Unification via the λs_e-Style of Explicit Substitutions

MAURICIO AYALA-RINCÓN[1] , *Departamento de Matemática, Universidade de Brasília, 70910-900 Brasília D.F., Brasil.*
E-mail: ayala@mat.unb.br

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

Abstract

A unification method based on the λs_e-style of explicit substitution is proposed. This method together with appropriate translations, provide a Higher Order Unification (HOU) procedure for the pure λ-calculus. Our method is influenced by the treatment introduced by Dowek, Hardin and Kirchner using the $\lambda\sigma$-style of explicit substitution. Correctness and completeness properties of the proposed λs_e-unification method are shown and its advantages, inherited from the qualities of the λs_e-calculus, are pointed out. Our method needs only one sort of objects: terms. And in contrast to the HOU approach based on the $\lambda\sigma$-calculus, it avoids the use of substitution objects. This makes our method closer to the syntax of the λ-calculus. Furthermore, detection of redices depends on the search for solutions of simple arithmetic constraints which makes our method more operational than the one based on the $\lambda\sigma$-style of explicit substitution.

Keywords: Higher order unification, explicit substitution, lambda-calculi.

1 Introduction

After Robinson's successful introduction of his well-known first order *Resolution Principle* based on substitution, unification and resolution [41], much work has been done in order to formalize these basic notions in other settings. Such extensions are essential for amongst other things, automated deduction in higher order logics. Mechanizations of second order and full higher order unification were initially formulated in [38] and [23]. In [22] Huet successfully formulated a practical higher order unification method, specifically for the typed λ-calculus. Since then several Higher Order Unification (HOU) approaches have been developed and used in practical languages and theorem provers such as λprolog and Isabelle [35, 37]. In most of these approaches, the notion of substitution plays an important role. The importance of the notion of substitution led to an explosion of work on making substitutions explicit in recent years [1, 7, 24, 26, 19, 9, 21]. Moreover, a number of works have been devoted to establishing the usefulness of explicit substitution to automated deduction and theorem proving [32, 34], to proof theory [43], to programming languages [29, 6, 8] and to HOU [16]. The latter paper [16] shows that in the HOU framework, if substitution was made

[1]Work carried out during a year study leave visit of this author at the ULTRA Group, CEE, Heriot-Watt University, Edinburgh, Scotland. This leave was funded by CAPES (BEX0384/99-2) Brazilian Foundation and partially supported by EPSRC grant numbers GR/L36963 and GR/L15685.

explicit, many benefits can be obtained in computation. In particular, [16] presented a HOU method based on the $\lambda\sigma$-calculus which was proved useful for deduction in the typed λ-calculus and subsequently generalized for treating higher order equational unification problems [30] and restricted for the case of higher order patterns [17]. The novelity of this method is that higher order unification problems in the language of the pure λ-calculus can be solved by first order unification over the language of the $\lambda\sigma$-calculus once they have been translated or *pre-cooked* into the language of the $\lambda\sigma$-calculus. Then, solutions can be translated *back* into the range of the *pre-cooking* translation and subsequently to solutions of the original problems. In this paper, we develop a unification method based on the λs_e-style of explicit substitution which jointly with adequate *pre-cooking* and *back* translations between the languages of the λ-calculus and the λs_e-calculus (cf. [2]) give a HOU procedure, which takes advantage of the qualities of the λs_e calculus. In particular, λs_e-unification avoids the use of two different sorts of objects as in the $\lambda\sigma$-calculus. Moreover, the decidability of the application of our unification rules (i.e., the detection of redices) depends on the search for natural solutions of simple arithmetic constraints. Since arithmetic decision mechanisms are built-in in most of the computational languages and automated assistants systems, this makes λs_e-HOU more operational than the $\lambda\sigma$-HOU.

1.1 Higher order unification

Higher order objects arise naturally in many fields of computer science. For example, in the context of implementation of functional languages it is necessary to develop mechanisms for the treatment of higher order functions. Take for instance, the rewriting system that specifies the well-known MAP function, which applies a function to all the elements of a list: $\text{MAP}(f, \text{NIL}) \rightarrow \text{NIL}$; $\text{MAP}(f, \text{CONS}(x, l)) \rightarrow \text{CONS}(f(x), \text{MAP}(f, l))$, where NIL and CONS are the usual LISP empty list and constructor list function. Observe that f appears both as a variable and as a functional symbol. From the point of view of first oder rewriting, it is not possible to manipulate this kind of objects; in fact, for simple rewrite based deduction processes such as one-step reduction or critical pair deduction, first order matching and unification, respectively, do not apply. The solution of these problems, at least in the rewriting context, is the λ-calculus. Rewriting could be performed modulo the rules of the λ-calculus or combining specifications with the rules of the λ-calculus.

The function MAP is a typical example of a second-order function, but functions of third-order or above have practical interest too. In [36], useful third- until sixth-order functions were presented in the context of combinator parsing.

A simple example of a HOU problem is to search for solutions for the equality $F(f(a)) = f(F(a))$. The identity function $\{F/\lambda_x.x\}$ is a solution, and so are the functions $\{F(x)/f^n(x) \mid n \in \mathbb{N}\}$.

HOU is essential in higher order automated reasoning, where it has formed the basis for generalizations of the Resolution Principle in second-order logic.

Huet's work [22] was relevant because he realized that to generalize Robinson's first order Resolution Principle [41] to higher order theories, it is useful to verify the existence of unifiers without computing them explicitly. Huet's algorithm is a semi-decision one that may never stop when the input unification problem has no unifiers, but when the problem has a solution it implicitly allows one to recover any unifier always. This completeness is an essential feature of Huet's algorithm. Unification for second-order logic was proved undecidable in general by Goldfarb [20]. Goldfarb's proof is based on a reduction from Hilbert's Tenth Problem. This result shows that there are arbitrary higher order theories where unification is undecidable, but there exist particular higher order languages of practical interest that have a decidable unification problem. In particular, for the second-order case, unification is decidable, when the language is restricted to monadic functions [18]. Another problem of HOU is that the notion of most general unifier does not apply and that a more complex notion than complete sets of unifiers is necessary. Huet has shown that equations of the form $(\lambda_x.F\ \ a) =^? (\lambda_x.G\ \ b)$ (called *flex-flex*) of third-order may not have minimal complete sets of unifiers and that there may exist an infinite chain of unifiers, one more general than the other, without having a most general one (section 4.1 in [39]).

