $p-1$, i.e., that the normal form of $T(\varphi_{i_{p-1}}^{j_{p-1}} \dots \varphi_{i_1}^{j_1}(X))$ is given by

$$X[\underline{1.2} \dots \underline{i_{p-1}-1.i_{p-1}.i_{p-1}+1+(j_{p-1}-1)} \dots \dots \underline{i_k-1-(p-k-1)+\sum_{l=k+1}^{p-1} j_l.i_k-(p-k-1)+\sum_{l=k+1}^{p-1} j_l} \dots \underline{i_k+1-(p-k)+\sum_{l=k}^{p-1} j_l} \dots \underline{i_1-1-(p-2)+\sum_{l=2}^{p-1} j_l} \dots \underline{i_1-(p-2)+\sum_{l=2}^{p-1} j_l} \uparrow^{i_1-(p-1)+\sum_{l=1}^{p-1} j_l}].$$

The induction step is given by:

$$T(\varphi_{i_p}^{j_p} \dots \varphi_{i_1}^{j_1}(X)) \stackrel{def}{=} T(\varphi_{i_{p-1}}^{j_{p-1}} \dots \varphi_{i_1}^{j_1}(X))[\underline{1.2} \dots \underline{i_p-1.i_p} \uparrow^{i_p+j_p-1}] \stackrel{HI}{=} X[\underline{1.2} \dots \underline{i_{p-1}-1.i_{p-1}.i_{p-1}+1+(j_{p-1}-1)} \dots \dots \underline{i_k-1-(p-k-1)+\sum_{l=k+1}^{p-1} j_l.i_k-(p-k-1)+\sum_{l=k+1}^{p-1} j_l} \dots \underline{i_k+1-(p-k)+\sum_{l=k}^{p-1} j_l} \dots \underline{i_1-1-(p-2)+\sum_{l=2}^{p-1} j_l} \dots \underline{i_1-(p-2)+\sum_{l=2}^{p-1} j_l} \uparrow^{i_1-(p-1)+\sum_{l=1}^{p-1} j_l}][\underline{1.2} \dots \underline{i_p-1.i_p} \uparrow^{i_p+j_p-1}]$$

and we get the result after σ -normalisation because $i_p < i_{p-1}$ according to Lemma 10. □

Lemma 13. *Let $\sigma^{i_p} \dots \sigma^{i_1}(X, a_1, \dots, a_p)$ be the skeleton of a flexible term in λs_e -normal form. Then, the $\lambda \sigma$ -normal form of $T(\sigma^{i_p} \dots \sigma^{i_1}(X, a_1, \dots, a_p))$ is given by*

$$X[\underline{1.2} \dots \underline{i_p-1.a_p}[\uparrow^{i_p-1}].\underline{i_p} \dots \underline{i_{p-1}-2.a_{p-1}}[\uparrow^{i_{p-1}-2}].\underline{i_{p-1}-1} \dots \dots \underline{i_k-1-(p-k).a_k}[\uparrow^{i_k-1-(p-k)}].\underline{i_k-(p-k)} \dots \underline{i_1-p.a_1}[\uparrow^{i_1-p}]. \uparrow^{i_1-p}].$$

Proof. By induction on p . If $p = 1$ then

$$T(\sigma^{i_1}(X, a_1)) = X[\underline{1.2} \cdots \underline{i_1 - 1}. a_1[\uparrow^{i_1-1}]. \uparrow^{i_1-1}]$$

is consequence of the definition of T . Suppose that the result holds for $p - 1$, that is, $T(\sigma^{i_{p-1}} \dots \sigma^{i_1}(X, a_1, \dots, a_{p-1})) = X[\underline{1.2} \cdots \underline{i_{p-1} - 1}.$

$$a_{p-1}[\uparrow^{i_{p-1}-1}]. \underline{i_{p-1}} \cdots \underline{i_k - p + k}. a_k[\uparrow^{i_k-p+k}]. \underline{i_k - p + k + 1} \cdots \underline{i_1 - p + 1}. a_1[\uparrow^{i_1-p+1}]. \uparrow^{i_1-p+1}].$$

The induction step is given by: $T(\sigma^{i_p} \dots \sigma^{i_1}(X, a_1, \dots, a_p)) \stackrel{def}{=} T(\sigma^{i_{p-1}} \dots \sigma^{i_1}(X, a_1, \dots, a_{p-1}))[\underline{1} \cdots \underline{i_p - 1}. a_p[\uparrow^{i_p-1}]. \uparrow^{i_p-1}] \stackrel{HI}{=} X[\underline{1.2} \cdots \underline{i_{p-1} - 1}. a_{p-1}[\uparrow^{i_{p-1}-1}]. \underline{i_{p-1}} \cdots \underline{i_k - p + k}. a_k[\uparrow^{i_k-p+k}]. \underline{i_k - p + k + 1} \cdots \underline{i_1 - p + 1}. a_1[\uparrow^{i_1-p+1}]. \uparrow^{i_1-p+1}][\underline{1.2} \cdots \underline{i_p - 1}. a_p[\uparrow^{i_p-1}]. \uparrow^{i_p-1}]$ and we get the result after the σ -normalisation because $i_p < i_{p-1}$ according to Lemma 10. \square

Lemma 14. *Let $\psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p)$ be the skeleton of a flexible term in λ_{s_e} -normal form. Then, the $\lambda\sigma$ -normal form of $T(\psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p))$ is a flexible term of the form $X[b_1 \cdots b_r. \uparrow^n]$ such that, for each $\psi_{i_k}^{j_k}$ ($1 \leq k \leq p$) which is a σ operator, the i_k -th element of the substitution $b_1 \cdots b_r. \uparrow^n$ is equal to $a_k[s]$ for some substitution s .*