For a very simple presentation of HOU see [42] and for a detailed introduction in the context of declarative programming see [39].

1.2 Contribution of this work

The $\lambda\sigma$-calculus [1] introduces two different sets of entities, one for terms and one for substitutions. The λs_e-calculus [27] insists on remaining closer to the λ-calculus and uses a philosophy started with de Bruijn in his system *AUTOMATH* and elaborated extensively through the new item notation [25]. The philosophy states that terms of the λ-calculus are either application terms such as a function applied to an argument, abstraction terms such as a function. Substitution or updating are made explicit in item notation, by introducing substitution terms and updating terms. The advantages of this philosophy are listed in [25] and include remaining as close as possible to the familiar λ-calculus. Therefore, we propose to study HOU in the λs_e-style of explicit substitution, which makes our approach closer to the syntax of the λ-calculus than that of the $\lambda\sigma$-approach in that we avoid the use of two different sorts of objects. We establish the following properties of λs_e-unification:

1. Correctness: If P and P' are unification problems such that P reduces to P' then every unifier of P' is a unifier of P.

2. Completeness: If P and P' are unification problems such that P reduces to P' then every unifier of P is a unifier of P'.

3. The search for unification redices and detection of *flex-flex* (i.e. implicitly solvable) equations is simpler in our approach than in the $\lambda\sigma$-approach.

In Section 2, we introduce the basic machinery. In Section 3, we recall the $\lambda\sigma$- and λs_e-calculi and establish λs_e-normalisation properties needed for unification. In Section 4, we recall the unification approach in the $\lambda\sigma$-calculus. In Section 5, we present our λs_e-unification method. In Section 6, we provide some arithmetic properties of the λs_e-unification rules. In section 7, we discuss the application of our unification method for higher order unification and conclude.

A preliminary version of this work was presented in [3].

2 Preliminaries

We assume familiarity with λ-calculus (cf. [5]) and the notion of term algebra $\mathcal{T}(\mathcal{F},\mathcal{X})$ built on a (countable) set of variables $\mathcal{X}$ and a set of operators $\mathcal{F}$. Variables in $\mathcal{X}$ are denoted by $X, Y, ...$ and for a term $a \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $var(a)$ denotes the set of variables occurring in a. In every calculus we consider, we use $a, b, c, \ldots$ to range over terms.

Additionally, we assume familiarity with basic notions of rewriting as presented in [4]. In particular, for a *reduction relation* R over a set A, we denote with $\overset{=}{\rightarrow}_R$ the **reflexive closure** of R, with $\rightarrow_R^*$ or just $\rightarrow^*$ the **reflexive and transitive closure** of R and with $\rightarrow_R^+$ or just $\rightarrow^+$ the **transitive closure** of R. When $a \rightarrow^* b$ we say that there exists a **derivation** from a to b. By $a \rightarrow^n b$, we mean that the derivation consists of n steps of reduction and call n the **length of the derivation**. Syntactical identity is denoted by $a = b$. For a reduction relation R over A, $(A, \rightarrow_R)$, we use the standard definitions of **(local) confluence** or (weakly) Church Rosser **(W)CR**, normal forms and **strong** and **weak normalization/termination SN** and **WN**.

A **valuation** is a mapping from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F},\mathcal{X})$. The homeomorphic extension of a valuation, θ, from its domain $\mathcal{X}$ to the domain $\mathcal{T}(\mathcal{F},\mathcal{X})$ is called the **grafting** of θ. As usual, valuations and their corresponding graftings are denoted by the same Greek letter. The application of a valuation θ or its corresponding grafting to a term $a \in \mathcal{T}(\mathcal{F},\mathcal{X})$ will be written in postfix notation $a\theta$. The **domain** of a grafting θ, is defined by $Dom(\theta) = \{X \mid X\theta \neq X, X \in \mathcal{X}\}$. Its **range**, is defined by $Ran(\theta) = \cup_{X \in Dom(\theta)} var(X\theta)$. We let $var(\theta) = Dom(\theta) \cup Ran(\theta)$. For explicit representations of a valuation and its corresponding grafting θ, we use the notation $\theta = \{X \mapsto X\theta \mid X \in Dom(\theta)\}$. Note that the notion of grafting, usually called first order substitution, corresponds to simple syntactic substitution without renaming.

2.1 The λ-calculus with names

In this section, we present the λ-calculus with names emphasizing the role of unification variables and substitutions. Let $\mathcal{V}$ be a (countable) set of variables (different from the ones in $\mathcal{X}$) denoted by lowercase last letters of the Roman alphabet $x, y, ...$

Definition 2.1 Terms $\Lambda(\mathcal{V})$, of the **λ-calculus with names** are inductively defined by:

$a ::= x \mid (a \ \ a) \mid \lambda_x.a$, where $x \in \mathcal{V}$.

$\lambda_x.a$ and $(a \ \ b)$ are called *abstraction* and *application* terms, respectively.

An abstraction $\lambda_x.a$ represents a function of parameter x, whose body is a. Its application to an argument b, $(\lambda_x.a \ \ b)$, returns the value of a, where the formal parameter x is replaced with the argument b. This replacement of formal parameters with arguments is known as **β-reduction**. In the first order context of the term algebra $\mathcal{T}(\{\lambda_x.\_ \mid x \in \mathcal{V}\} \cup \{(\_ \ \ \_)\}, \mathcal{V})$ and its first order substitution or grafting, β-reduction would be defined by $(\lambda_x.a \ \ b) \rightarrow a\{x \mapsto b\}$.

But in this context some problems arise making it necessary to rename bound variables, i.e. executing **α-conversion**. In fact, firstly suppose $\theta = \{x \mapsto b\}$. There are no semantic differences between the abstractions $\lambda_x.x$ and $\lambda_z.z$; both abstractions represent the identity function. But $(\lambda_x.x)\theta = \lambda_x.b$ and $(\lambda_z.z)\theta = \lambda_z.z$ are different. Secondly, suppose $\theta = \{x \mapsto y\}$. $(\lambda_y.x)\theta = \lambda_y.y$ and $(\lambda_z.x)\theta = \lambda_z.y$, thus a capture is possible. Consequently, β-reduction, should be defined in a way that takes care of renaming bound variables when necessary to avoid harmful capture of variables.

Most of the literature on unification and on the λ-calculus considers substitution as an atomic operation leaving implicit the computational steps needed to effectively perform computational operations based on substitution such as matching and unification. In any real higher order deductive system, the substitution required by basic operations such as β-reduction should be implemented via smaller operations. Explicit substitution is an appropriate formalism for reasoning about the operations involved in real implementations of substitution. Since explicit substitution is closer to real implementations than to the classic theory of the λ-calculus, it provides a more accurate theoretical model to analyze essential properties of real systems (such as termination, confluence, correctness, completeness, etc.) as well as their time/space complexity. For further details of the importance of explicit substitution see [29].

We denote by $\alpha^V(a)$ the α-conversion of a resulting by renaming the variables in $V \subseteq \mathcal{V}$ occurring at $a \in \Lambda(\mathcal{V})$ with *fresh* variables (i.e. variables not yet used).

Definition 2.2 Let $V \subset \mathcal{V}$. The **renaming application** α^V is defined by structural induction on $\Lambda(\mathcal{V})$ as follows:

1) $\alpha^V(x) = x$
2) $\alpha^V((a\ \ b)) = (\alpha^V(a)\ \ \alpha^V(b))$
3) $\alpha^V(\lambda_x.a) = \begin{cases} \lambda_x.\alpha^V(a), & \text{if } x \notin V \\ \lambda_y.(\alpha^V(a))\{x \mapsto y\}, & \text{if } x \in V \text{ where } y \text{ is a fresh variable} \\ & \text{neither occurring in } a \text{ nor in } V \end{cases}$

Now we are able to define the usual substitution operation.

Definition 2.3 For a valuation (over $\mathcal{V}$) $\theta = \{x_1 \mapsto a_1, \ldots, x_n \mapsto a_n\}$, the **substitution** extending θ, written θ^{ext}, is defined by induction structural as follows:

1) $\theta^{ext}(x) = x\theta$ if $x \in \mathcal{V}$
2) $\theta^{ext}((a\ \ b)) = (\theta^{ext}(a)\ \ \theta^{ext}(b))$
3) $\theta^{ext}(\lambda_x.a) = \lambda_z.\theta^{ext}((\alpha^{var(\theta)\cup\{x\}\cup\{z\}}(a))\{x/z\})$, where z is a fresh variable; i.e., $z \notin var(\theta)$ and z does not occur in a.

The substitution θ^{ext} is explicitly denoted by $\theta^{ext} = \{x_1/a_1, \ldots, x_n/a_n\}$.

When no confusion arises we use θ to denote both a valuation θ and its corresponding substitution. In this section, in order to emphasize the difference between valuations and substitutions, we use prefixed notation $\theta(a)$ for the application of substitution θ to term a while keeping $a\theta$ for the application of grafting.

The third item of Definition 2.3 means that bound α-conversion or variable renaming should be performed before applying the substitution in the body of an abstraction. The grafting of a fresh variable avoids the possibility of capture. Again it is very important to remark that the renaming application selects fresh variables that are not used previously in the process. Additionally, observe that since fresh variables are selected randomly, the result of the application of a substitution can be conceived as a class of equivalence terms rather than only one.

Definition 2.4 **β-reduction** is the rewriting relation defined by the rewrite rule (β) and **η-reduction** is the rewriting relation defined by the rewrite rule (η), where:

$(\beta)\ \ (\lambda_x.a\ \ b) \to \{x/b\}^{ext}(a)$ and $(\eta)\ \ \lambda_x.(a\ \ x) \to a$, if $x \notin \mathcal{F}var(a)$

Unification in $\Lambda(\mathcal{V})$ differs from the one in the context of first order term algebras, because bound variables in $\Lambda(\mathcal{V})$ are not affected by unification substitutions.

Notice that our notion of substitution is not completely satisfactory because the idea of fresh variables is implicit and depends on the history of the renaming process.

Unification variables in the λ-calculus are free variables. Thus free variables occurring in terms of a unification problem can be partitioned into true **unification variables** and **constants**, that cannot be bound by the unifiers. Observe that constants, as free variables, cannot be changed by the β-reduction process. However, from the point of view of unification, both constants and bound variables can be considered to be of the same syntactical category, since they cannot belong to the domain of unifiers. To differentiate between unification and constant variables, we will consider unification variables as meta-variables in a set $\mathcal{X}$. Thus, λ-calculus is defined as the term algebra over the set of operators $\{\lambda_x.\_ \mid x \in \mathcal{V}\} \cup \{(\_\ \_)\} \cup \mathcal{V}$ and the set of variables $\mathcal{X}$. Unification and constant variables are written as uppercase $(X, Y, \dots)$ and lowercase $(x, y, \dots)$ last letters of the Roman alphabet, respectively.