Proof. Suppose that $\psi_{i_k}^{j_k}$ is the first σ operator from the left to right in $\psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p)$. Then, according to Lemma 12, we have that

$$\begin{aligned} T(\psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p)) &= T(\sigma^{i_k} \psi_{i_{k-1}}^{j_{k-1}} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_k))[\underline{1.2} \cdots \underline{i_p - 1}. \underline{i_p}. \underline{i_p + 1} + (\underline{j_p} - 1). \cdots \underline{i_{p-1} - 1} + (\underline{j_p} - 1). \underline{i_{p-1}} + (\underline{j_p} - 1). \\ &\underline{i_{p-1} + 1} + (\underline{j_p} + \underline{j_{p-1}} - 2). \cdots \underline{i_{k+1} - 1} - (\underline{p} - \underline{k} - 1) + \sum_{l=k+2}^p \underline{j_l}. \\ &\underline{i_{k+1} - (\underline{p} - \underline{k} - 1) + \sum_{l=k+2}^p \underline{j_l}}. \uparrow^{i_{k+1} - (p-k) + \sum_{l=k+1}^p j_l}] = \\ &T(\psi_{i_{k-1}}^{j_{k-1}} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_{k-1}))[\underline{1} \cdots \underline{i_k - 1}. a_k[\uparrow^{i_k-1}]. \uparrow^{i_k-1}][\underline{1.2} \cdots \underline{i_p - 1}. \underline{i_p}. \underline{i_p + 1} + (\underline{j_p} - 1). \cdots \\ &\underline{i_{p-1} - 1} + (\underline{j_p} - 1). \underline{i_{p-1}} + (\underline{j_p} - 1). \underline{i_{p-1} + 1} + (\underline{j_p} + \underline{j_{p-1}} - 2). \cdots \\ &\underline{i_{k+1} - 1} - (\underline{p} - \underline{k} - 1) + \sum_{l=k+2}^p \underline{j_l}. \underline{i_{k+1} - (\underline{p} - \underline{k} - 1) + \sum_{l=k+2}^p \underline{j_l}}. \\ &\underline{i_{k+1} - (\underline{p} - \underline{k} - 1) + \sum_{l=k+2}^p \underline{j_l}}. \uparrow^{i_{k+1} - (p-k) + \sum_{l=k+1}^p j_l}] = \\ &T(\psi_{i_{k-1}}^{j_{k-1}} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_{k-1}))[\underline{1.2} \cdots \underline{i_p - 1}. \underline{i_p}. \underline{i_p + 1} + (\underline{j_p} - 1). \cdots \\ &\underline{i_{p-1} - 1} + (\underline{j_p} - 1). \underline{i_{p-1}} + (\underline{j_p} - 1). \underline{i_{p-1} + 1} + (\underline{j_p} + \underline{j_{p-1}} - 2). \cdots \\ &\underline{i_{k+1} - 1} - (\underline{p} - \underline{k} - 1) + \sum_{l=k+2}^p \underline{j_l}. \underline{i_{k+1} - (\underline{p} - \underline{k} - 1) + \sum_{l=k+2}^p \underline{j_l}}. \\ &\underline{i_{k+1} + 1} - (\underline{p} - \underline{k}) + \sum_{l=k+1}^p \underline{j_l} \cdots \underline{i_k - 1} - (\underline{p} - \underline{k}) + \sum_{l=k+1}^p \underline{j_l}. \\ &a_k[\uparrow^{i_k-1-(p-k)+\sum_{l=k+1}^p j_l}]. \uparrow^{i_k-1-(p-k)+\sum_{l=k+1}^p j_l}] \end{aligned}$$

and since $j_k = 0$ this term can be rewritten as

$$\begin{aligned} T(\psi_{i_{k-1}}^{j_{k-1}} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_{k-1})) &[\underline{1.2} \cdots \underline{i_p - 1}. \underline{i_p}. \underline{i_p + 1} + (\underline{j_p} - 1). \cdots \\ &\underline{i_{p-1} - 1} + (\underline{j_p} - 1). \underline{i_{p-1}} + (\underline{j_p} - 1). \underline{i_{p-1} + 1} + (\underline{j_p} + \underline{j_{p-1}} - 2). \cdots \\ &\underline{i_{k+1} - 1} - (\underline{p} - \underline{k} - 1) + \sum_{l=k+2}^p \underline{j_l}. \underline{i_{k+1} - (\underline{p} - \underline{k} - 1) + \sum_{l=k+2}^p \underline{j_l}}. \\ &\underline{i_{k+1} + 1} - (\underline{p} - \underline{k}) + \sum_{l=k+1}^p \underline{j_l} \cdots \underline{i_k - 1} - (\underline{p} - \underline{k}) + \sum_{l=k}^p \underline{j_l}. \end{aligned}$$