Definition 2.5 Terms $\Lambda(\mathcal{V}, \mathcal{X})$, of the **$\lambda$-calculus with names** are inductively defined by:

$$a ::= x \mid X \mid (a\ \ a) \mid \lambda_x.a \qquad \text{where } x \in \mathcal{V} \text{ and } X \in \mathcal{X}.$$

Now, substitution over $\mathcal{X}$ should be defined and substitution is modified to include:
Modified Definition 2.3 4) $\theta^{ext}(X) = X$, if $X \in \mathcal{X}$.

Grafting appears to be appropriate for the substitution of meta-variables since bound variables (in $\mathcal{V}$) remain unchanged when grafting variables in $\mathcal{X}$. But the problem of capture by abstractors remains when a meta-variable is replaced with a term containing constants; for instance, consider the grafting $\theta = \{X \mapsto x\}$ and the term $\lambda_x.X$. Then $(\lambda_x.X)\theta = \lambda_x.x$. Consequently, the notion of substitution for meta-variables should involve bound variable renaming.

Definition 2.6 Let θ a valuation from $\mathcal{X}$ to $\Lambda(\mathcal{V}, \mathcal{X})$. The **substitution** extending θ, denoted by θ^{ext} is defined by induction on the structure of terms in $\Lambda(\mathcal{V}, \mathcal{X})$ as follows:
1) $\theta^{ext}(X) = X\theta$, if $X \in \mathcal{X}$; 2) $\theta^{ext}(x) = x$ if $x \in \mathcal{V}$; 3) $\theta^{ext}((a\ \ b)) = (\theta^{ext}(a)\ \ \theta^{ext}(b))$;
4) $\theta^{ext}(\lambda_x.a) = \lambda_z.\theta^{ext}((\alpha^{var(\theta)\cup\{x\}\cup\{z\}}(a))\{x \mapsto z\})$, where z is a fresh variable.

It can be easily checked that the non commutativity problem of β- or η-reduction and grafting does not occur with our previous notion of substitution.

Lemma 2.7 β-reduction as well as η-reduction commute with substitution.

2.2 The λ-calculus in de Bruijn notation

In the previous section we have seen that the names of bound variables and their corresponding abstractors play a semantically irrelevant role in the λ-calculus. So any term in $\Lambda(\mathcal{V})$ (or in $\Lambda(\mathcal{V}, \mathcal{X})$) can be seen as a syntactical representative of its obvious equivalence class. Thus, one can conclude that the role that names of bound variables and their corresponding abstractors play, when treating syntactically unification in the λ-calculus, increases the complexity of the process and creates confusion.

Consequently, avoiding names in the λ-calculus is an effective way of clarifying the meaning of λ-terms and, for the unification process, of eliminating dummy and redundant renaming. N. de Bruijn developed a notation for the λ-calculus where names of bound variables were replaced by indices [13, 15, 14]. These indices relate bound variables to their corresponding abstractors.

It is clear that the correspondence between an occurrence of a bound variable and its associated abstractor operator is uniquely determined by its *depth*, that is the number of abstractors between them. Hence, λ terms can be written in a term algebra over the natural numbers $\mathbb{N}$, representing depth indices, the application operator $(\_\ \_)$ and a sole abstractor operator $\lambda.\_$; i.e., $\mathcal{T}(\{(\_\ \_), \lambda.\_\} \cup \mathbb{N})$.

In de Bruijn's notation, the solution for indexing occurrences of free variables is given by the creation of a *referential* according to a fixed enumeration of the set of variables $\mathcal{V}$, say $x, y, z, \ldots$, and prefixing all λ-terms with $\ldots \lambda_z.\lambda_y.\lambda_x.\_$.

Example 2.8 Using the referential $x, y, z, \ldots$ the term $\lambda_x.((\lambda_z.(x\ \ \lambda_x.(z\ \ x))\ \ x)\ \ \lambda_x.(x\ \ y))$ is rewritten as $\lambda.((\lambda.(\mathtt{2}\ \ \lambda.(\mathtt{2}\ \ \mathtt{1}))\ \ \mathtt{1})\ \ \lambda.(\mathtt{2}\ \ \mathtt{4}))$ and $\lambda_x.((\lambda_z.(y\ \ \lambda_x.(y\ \ x))\ \ y)\ \ \lambda_x.(z\ \ y))$, which has a multiple occurrence of free variables, as $\lambda.((\lambda.(\mathtt{4}\ \ \lambda.(\mathtt{5}\ \ \mathtt{1}))\ \ \mathtt{3})\ \ \lambda.(\mathtt{5}\ \ \mathtt{4}))$.

Now we can define the λ-calculus in de Bruijn notation with meta-variables.

Definition 2.9 The set $\Lambda_{dB}(\mathcal{X})$ of **λ-terms in de Bruijn notation** is defined inductively as:

$$a ::= \mathtt{n} \mid X \mid (a\ \ a) \mid \lambda.a \qquad \text{where } X \in \mathcal{X} \text{ and } \mathtt{n} \in \mathbb{N} \setminus \{0\}.$$

We type de Bruijn indices as $\mathtt{1}, \mathtt{2}, \mathtt{3}, \ldots, \mathtt{n}, \ldots$, to distinguish them from scripts.

An attempt to define β-reduction in the context of the λ-calculus in de Bruijn notation is $(\lambda.a\ \ b) \to \{1/b\}a$ where $\{1/b\}a$ is the substitution of the index 1 in a with b. But it fails because: firstly, when eliminating the leading abstractor all indices associated with free variable occurrences in a should be decremented by one; secondly, when propagating the substitution $\{1/b\}$ crossing abstractors through a the indices of the substitution (initially 1) and of the free variables in b should be incremented.

Consequently, we need new operators for detecting and incrementing and decrementing free variables to define a new notion of substitution.

Definition 2.10 Let $a \in \Lambda_{dB}(\mathcal{X})$. The **$i$-lift** of a, denoted a^{+i} is defined inductively as follows:

1) $X^{+i} = X$, for $X \in \mathcal{X}$ 2) $(a_1\ \ a_2)^{+i} = (a_1^{+i}\ \ a_2^{+i})$

3) $(\lambda.a_1)^{+i} = \lambda.a_1^{+(i+1)}$ 4) $\mathtt{n}^{+i} = \begin{cases} \mathtt{n}+\mathtt{1}, \text{ if } n > i \\ \mathtt{n}, \text{ if } n \leq i \end{cases}$ for $n \in \mathbb{N}$.

The **lift** of a term a is its 0-lift and is denoted briefly as a^+.

Definition 2.11 The application of the **substitution** with b at the depth $n-1, n \in \mathbb{N} \setminus \{0\}$, denoted $\{\mathtt{n}/b\}a$, on a term a in $\Lambda_{dB}(\mathcal{X})$ is defined inductively as follows:

1) $\{\mathtt{n}/b\}X = X$, for $X \in \mathcal{X}$ 2) $\{\mathtt{n}/b\}(a_1\ \ a_2) = (\{\mathtt{n}/b\}a_1\ \ \{\mathtt{n}/b\}a_2)$

3) $\{\mathtt{n}/b\}\lambda.a_1 = \lambda.\{\mathtt{n}+\mathtt{1}/b^+\}a_1$ 4) $\{\mathtt{n}/b\}\mathtt{m} = \begin{cases} \mathtt{m}-\mathtt{1}, \text{ if } m > n \\ b, \text{ if } m = n \\ \mathtt{m}, \text{ if } m < n \end{cases}$ if $m \in \mathbb{N}$.

Now we can define β-reduction in $\Lambda_{dB}(\mathcal{X})$.

Definition 2.12 **β-reduction** in the λ-calculus with de Bruijn indices is defined as $(\lambda.a\ \ b) \to \{\mathtt{1}/b\}a$.

Observe that the rewriting system of the sole β-reduction rule is left linear and non overlapping (i.e. orthogonal). Consequently, the rewriting system defined over $\Lambda_{dB}(\mathcal{X})$ by the β-reduction rule is CR.

Turning to the η-reduction rule, in the setting of the λ-calculus with names, this is defined as $\lambda_x.(a\ \ x) \to a$, if $x \notin \mathcal{F}var(a)$. In the language of $\Lambda_{dB}(\mathcal{X})$, the left side of this rule is written as $\lambda.(a'\ \ 1)$, where a' stands for the corresponding translation of a under some fixed referential of variables into the language of $\Lambda_{dB}(\mathcal{X})$. "$a$ has no free occurrences of x" means, in $\Lambda(\mathcal{X})$, that there are neither occurrences in a' of the index 1 at height zero nor of the index 2 at height one nor of the index 3 at height two etc. This means, in general, that there exists a term b such that $b^+ = a$.

Definition 2.13 **η-reduction** in the λ-calculus with de Bruijn indices is: $\lambda.(a\ \ 1) \to b$ if $\exists b\ b^+ = a$.

Definition 2.14 Let $\theta = \{X_1 \mapsto a_1, \ldots, X_n \mapsto a_n\}$ be a valuation from the set of meta-variables $\mathcal{X}$ to $\Lambda_{dB}(\mathcal{X})$. The corresponding **substitution**, also denoted θ, is defined inductively by:

1) $\theta(\mathtt{m}) = \mathtt{m}$ for $m \in \mathbb{N}$ 2) $\theta(X) = X\theta$, for $X \in \mathcal{X}$

3) $\theta(a_1\ \ a_2) = (\theta(a_1)\ \ \theta(a_2))$ 4) $\theta\lambda.a_1 = \lambda.\theta^+(a_1)$

where θ^+ denotes the substitution $\theta^+ = \{X_1/a_1^+, \ldots, x_n/a_n^+\}$ built from the grafting $\{X_1 \mapsto a_1^+, \ldots, x_n \mapsto a_n^+\}$.

3 Calculi à la $\lambda\sigma$ and λs_e

In this section we present the $\lambda\sigma$- and λs_e-calculi and their typed versions and establish properties of the λs_e-calculus needed for the unification process.

3.1 The $\lambda\sigma$-calculus

We introduce the $\lambda\sigma$-calculus which works on 2-sorted terms: *(proper) terms* and *substitutions.* We use $s, t, \ldots$ to range over the set of substitutions.

Definition 3.1 The $\lambda\sigma$-calculus is defined as the calculus of the rewriting system $\lambda\sigma$ of Table 1 where TERMS $a\ ::=\ \mathtt{1} \mid X \mid (a\ \ a) \mid \lambda a \mid a[s]$ and SUBS $s\ ::=\ id \mid \uparrow \mid a.s \mid s \circ s$.

For every substitution s we define the *iteration of the composition of* s inductively as $s^1 = s$ and $s^{n+1} = s \circ s^n$. We use the convention $s^0 = id$. Note that the only de Bruijn index used is $\mathtt{1}$, but we can code $\mathtt{n}$ by the term $\mathtt{1}[\uparrow^{n-1}]$.

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

The rewriting system $\lambda\sigma$ is locally confluent [1], CR on substitution-closed terms (i.e., terms without substitution variables) [40] and not CR on open terms (i.e., terms with term and substitution variables) [12, 11]. The possible forms of a $\lambda\sigma$-term in $\lambda\sigma$-*normal form* were given in [40] as: 1. $\lambda.a$, where a is a normal term; 2. $a_1 \ldots a_p. \uparrow^n$, where $a_1, \ldots, a_p$ are normal terms and $a_p \neq \mathtt{n}$ or 3. $(a\ b_1 \ldots b_n)$, where a is either $\mathtt{1}$, $\mathtt{1}[\uparrow^n]$, X or $X[s]$ for s a substitution term different from id in normal form.