$$\begin{aligned}
 & a_k [\uparrow^{i_k-1-(p-k)+\sum_{l=k}^p j_l}]. \uparrow^{i_k-1-(p-k)+\sum_{l=k}^p j_l}]. \\
 & \text{If } \psi_{i_{k-1}}^{j_{k-1}} = \sigma^{i_{k-1}} \text{ then } T(\psi_{i_{k-2}}^{j_{k-2}} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_{k-2})) [\underline{1.2.} \dots \underline{i_{k-1}-1.} \\
 & a_{k-1} [\uparrow^{i_{k-1}-1}]. \uparrow^{i_{k-1}-1}][\underline{1.2.} \dots \underline{i_p-1. i_p. i_p+1+(j_p-1).} \dots \dots \\
 & \underline{i_{p-1}-1+(j_p-1). i_{p-1}+(j_p-1). i_{p-1}+1+(j_p+j_{p-1}-2).} \dots \dots \\
 & \underline{i_{k+1}-1-(p-k-1)+\sum_{l=k+2}^p j_l. i_{k+1}-(p-k-1)+\sum_{l=k+2}^p j_l.} \\
 & \underline{i_{k+1}+1-(p-k)+\sum_{l=k+1}^p j_l.} \dots \dots \underline{i_k-1-(p-k)+\sum_{l=k}^p j_l.} \\
 & a_k [\uparrow^{i_k-1-(p-k)+\sum_{l=k}^p j_l}]. \uparrow^{i_k-1-(p-k)+\sum_{l=k}^p j_l} \\
 & \text{and since } i_{k-1} > i_k, \text{ we have, after a } \sigma\text{-normalisation, that:} \\
 & T(\psi_{i_{k-2}}^{j_{k-2}} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_{k-2})) [\underline{1.2.} \dots \underline{i_p-1. i_p. i_p+1+(j_p-1).} \dots \\
 & \underline{i_{p-1}-1+(j_p-1). i_{p-1}+(j_p-1). i_{p-1}+1+(j_p+j_{p-1}-2).} \dots \dots \\
 & \underline{i_{k+1}-1-(p-k-1)+\sum_{l=k+2}^p j_l. i_{k+1}-(p-k-1)+\sum_{l=k+2}^p j_l.} \\
 & \underline{i_{k+1}+1-(p-k)+\sum_{l=k+1}^p j_l.} \dots \dots \underline{i_k-1-(p-k)+\sum_{l=k}^p j_l.} \\
 & a_k [\uparrow^{i_k-1-(p-k)+\sum_{l=k}^p j_l}]. \underline{i_k+1-(p-k)+\sum_{l=k}^p j_l.} \dots \dots \\
 & \underline{i_{k-1}-1-(p-k)+\sum_{l=k}^p j_l.} a_{k-1} [\uparrow^{i_{k-1}-1-(p-k)+\sum_{l=k}^p j_l}]. \uparrow^{i_{k-1}-(p-k)+\sum_{l=k}^p j_l} \\
 & \text{and since } j_{k-1} = 0 \text{ this term can be rewritten as} \\
 & T(\psi_{i_{k-2}}^{j_{k-2}} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_{k-2})) [\underline{1.2.} \dots \underline{i_p-1. i_p. i_p+1+(j_p-1).} \dots \\
 & \underline{i_{p-1}-1+(j_p-1). i_{p-1}+(j_p-1). i_{p-1}+1+(j_p+j_{p-1}-2).} \dots \dots \\
 & \underline{i_{k+1}-1-(p-k-1)+\sum_{l=k+2}^p j_l. i_{k+1}-(p-k-1)+\sum_{l=k+2}^p j_l.} \\
 & \underline{i_{k+1}+1-(p-k)+\sum_{l=k+1}^p j_l.} \dots \dots \underline{i_k-1-(p-k)+\sum_{l=k}^p j_l.} \\
 & a_k [\uparrow^{i_k-1-(p-k)+\sum_{l=k}^p j_l}]. \underline{i_k+1-(p-k)+\sum_{l=k}^p j_l.} \dots \dots \\
 & \underline{i_{k-1}-1-(p-k)+\sum_{l=k-1}^p j_l.} a_{k-1} [\uparrow^{i_{k-1}-1-(p-k)+\sum_{l=k-1}^p j_l}]. \\
 & \uparrow^{i_{k-1}-(p-k)+\sum_{l=k-1}^p j_l} \\
 & \text{and if the other operators are } \sigma \text{ operators the process continues in this way} \\
 & \text{because, in this case,} \\
 & i_{k-1} > i_{k-2} > \dots > i_1. \text{ If } \psi_{i_{k-1}}^{j_{k-1}} = \varphi_{i_{k-1}}^{j_{k-1}} \text{ then} \\
 & T(\psi_{i_{k-2}}^{j_{k-2}} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_{k-2})) [\underline{1.2.} \dots \underline{i_{k-1}.} \uparrow^{i_{k-1}+j_{k-1}-1}][\underline{1.2.} \dots \underline{i_p-1. i_p.} \\
 & \underline{i_p+1+(j_p-1).} \dots \dots \underline{i_{p-1}-1+(j_p-1). i_{p-1}+(j_p-1).} \\
 & \underline{i_{p-1}+1+(j_p+j_{p-1}-2).} \dots \dots \underline{i_{k+1}-1-(p-k-1)+\sum_{l=k+2}^p j_l.} \\
 & \underline{i_{k+1}-(p-k-1)+\sum_{l=k+2}^p j_l. i_{k+1}+1-(p-k)+\sum_{l=k+1}^p j_l.} \dots \dots \\
 & \underline{i_k-1-(p-k)+\sum_{l=k}^p j_l.} a_k [\uparrow^{i_k-1-(p-k)+\sum_{l=k}^p j_l}]. \uparrow^{i_k-1-(p-k)+\sum_{l=k}^p j_l}]. \\
 & \text{According to Lemma 10, we have that } i_{k-1} \geq i_k \text{ and hence the } \sigma\text{-normalisation} \\
 & \text{of the last term is given by} \\
 & T(\psi_{i_{k-2}}^{j_{k-2}} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_{k-2})) [\underline{1.2.} \dots \underline{i_p-1. i_p. i_p+1+(j_p-1).} \dots \\
 & \underline{i_{p-1}-1+(j_p-1). i_{p-1}+(j_p-1). i_{p-1}+1+(j_p+j_{p-1}-2).} \dots \dots \\
 & \underline{i_{k+1}-1-(p-k-1)+\sum_{l=k+2}^p j_l. i_{k+1}-(p-k-1)+\sum_{l=k+2}^p j_l.} \\
 & \underline{i_{k+1}+1-(p-k)+\sum_{l=k+1}^p j_l.} \dots \dots \underline{i_k-1-(p-k)+\sum_{l=k}^p j_l.} \\
 & a_k [\uparrow^{i_k-1-(p-k)+\sum_{l=k}^p j_l}]. \underline{i_k+1-(p-k+1)+\sum_{l=k}^p j_l.} \dots \dots \\
 & \underline{i_{k-1}-1-(p-k+1)+\sum_{l=k}^p j_l. i_{k-1}-(p-k+1)+\sum_{l=k}^p j_l.} \\
 & \uparrow^{i_{k-1}-1-(p-k+1)+\sum_{l=k}^p j_l}
 \end{aligned}$$

and if other operators in this sequence are φ operators, the process continues in this way because, in this case, $i_{k-1} > i_{k-2} > \dots > i_r$ ($r \geq 1$). If i_r is again a σ operator we are in the same situation of the beginning of the proof. \square