TABLE 1. The $\lambda\sigma$ Rewriting System of the $\lambda\sigma$-calculus

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

In the λ-calculus with names or de Bruijn indices, the rule $X\{y/a\} = X$, where y is an element of $\mathcal{V}$ or a de Bruijn index, respectively, is necessary because there is no way to suspend the substitution $\{y/a\}$ until X is instantiated. In the $\lambda\sigma$-calculus, the application of this substitution can be delayed, since the term $X[s]$ does not reduce to X. The fact that the application of a substitution to a meta-variable can be suspended until the meta-variable is instantiated will be used to code the substitution of variables in $\mathcal{X}$ by "$\mathcal{X}$-grafting" and explicit lifting. Consequently a notion of $\mathcal{X}$-substitution in the $\lambda\sigma$-calculus is unnecessary. Observe that the condition $a =_\sigma b[\uparrow]$ of the *Eta* rule is stronger than the condition $a = b^+$ given in Definition 2.13 as $X = X^+$, but there exists no term b such that $X =_\sigma b[\uparrow]$. Note that $\lambda\sigma$-reduction is compatible with first order substitution or grafting and hence $\mathcal{X}$-grafting and $\lambda\sigma$-reduction commute.

Definition 3.2 (The $\lambda\sigma_{dB}$-calculus) The syntax of the $\lambda\sigma_{dB}$-calculus is that of the $\lambda\sigma$-calculus where 1 is replaced by $\mathbb{N}$. The set, $\lambda\sigma_{dB}$, of rules of the $\lambda\sigma_{dB}$*-calculus* is $\lambda\sigma$ where *(VarId)* is replaced by the four rules: $a[id] \to a$, $\mathtt{n}+\mathtt{1}[a \cdot s] \to \mathtt{n}[s]$, $\mathtt{n}[\uparrow] \to \mathtt{n}+\mathtt{1}$ and $\mathtt{n}[\uparrow \circ s] \to \mathtt{n}+\mathtt{1}[s]$.

Notice that the $\lambda\sigma_{dB}$-calculus consists of an infinite set of rules that should be treated modulo linear arithmetic.

3.2 The λs-calculus

The λs-calculus was introduced in [26] with the aim of providing a calculus that preserves strong normalization and has a confluent extension on open terms [27]. It avoids introducing two different sets of entities and insists on remaining close to the syntax of the λ-calculus. Next to λ and application, substitution (σ) and updating

(φ) operators are introduced. A term containing neither σ nor φ is called a *pure term.* The λs-calculus is CR on closed terms, preserves strong normalisation, its substitution calculus is SN, and it has a confluent extension on open terms, the λs_e-calculus. This calculus was originally introduced without the *Eta* rule that we consider here.

Definition 3.3 Terms of the **λs-calculus** are given by:
$\Lambda s ::= \mathbb{N} \mid \Lambda s \Lambda s \mid \lambda \Lambda s \mid \Lambda s\, \sigma^i \Lambda s \mid \varphi^i_k \Lambda s$
where $i \geq 1$, $k \geq 0$. The set of rules λs is given in Table 2.

TABLE 2. The Rewriting System of the λs-calculus with η-rule

(σ-generation)	$(\lambda.a\ \ b)$	$\longrightarrow$	$a\,\sigma^1\, b$
(σ-λ-transition)	$(\lambda.a)\,\sigma^i b$	$\longrightarrow$	$\lambda.(a\,\sigma^{i+1}\, b)$
(σ-app-transition)	$(a_1\ \ a_2)\,\sigma^i b$	$\longrightarrow$	$((a_1\,\sigma^i b)\ \ (a_2\,\sigma^i b))$
(σ-destruction)	$\mathtt{n}\,\sigma^i b$	$\longrightarrow$	$\begin{cases} \mathtt{n}-\mathtt{1} & \text{if } n > i \\ \varphi^i_0\, b & \text{if } n = i \\ \mathtt{n} & \text{if } n < i \end{cases}$
(φ-λ-transition)	$\varphi^i_k(\lambda.a)$	$\longrightarrow$	$\lambda.(\varphi^i_{k+1}\, a)$
(φ-app-transition)	$\varphi^i_k(a_1\ \ a_2)$	$\longrightarrow$	$((\varphi^i_k\, a_1)\ \ (\varphi^i_k\, a_2))$
(φ-destruction)	$\varphi^i_k\,\mathtt{n}$	$\longrightarrow$	$\begin{cases} \mathtt{n}+\mathtt{i}-\mathtt{1} & \text{if } n > k \\ \mathtt{n} & \text{if } n \leq k \end{cases}$
(Eta)	$\lambda.(a\ \ \mathtt{1})$	$\longrightarrow$	b if $a =_s \varphi^2_0 b$

The equational theory associated to the rewriting system λs defines a congruence $=_{\lambda s}$. The congruence obtained by dropping *σ-generation* and *Eta* is denoted by $=_s$.

In order to clarify differences between the $\lambda\sigma$-calculus and the λs-calculus, we show the correspondence between their *Eta* rules; i.e., the correspondence between both conditions $b[\uparrow] = a$ and $\varphi^2_0 b = a$ of their associated *Eta* rules.

Lemma 3.4 Let $\mathtt{n} \in \mathbb{N}$ a de Bruijn index. Then for all $k \geq 0$ the s-normal form of $\varphi^2_k\mathtt{n}$ and the σ-normal form of $\mathtt{n}[\mathtt{1}.\mathtt{1}[\uparrow].\mathtt{1}[\uparrow^2].\dots.\mathtt{1}[\uparrow^{k-1}].\uparrow^{k+1}]$ are a de Bruijn index and its corresponding code in the language of the $\lambda\sigma$-calculus.