Theorem 6 (Correspondence from $\text{MATCH}_{\lambda s_e}$ to $\text{MATCH}_{\lambda\sigma}$). *Let $\psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p) =_{\lambda s_e}^? (\underline{\mathbf{m}} b_1 \dots b_q)$ be a flexible-rigid equation in the λs_e -calculus, where X has atomic type. Then, for each equation generated by the rule **Exp-App** $_{\lambda s_e}$ there exists a corresponding equation in the $\lambda\sigma$ -calculus generated by the rule **Exp-App** $_{\lambda\sigma}$ for the equation*

$$T(\psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p)) =_{\lambda\sigma}^? T(\underline{\mathbf{m}} b_1 \dots b_q)$$

Proof. Suppose that one step of the rule **Exp-App** $_{\lambda s_e}$ generates the equation $X =_{\lambda s_e}^? (\underline{\mathbf{x}} H_1 \dots H_s)$. Then we have two cases to consider:

CASE 1: $\underline{\mathbf{x}} \in R_p$ where R_p is a subset of $\{i_1, \dots, i_p\}$ of the superscripts of the σ operator. Then there exists $1 \leq k \leq p$ such that $r = i_k$. This is possible only if the target type of a_k coincides with the (atomic) type of X . Therefore, since T is a type preserving operator, we have, as consequence of Lemma 14, that the k -th element of the substitution in the image of the given flexible-rigid equation also has target type equal to the type of X ; and this is exactly the condition required by the rule **Exp-App** $_{\lambda\sigma}$ to generate the equation $X =_{\lambda\sigma}^? (\underline{\mathbf{i}}_k Y_1 \dots Y_s)$, which corresponds to the given substitution in the λs_e -calculus.

CASE 2: $\underline{\mathbf{x}} \in R_i$, where $R_i = \bigcup_{k=0}^p$ if $i_k \geq m + p - k - \sum_{l=k+1}^p j_l > i_{k+1}$ then $\{m + p - k - \sum_{l=k+1}^p j_l\}$ else \emptyset , and $i_0 = \infty$ and $i_{p+1} = 0$. By hypotheses, there exists $0 \leq k \leq p$ such that $r = m + p - k - \sum_{l=k+1}^p j_l$.

If $k = p$ then $r = m$ and $m \leq i_p$. If $\psi_{i_p}^{j_p} = \sigma^{i_p}$ then $m < i_p$ because we already considered the case where r coincides with the superscript of a σ operator. From the definition of T , we have that the $\lambda\sigma$ -normal form of $T(\psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p))$ is a term of the form $X[\underline{\mathbf{1}}.\underline{\mathbf{2}}.\dots.\underline{\mathbf{i}}_p - \underline{\mathbf{1}}.c_1.\dots.c_r.\uparrow^s]$, where c_1, \dots, c_r are terms in $\lambda\sigma$ -normal form and $s \in \mathbb{N}$. Since T is a type preserving operator, we conclude that the m -th element in the substitution $\underline{\mathbf{1}}.\underline{\mathbf{2}}.\dots.\underline{\mathbf{i}}_p - \underline{\mathbf{1}}.c_1.\dots.c_r.\uparrow^s$ has target type equal to the type of X and hence the rule **Exp-App** $_{\lambda\sigma}$ will generate the substitution $X =_{\lambda\sigma}^? (\underline{\mathbf{m}} Y_1 \dots Y_s)$. If $\psi_{i_p}^{j_p} = \varphi_{i_p}^{j_p}$ then the $\lambda\sigma$ -normal form of $T(\psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p))$ is a term of the form $X[\underline{\mathbf{1}}.\underline{\mathbf{2}}.\dots.\underline{\mathbf{i}}_p - \underline{\mathbf{1}}.\underline{\mathbf{i}}_p.c_1.\dots.c_r.\uparrow^s]$, where c_1, \dots, c_r are terms in $\lambda\sigma$ -normal form and $s \in \mathbb{N}$. Since T is a type preserving operator, we conclude that the m -th element in the substitution $\underline{\mathbf{1}}.\underline{\mathbf{2}}.\dots.\underline{\mathbf{i}}_p - \underline{\mathbf{1}}.\underline{\mathbf{i}}_p.c_1.\dots.c_r.\uparrow^s$ has target type equal to the type of X and hence the rule **Exp-App** $_{\lambda\sigma}$ will generate the substitution $X =_{\lambda\sigma}^? (\underline{\mathbf{m}} Y_1 \dots Y_s)$.

If $k = 0$ then $r = m + p - \sum_{l=1}^p j_l$ and $r > i_1$, that is, $m + p - \sum_{l=1}^p j_l > i_1 \Rightarrow m > i_1 - p + \sum_{l=1}^p j_l$. We also have that the normal form of

$$T(\psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p))$$

is of the form $X[c_1 \dots c_{i_1} \cdot \uparrow^{i_1 - p + \sum_{l=1}^p j_l}]$ and hence the rule **Exp-App** $_{\lambda\sigma}$ generates the equation

$$X =_{\lambda\sigma}^? (\underline{m} - (\underline{i}_1 - \underline{p} + \sum_{l=1}^p \underline{j}_l) + \underline{i}_1 Y_1 \dots Y_s)$$

that is, $X =_{\lambda\sigma}^? (\underline{m} + \underline{p} - \sum_{l=1}^p \underline{j}_l Y_1 \dots Y_s) = (\underline{r} Y_1 \dots Y_s)$.

If $0 < k < p$ and since $i_{k+1} < r \leq i_k$, i.e., $i_{k+1} < m + p - k - \sum_{l=k+1}^p j_l \leq i_k$, we have that $m > i_{k+1} - p + k + \sum_{l=k+1}^p j_l$. The $\lambda\sigma$ -normal form of

$$T(\psi_{i_p}^{j_p} \dots \psi_{i_1}^{j_1}(X, a_1, \dots, a_p))$$

has shape of the form $X[c_1 \dots c_{i_1} \cdot \uparrow^s]$, where the r -th element of the substitution $c_1 \dots c_{i_1} \cdot \uparrow^s$ is a de Bruijn index and has target type equal to the type of X because T is a type preserving operator. Therefore, the rule **Exp-App** $_{\lambda\sigma}$ will generate the substitution $X =_{\lambda\sigma}^? (\underline{r} Y_1 \dots Y_s)$. \square

Example 3. Consider the unification problem

$$\varphi_1^4(((\varphi_7^3 X)\sigma^5 a)\sigma^3 b) =_{\lambda s_e}^? (\underline{6} b_1 \dots b_q)$$

The **Exp-App** $_{\lambda s_e}$ rule generates the equation $X =_{\lambda s_e}^? (\underline{4} H_1 \dots H_q)$. The $\lambda\sigma$ -normal form of $T(\varphi_1^4(((\varphi_7^3 X)\sigma^5 a)\sigma^3 b))$ is computed according to the following steps:

$$\begin{aligned} & T(\varphi_1^4(((\varphi_7^3 X)\sigma^5 a)\sigma^3 b)) = \\ & T(((\varphi_7^3 X)\sigma^5 a)\sigma^3 b)[\underline{1} \cdot \uparrow^4] = \\ & T((\varphi_7^3 X)\sigma^5 a)[\underline{1} \cdot \underline{2} \cdot T(b)[\uparrow^2] \cdot \uparrow^2][\underline{1} \cdot \uparrow^4] \rightarrow_{\sigma}^* \\ & T((\varphi_7^3 X)\sigma^5 a)[\underline{1} \cdot \underline{5} \cdot T(b)[\uparrow^5] \cdot \uparrow^5] = \\ & T(\varphi_7^3 X)[\underline{1} \cdot \underline{2} \cdot \underline{3} \cdot \underline{4} \cdot T(a)[\uparrow^4] \cdot \uparrow^4][\underline{1} \cdot \underline{5} \cdot T(b)[\uparrow^5] \cdot \uparrow^5] \rightarrow_{\sigma}^* \\ & T(\varphi_7^3 X)[\underline{1} \cdot \underline{5} \cdot T(b)[\uparrow^5] \cdot \underline{6} \cdot T(a)[\uparrow^6] \cdot \uparrow^6] = \\ & X[\underline{1} \cdot \underline{2} \cdot \underline{3} \cdot \underline{4} \cdot \underline{5} \cdot \underline{6} \cdot \underline{7} \cdot \uparrow^9][\underline{1} \cdot \underline{5} \cdot T(b)[\uparrow^5] \cdot \underline{6} \cdot T(a)[\uparrow^6] \cdot \uparrow^6] \rightarrow_{\sigma}^* \\ & X[\underline{1} \cdot \underline{5} \cdot T(b)[\uparrow^5] \cdot \underline{6} \cdot T(a)[\uparrow^6] \cdot \underline{7} \cdot \underline{8} \cdot \uparrow^{10}] \end{aligned}$$

The rule **Exp-App** $_{\lambda\sigma}$ generates the corresponding equation $X =_{\lambda\sigma}^? \underline{4}(Y_1 \dots Y_q)$ which corresponds to the selection of the de Bruijn index $\underline{6}$ inside the substitution $\underline{1} \cdot \underline{5} \cdot T(b)[\uparrow^5] \cdot \underline{6} \cdot T(a)[\uparrow^6] \cdot \underline{7} \cdot \underline{8} \cdot \uparrow^{10}$.

The surprising result is that the other direction of the previous theorem is also true, that is, if an application of the rule **Exp-App** $_{\lambda\sigma}$ generates an equation in the $\lambda\sigma$ -calculus then the application of the rule **Exp-App** $_{\lambda s_e}$ to the translation of this equation to the λs_e -calculus generates a corresponding equation. In order to prove this result, let us first define a partial translation operator, which is able to express the simultaneous substitutions of the $\lambda\sigma$ -calculus into the λs_e -calculus.

Definition 22. The operator $L : \Lambda_{\lambda\sigma\text{-terms}} \rightarrow \Lambda_{\lambda s_e}$ is defined inductively as:

$$\begin{aligned} L(X) &= X \\ L(\underline{1}[\uparrow^{m-1}]) &= \underline{m}, \text{ where } m \in \mathbb{N} \end{aligned}$$

$$\begin{aligned}
L(a\ b) &= L(a)\ L(b) \\
L(\lambda.a) &= \lambda.L(a) \\
L(a[a_1.a_2.\dots.a_p.\ \uparrow^n]) &= \sigma^1.\dots.\sigma^{p-1}\sigma^p\varphi_p^{n+1}(L(a), L(a_p), L(a_{p-1}), \dots, L(a_2), L(a_1)), \\
&\text{where } a_1.a_2.\dots.a_p.\ \uparrow^n \text{ is a substitution in } \lambda\sigma\text{-normal form, and } n, p \geq 0.
\end{aligned}$$

It is important to note that the operator L is partial, since there is no translation for substitutions. In addition, the substitutions s that appear in terms of the form $a[s]$ must be in normal $\lambda\sigma$ -form. This is not a severe restriction because for a given unification problem which is not in $\lambda\sigma$ -normal form, one can always apply the rule **Normalise** and get an equivalent unification problem which is normalised. The operator L also preserves types as is proved by the following proposition.

Proposition 8 (Preservation of types by L). *Let Γ be a context, A a type and a a term in the language of the $\lambda\sigma$ -calculus such that $\Gamma \vdash a : A$. Then $\Gamma \vdash L(a) : A$.*

Proof. Let $a \in \Lambda_\sigma$ such that $\Gamma \vdash a : A$. The proof is by structural induction on a :