PROOF. If $k = 0$ then we have $\mathtt{n}[\uparrow] = \mathtt{1}[\uparrow^{n-1}][\uparrow] \longrightarrow \mathtt{1}[\uparrow^n] = \mathtt{n}+\mathtt{1}$ and $\varphi^2_0\mathtt{n} \longrightarrow \mathtt{n}+\mathtt{1}$ else if $k > 0$, by applying rule *clos* once, we have:
$\mathtt{1}[\uparrow^{n-1}][\mathtt{1}.\mathtt{1}[\uparrow].\mathtt{1}[\uparrow^2].\dots.\mathtt{1}[\uparrow^{k-1}].\uparrow^{k+1}] \longrightarrow \mathtt{1}[\uparrow^{n-1}\circ(\mathtt{1}.\mathtt{1}[\uparrow].\dots.\mathtt{1}[\uparrow^{k-1}].\uparrow^{k+1})]$.
Two cases should be considered noting that if $n > k$ then $\varphi^2_k\mathtt{n} \longrightarrow_{\varphi-destruction} \mathtt{n}+\mathtt{1}$ and if $n \leq k$ then $\varphi^2_k\mathtt{n} \longrightarrow_{\varphi-destruction} \mathtt{n}$.

Subcase 1: $n \leq k$. Then $\mathtt{1}[\uparrow^{n-1}\circ(\mathtt{1}.\mathtt{1}[\uparrow].\dots.\mathtt{1}[\uparrow^{k-1}].\uparrow^{k+1})] \longrightarrow^{n-1}_{ShiftCons}$
$\mathtt{1}[\mathtt{1}[\uparrow^{n-1}].\dots.\mathtt{1}[\uparrow^{k-1}].\uparrow^{k+1}] \longrightarrow_{VarCons} \mathtt{1}[\uparrow^{n-1}] = \mathtt{n}$

Subcase 2: $n > k$. Then $\mathtt{1}[\uparrow^{n-1}\circ(\mathtt{1}.\mathtt{1}[\uparrow].\dots.\mathtt{1}[\uparrow^{k-1}].\uparrow^{k+1})] \longrightarrow^{k}_{ShiftCons}$
$\mathtt{1}[\mathtt{1}[\uparrow^{n-1-k}]\circ\uparrow^{k+1}] = \mathtt{1}[\uparrow^n] = \mathtt{n}+\mathtt{1}$ ∎

Lemma 3.5 Let $\lambda.a$ be an abstraction over the language of Λ_{dB}. Then we have: for $k \geq 0$, $(\lambda.a)[\mathtt{1}.\mathtt{1}[\uparrow]. \ldots .\mathtt{1}[\uparrow^{k-1}]. \uparrow^{k+1}]$ σ-reduces into $\lambda.(a[\mathtt{1}.\mathtt{1}[\uparrow]. \ldots .\mathtt{1}[\uparrow^{k}]. \uparrow^{k+2}])$.

Proof. If $k = 0$ then $\lambda.a[\uparrow] \longrightarrow_{Abs} \lambda.a[1. \uparrow^2]$. If $k > 0$ then $\lambda.a[\mathtt{1}.\mathtt{1}[\uparrow]. \ldots .\mathtt{1}[\uparrow^{k-1}]. \uparrow^{k+1}] \longrightarrow_{Abs} \lambda.a[1.((\mathtt{1}.\mathtt{1}[\uparrow]. \ldots .\mathtt{1}[\uparrow^{k-1}]. \uparrow^{k+1}) \circ \uparrow)] \longrightarrow^{k}_{Map}$ $\lambda.a[1.(\mathtt{1}[\uparrow].\mathtt{1}[\uparrow][\uparrow]. \ldots .\mathtt{1}[\uparrow^{k-1}][\uparrow]. \uparrow^{k+1} \circ \uparrow)] \longrightarrow^{k-1}_{Clos} \lambda.a[\mathtt{1}.\mathtt{1}[\uparrow]. \ldots .\mathtt{1}[\uparrow^{k}]. \uparrow^{k+2}]$. ∎

The correspondence between $b[\uparrow]$ and $\varphi^2_0 b$ is the case $k = 0$ of the following lemma.

Lemma 3.6 Let $a \in \Lambda_{dB}$ and a' its translation in the $\lambda\sigma$-calculus, where all indices $\mathtt{n} \in \mathbb{N}$ occurring at a are replaced with $\mathtt{1}[\uparrow^{n-1}]$. Then, for all $k \geq 0$, the σ-normal form of $a'[\mathtt{1}.\mathtt{1}[\uparrow]. \ldots .\mathtt{1}[\uparrow^{k-1}]. \uparrow^{k+1}]$ is the translation of the s-normal form of $\varphi^2_k a$.

Proof. This is proved by induction on the structure of terms.

Firstly, observe that it holds for $a = \mathtt{n} \in \mathbb{N}$ because of Lemma 3.4.

Secondly, suppose it holds for all $k \geq 0$ for terms a and b. Then for the application $(a \;\; b)$ we have $\varphi^2_k(a \;\; b) \longrightarrow_{\varphi-app-transition} (\varphi^2_k a \;\; \varphi^2_k b)$ and $(a \;\; b)[\mathtt{1}.\mathtt{1}[\uparrow]. \ldots .\mathtt{1}[\uparrow^{k-1}]. \uparrow^{k+1}] \longrightarrow_{App} (a[\mathtt{1}.\mathtt{1}[\uparrow]. \ldots .\mathtt{1}[\uparrow^{k-1}]. \uparrow^{k+1}] \;\; b[\mathtt{1}.\mathtt{1}[\uparrow]. \ldots .\mathtt{1}[\uparrow^{k-1}]. \uparrow^{k+1}])$.