- If $a_A^\Gamma = X_A^\Gamma$ then $L(X_A^\Gamma) = X_A^\Gamma = L(a)_A^\Gamma$.
- If $a_A^\Gamma = (\underline{1}[\uparrow^{n-1}])_A^\Gamma$ then the n -th element of Γ must be the type A , therefore $L((\underline{1}[\uparrow^{n-1}])_A^\Gamma) = \underline{a}_A^\Gamma = L(a)_A^\Gamma$.
- If $a_A^\Gamma = (b\ c)_A^\Gamma$ then, by the type rules of the $\lambda\sigma$ -calculus, there exists a type B such that $b_{B \rightarrow A}^\Gamma$, and hence $(b_{B \rightarrow A}^\Gamma\ c_B^\Gamma)_A^\Gamma$. Therefore, $L(a_A^\Gamma) = L(b_{B \rightarrow A}^\Gamma\ c_B^\Gamma) \stackrel{def}{=} L(b_{A \rightarrow B}^\Gamma)\ L(c_A^\Gamma) \stackrel{HI}{=} L(b)_{A \rightarrow B}^\Gamma\ L(c)_A^\Gamma = (L(b)\ L(c))_A^\Gamma = L(a)_A^\Gamma$.
- If $a_A^\Gamma = (\lambda_B.b)_A^\Gamma$ then $b_C^{A,\Gamma}$ and $L(\lambda_A.a_B^\Gamma) = \lambda_A.L(a)_B^\Gamma$
- $L(a_{A_1 \dots A_p}^{A_1 \dots A_p, \Gamma \geq n+1}[(a_1_{A_1}^\Gamma \dots a_p_{A_p}^\Gamma \cdot \uparrow_{\Gamma \geq n+1}^\Gamma)_{A_1 \dots A_p, \Gamma \geq n+1}^\Gamma])_A^\Gamma = \sigma^1.\dots.\sigma^{p-1}\sigma^p(\varphi_p^{n+1}(L(a)_{A_1 \dots A_p, \Gamma \geq n+1}^{A_1 \dots A_p, \Gamma \geq n+1})_{A_1 \dots A_p, \Gamma}^\Gamma, L(a_p)_{A_p}^\Gamma, L(a_{p-1})_{A_{p-1}}^\Gamma, \dots, L(a_2)_{A_2}^\Gamma, L(a_1)_{A_1}^\Gamma)_A^\Gamma$

□

Theorem 7 (Correspondence from $\text{MATCH}_{\lambda\sigma}$ to $\text{MATCH}_{\lambda s_e}$). *Let $X[a_1.\dots.a_p.\ \uparrow^n] \stackrel{?}{=}_{\lambda\sigma} (\underline{m}\ b_1 \dots b_q)$ be a flexible-rigid equation in the $\lambda\sigma$ -calculus, where X has atomic type and $a_1.\dots.a_p.\ \uparrow^n$ is a $\lambda\sigma$ -normal substitution. Then, for each equation generated by the rule **Exp-App** $_{\lambda\sigma}$ there exists a corresponding equation in the λs_e -calculus generated by the rule **Exp-App** $_{\lambda s_e}$ for the equation*

$$L(X[a_1.\dots.a_p.\ \uparrow^n]) \stackrel{?}{=}_{\lambda s_e} (\underline{m}\ L(b_1) \dots L(b_q))$$

Proof. Suppose that the equation $X \stackrel{?}{=}_{\lambda\sigma} \underline{x}H_1 \dots H_s$ is generated after an application of **Exp-App** $_{\lambda\sigma}$ to the equation $X[a_1.\dots.a_p.\ \uparrow^n] \stackrel{?}{=}_{\lambda\sigma} (\underline{m}\ b_1 \dots b_q)$. If $r \in \{1, \dots, p\}$ then the target type of a_r must be equal to the type of X . Since the operator L preserves types (Proposition 8), we have that the target type of $L(a_r)$ is also equal to the type of X and then, for the equation

$$\sigma^1 \dots \sigma^{p-1} \sigma^p \varphi_p^{n+1}(X, L(a_p), L(a_{p-1}), \dots, L(a_1)) \stackrel{?}{=}_{\lambda s_e} \underline{m}\ L(b_1) \dots L(b_q)$$

we have that $\mathbf{Exp-App}_{\lambda s_e}$ will generate the projection $X =_{\lambda s_e}^? \underline{\mathbf{r}} Y_1 \dots Y_s$.

If $r = m - n + p$ then $m > n$ and this corresponds to an imitation step, i.e., $\underline{\mathbf{r}}$ is such that, after the normalisation step, its normal form corresponds to $\underline{\mathbf{m}}$. Of course, the target type of $\underline{\mathbf{r}}$ must be equal to the type of X . It is easy to verify that the sole possible imitation generated by $\mathbf{Exp-App}_{\lambda s_e}$ is $X =_{\lambda s_e}^? \underline{\mathbf{m} - \mathbf{n} + \mathbf{p}} Y_1 \dots Y_s$ which corresponds to the equation generated in the $\lambda\sigma$ -calculus. \square

The comparison of the $\lambda\sigma$ - and λs_e -styles of unification is an important step in understanding the relation among different styles of explicit substitutions as well as the benefits which can be obtained including HOU via explicit substitutions in computational systems such as programming languages and proof assistants.

The results of this section allow us to give a general overview of the unification process in both $\lambda\sigma$ - and λs_e -styles of explicit substitutions. These results can be formalised as follows.

Theorem 8 (Correspondence between $\mathbf{MATCH}_{\lambda s_e}$ and $\mathbf{MATCH}_{\lambda\sigma}$). *Let P be a unification problem in the simply typed λ -calculus, and P_ξ its precooking translation to the ξ -calculus of explicit substitutions, where $\xi \in \{\lambda\sigma, \lambda s_e\}$. Then $P_{\lambda\sigma}$ is unifiable if and only if $P_{\lambda s_e}$ is unifiable. Moreover, whenever unifiers exist, they are associated in the sense of Definition 20.*

Proof. This is a corollary of Theorems 6 and 7. \square

The previous result can be extended to unification problems of the language of the explicit substitutions calculus that are not necessarily generated by unification problems from the pure λ -calculus. In fact, unification problems in the $\lambda\sigma$ -calculus are conjunctions of equations between terms and, since there is no $\lambda\sigma$ -unification rules that apply to substitutions, it is always possible to translate a unification problem from the $\lambda\sigma$ -calculus to the λs_e -calculus by the given translation operator.

Corollary 1. *Let $P_{\lambda\sigma}$ be a unification problem in the $\lambda\sigma$ -calculus of explicit substitutions, and $L(P_{\lambda\sigma})$ its translation to the λs_e -calculus of explicit substitutions. Then $P_{\lambda\sigma}$ is unifiable if and only if $L(P_{\lambda\sigma})$ is unifiable. Moreover, whenever unifiers exist, they are associated in the sense of Definition 20.*

7 Conclusions and Future Work

This paper focuses on the formalisation of Huet's higher order unification (HOU) and its relation to HOU based on explicit substitutions calculi. For this purpose we introduced a structural notation called *unification tree* which allows a clear presentation of Huet's algorithm in de Bruijn notation. This notation includes a tree structure and subscripts used for interconnecting in a natural way unification problems derived from applications of the procedures `SIMPL` and `MATCH` of Huet's algorithm. In the context of this paper, unification trees were applied to

λ -terms with de Bruijn indexes, but they can also be applied to λ -calculus with names without major modifications.

With the above formalisation of Huet’s HOU in the λ -calculus with de Bruijn notation, we could formalise the relation between Huet’s HOU and HOU methods based on explicit substitutions. We compared the classical method of Huet for HOU in the simply typed λ -calculus [Hue75] and the $\lambda\sigma$ -HOU algorithm presented by Dowek, Hardin and Kirchner [DHK00]. To do so, we translated Huet’s algorithm to the language of the simply typed λ -calculus in de Bruijn notation and, using particular strategies of the inference rules of the $\lambda\sigma$ -HOU algorithm, we described the counterpart of the procedures SIMPL and MATCH, called $\text{SIMPL}_{\lambda\sigma}$ and $\text{MATCH}_{\lambda\sigma}$, respectively. We proved that there exists a correspondence between the substitutions generated by Huet’s algorithm and the graftings generated by the $\lambda\sigma$ -HOU algorithm for unification problems which are in the image of the precooking translation. Nevertheless, if we drop η -conversion, some imitation substitutions are simulated by elements in R_p which are supposed to be projections. In addition, our correspondence allows us to formalise the fact that the method of HOU via explicit substitution à la [DHK00] is a generalisation of Huet’s HOU algorithm. This comparison was easily extended to the λs_e -HOU algorithm [ARK01]. We compared the $\lambda\sigma$ - and the λs_e -styles of unification by translating λs_e -terms into equivalent $\lambda\sigma$ -terms and vice-versa. In this comparison, we concluded that the $\lambda\sigma$ - and the λs_e -HOU algorithms generate associated graftings. Therefore, although the language of the $\lambda\sigma$ -calculus can be seen as a refinement of the language of the λs_e -calculus, because every λs_e -expression can be translated into a $\lambda\sigma$ -expression but not vice-versa, the λs_e -calculus is powerful enough to solve exactly the same problems that are solvable in the $\lambda\sigma$ -calculus using the translation operators. In this sense, several advantages of using the λs_e -calculus can be pointed out:

- The λs_e -calculus has a simple grammar that uses just an updating and a substitution operators, namely φ_k^i and σ^j , respectively.
- The λs_e -calculus is closer to the pure λ -calculus and uses all de Bruijn indexes, instead of the codification used in the $\lambda\sigma$ -calculus. Using all de Bruijn indexes is easier for human reading.
- The λs_e -calculus is expressive enough to solve all the unification problems that are solvable in the $\lambda\sigma$ -calculus, even the ones that are not originated from the pure λ -calculus.

We believe this formalisation is worthwhile for having a good understanding of the novel alternative of HOU via general explicit substitutions calculi and contributes to current research related to the design and implementation of higher-order computational environments.

Acknowledgments. We would like to thank Claude Kirchner for some useful comments on this work.

References

- [ACCL91] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit Substitutions. *J. of Func. Programming*, 1(4):375–416, 1991.
- [ARK01] M. Ayala-Rincón and F. Kamareddine. Unification via the λ_{s_e} -Style of Explicit Substitution. *The Logical Journal of the Interest Group in Pure and Applied Logics*, 9(4):489–523, 2001.
- [ARK03] M. Ayala-Rincón and F. Kamareddine. On Applying the λ_{s_e} -Style of Unification for Simply-Typed Higher Order Unification in the Pure lambda Calculus. Special Issue of WoLLIC 2001 selected papers, John T. Baldwin, Ruy J. G. B. de Queiroz and Edward H. Haeusler Eds. *Matemática Contemporânea*, 24:1–22, 2003.
- [Bar84] H. P. Barendregt. *The Lambda Calculus : Its Syntax and Semantics (revised edition)*. North Holland, 1984.
- [Bor95] 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 *LNCS*, pages 363–368. Springer Verlag, 1995.
- [dB72] N.G. de Bruijn. Lambda-Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem. *Indag. Mat.*, 34(5):381–392, 1972.
- [DHK00] G. Dowek, T. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions. *Information and Computation*, 157:183–235, 2000.
- [Dow01] G. Dowek. Higher-Order Unification and Matching. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 16, pages 1009–1062. MIT press and Elsevier, 2001.
- [Hue75] G. Huet. A Unification Algorithm for Typed λ -Calculus. *TCS*, 1:27–57, 1975.
- [Hue02] G. Huet. Higher Order Unification 30 Years Later. In V. A. Carreño, C. A. Muñoz, and S. Tahar, editors, *Theorem Proving in Higher Order Logics - TPHOLS 2002*, volume 2410 of *LNCS*, pages 3–12. Springer, 2002.
- [KN96] F. Kamareddine and R. P. Nederpelt. A useful λ -notation. *TCS*, 155:85–109, 1996.
- [KR95] F. Kamareddine and A. Ríos. A λ -calculus à la de Bruijn with Explicit Substitutions. In *Proc. of PLILP'95*, volume 982 of *LNCS*, pages 45–62. Springer, 1995.
- [KR97] F. Kamareddine and A. Ríos. Extending a λ -calculus with Explicit Substitution which Preserves Strong Normalisation into a Confluent Calculus on Open Terms. *Journal of Functional Programming*, 7:395–420, 1997.
- [NGdV94] R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer. *Selected papers on Automath*. North-Holland, 1994.
- [Río93] A. Ríos. *Contributions à l'étude de λ -calculs avec des substitutions explicites*. Thèse de doctorat, Université Paris VII, 1993.

A Appendix: Unification in the $\lambda\sigma$ -calculus

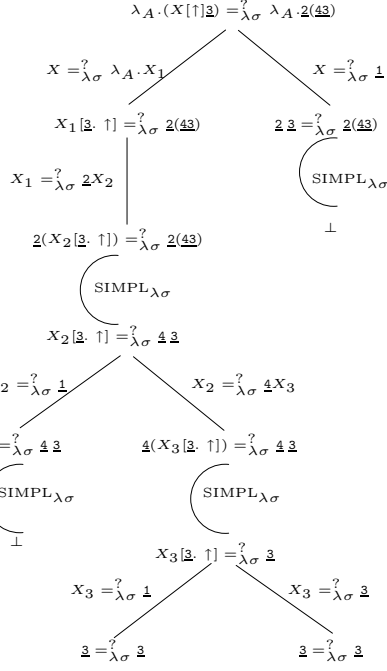


Fig. 6. $\lambda\sigma$ -matching tree (without **Eta**-conversion).

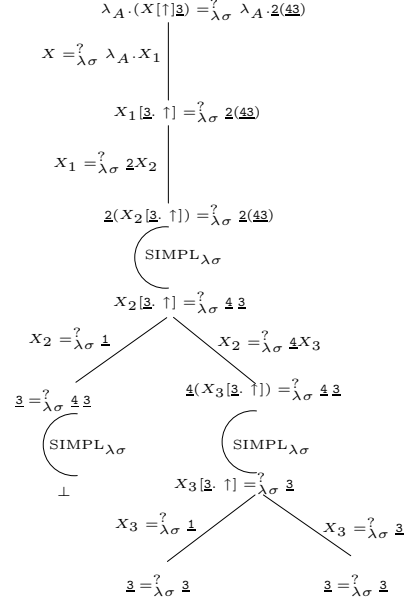


Fig. 7. $\lambda\sigma$ -matching tree (with **Eta**-conversion).

Consider the unification problem $P = \{\lambda_A.(X \ z) =?_{\lambda\sigma} \lambda_A.2(4 \ z)\}$ over the context $\Gamma = A \rightarrow B.A.A \rightarrow A.nil$ and $\Gamma \vdash X : A \rightarrow B$. The precooked translation of P is given by $P_F = \{\lambda_A.(X[\uparrow] \ z) =?_{\lambda\sigma} \lambda_A.2(4 \ z)\}$. Note that, for the sake of clarity, we use \underline{n} to denote $\underline{1}[\uparrow^{n-1}]$, i.e., we avoid writing the usual codification of de Bruijn indexes in the $\lambda\sigma$ -calculus. The algorithm (without **Eta**-conversion) runs as follows:

$$\lambda_A.(X[\uparrow] \ z) =?_{\lambda\sigma} \lambda_A.2(4 \ z)$$

→ **Dec- λ**

$$(X[\uparrow] \ z) =?_{\lambda\sigma} 2(4 \ z)$$

→ **Exp**

$$\exists X_1((X[\uparrow] \ z) =?_{\lambda\sigma} 2(4 \ z) \wedge X =?_{\lambda\sigma} \underline{1}) \vee (X[\uparrow] \ z) =?_{\lambda\sigma} 2(4 \ z) \wedge X =?_{\lambda\sigma} \lambda_A.X_1)$$

→ **Replace**

$$\exists X_1((2 \ z) =?_{\lambda\sigma} 2(4 \ z) \wedge X =?_{\lambda\sigma} \underline{1}) \vee ((\lambda_A.X_1)[\uparrow] \ z) =?_{\lambda\sigma} 2(4 \ z) \wedge X =?_{\lambda\sigma} \lambda_A.X_1)$$

→ **Normalise**

$$\exists X_1((2 \ z) =?_{\lambda\sigma} 2(4 \ z) \wedge X =?_{\lambda\sigma} \underline{1}) \vee (X_1[z. \uparrow] =?_{\lambda\sigma} 2(4 \ z) \wedge X =?_{\lambda\sigma} \lambda_A.X_1)$$

→ **Dec-App**

$$\exists X_1((z) =?_{\lambda\sigma} (4 \ z) \wedge X =?_{\lambda\sigma} \underline{1}) \vee (X_1[z. \uparrow] =?_{\lambda\sigma} 2(4 \ z) \wedge X =?_{\lambda\sigma} \lambda_A.X_1)$$

$\exists X_1 \exists X_2 \exists X_3 (\underline{\exists} =_{\lambda\sigma}^? \underline{\exists} \wedge X =_{\lambda\sigma}^? \lambda_A.(\underline{2} (\underline{4} \underline{\exists})) \wedge X_1 =_{\lambda\sigma}^? \underline{2} (\underline{4} \underline{\exists}) \wedge X_2 =_{\lambda\sigma}^? \underline{4} \underline{\exists} \wedge X_3 =_{\lambda\sigma}^? \underline{\exists})$

Then we have found two grafting solutions for the precooked problem

$$\lambda_A.(X[\uparrow] \underline{\exists}) =_{\lambda\sigma}^? \lambda_A.\underline{2}(\underline{4} \underline{\exists})$$

given by $X \mapsto \lambda_A.(\underline{2} (\underline{4} \underline{1}))$ and $X \mapsto \lambda_A.(\underline{2} (\underline{4} \underline{\exists}))$. In fact, both

$$\lambda_A.(\lambda_A.\underline{2} (\underline{4} \underline{1}))[\uparrow] \underline{\exists}$$

and

$$\lambda_A.(\lambda_A.\underline{2} (\underline{4} \underline{\exists}))[\uparrow] \underline{\exists}$$

reduce to $\lambda_A.\underline{2}(\underline{4} \underline{\exists})$. From these grafting solutions we can extract the solutions that are the substitutions $X/\lambda_A.(\underline{2} (\underline{4} \underline{1}))$ and $X/\lambda_A.(\underline{2} (\underline{4} \underline{\exists}))$ which are those computed by Huet's algorithm.