Finally, suppose it holds for all $k \geq 0$ and for a term a. Thus by Lemma 3.5 we have $(\lambda.a)[\mathtt{1}.\mathtt{1}[\uparrow]. \ldots .\mathtt{1}[\uparrow^{k-1}]. \uparrow^{k+1}] \longrightarrow^{*} \lambda.(a[\mathtt{1}.\mathtt{1}[\uparrow]. \ldots .\mathtt{1}[\uparrow^{k}]. \uparrow^{k+2}])$ and $\varphi^2_k \lambda.a \longrightarrow \lambda.\varphi^2_{k+1} a$. By the induction hypothesis the lemma holds for the corresponding normal forms of $\varphi^2_{k+1} a$ and $a[\mathtt{1}.\mathtt{1}[\uparrow]. \ldots .\mathtt{1}[\uparrow^{k}]. \uparrow^{k+2}]$. Hence, it holds for the abstraction. ∎

The previous lemma can be easily extended for terms $a \in \Lambda_{dB}(\mathcal{X})$. In fact, observe that for a meta-variable $X \in \mathcal{X}$ at a position $i \in O(a)$, the corresponding subterms of the σ- and s-normal forms of $a[\uparrow]$ and $\varphi^2_0 a$ are of the form $X[\mathtt{1}.\mathtt{1}[\uparrow]. \ldots .\mathtt{1}[\uparrow^{k-1}]. \uparrow^{k+1}]$ and $\varphi^2_k X$, respectively, when the height of the occurrence of X at position i is k.

3.3 The λs_e-calculus

We introduce the open terms and the rules that extend λs to obtain the λs_e-calculus.

Definition 3.7 The set of *open terms*, noted Λs_{op} is given as follows:

$\Lambda s_{op} ::= \mathcal{X} \,|\, \mathbb{N} \,|\, \Lambda s_{op}\Lambda s_{op} \,|\, \lambda\Lambda s_{op} \,|\, \Lambda s_{op}\, \sigma^j \Lambda s_{op} \,|\, \varphi^i_k \Lambda s_{op}$ where $j, i \geq 1$, $k \geq 0$ and $\mathcal{X}$ stands for a set of variables, over which X, Y, ... range. *Closures*, *pure terms* and *compatibility* are defined as for Λs.

Working with open terms one loses confluence as shown by the following example:

$$((\lambda X)Y)\sigma^1\mathtt{1} \to (X\sigma^1 Y)\sigma^1\mathtt{1} \qquad ((\lambda X)Y)\sigma^1\mathtt{1} \to ((\lambda X)\sigma^1\mathtt{1})(Y\sigma^1\mathtt{1})$$

and $(X\sigma^1 Y)\sigma^1\mathtt{1}$ and $((\lambda X)\sigma^1\mathtt{1})(Y\sigma^1\mathtt{1})$ have no common reduct. This example shows that even the WCR property is lost. But the solution lies in the properties of meta-substitutions and updating functions of the λ-calculus in de Bruijn notation [27]. These properties are equalities which can be given a suitable orientation and the new rules, thus obtained, added to λs give origin to a rewriting system which is WCR.

Definition 3.8 The set of rules λs_e is obtained by adding the rules given in Table 3 to the set λs in Table 2. The **λs_e-calculus** is the reduction system $(\Lambda s_{op}, \to_{\lambda s_e})$ where $\to_{\lambda s_e}$ is the least compatible reduction on Λs_{op} generated by the set of rules λs_e. The **calculus of substitutions associated with the λs_e-calculus** is the rewriting system generated by the set of rules $s_e = \lambda s_e - \{\sigma\text{-}generation, Eta\}$ and we

TABLE 3. The Rewriting System of the λs_e-calculus without rules in Table 2

(σ-σ-transition)	$(a\,\sigma^i b)\,\sigma^j\, c$	$\longrightarrow$	$(a\,\sigma^{j+1}\, c)\;\sigma^i\,(b\,\sigma^{j-i+1}\, c)$	if $i \leq j$
(σ-φ-transition 1)	$(\varphi^i_k\, a)\,\sigma^j\, b$	$\longrightarrow$	$\varphi^{i-1}_k\, a$	if $k < j < k+i$
(σ-φ-transition 2)	$(\varphi^i_k\, a)\,\sigma^j\, b$	$\longrightarrow$	$\varphi^i_k(a\,\sigma^{j-i+1}\, b)$	if $k+i \leq j$
(φ-σ-transition)	$\varphi^i_k(a\,\sigma^j\, b)$	$\longrightarrow$	$(\varphi^i_{k+1}\, a)\,\sigma^j\,(\varphi^i_{k+1-j}\, b)$	if $j \leq k+1$
(φ-φ-transition 1)	$\varphi^i_k\,(\varphi^j_l\, a)$	$\longrightarrow$	$\varphi^j_l\,(\varphi^i_{k+1-j}\, a)$	if $l+j \leq k$
(φ-φ-transition 2)	$\varphi^i_k\,(\varphi^j_l\, a)$	$\longrightarrow$	$\varphi^{j+i-1}_l\, a$	if $l \leq k < l+j$

call it s_e**-calculus**. Additionally, condition of the *Eta* rule should be changed with $\varphi^2_0 b =_{s_e} a$.

We can describe operators of the λs_e-calculus over the signature of a first order sorted term algebra $\mathcal{T}_{\lambda s_e}(\mathcal{X})$ built on $\mathcal{X}$, the set of variables of sort TERM and its subsort NAT$\subset$TERM:

$$\begin{array}{rcrcl}
\mathtt{n} & : & & \rightarrow & \text{NAT}, \quad \forall n \in \mathbb{N} \setminus \{0\} \\
(\_\ \_) & : & \text{TERM} \times \text{TERM} & \rightarrow & \text{TERM} \\
\lambda.\_ & : & \text{TERM} & \rightarrow & \text{TERM} \\
\_\sigma^i\_ & : & \text{TERM} \times \text{TERM} & \rightarrow & \text{TERM}, \quad \forall i \in \mathbb{N} \setminus \{0\} \\
\varphi^i_k\_ & : & \text{TERM} & \rightarrow & \text{TERM}, \quad \forall i \in \mathbb{N}, k \in \mathbb{N} \setminus \{0\}
\end{array}$$

Notice that for the $\lambda\sigma$-calculus we need two sorts: TERM and SUBSTITUTION [16]. The set of variables of sort TERM in a term $a \in \mathcal{T}_{\lambda s_e}(\mathcal{X})$ is denoted by $\mathcal{T}var(a)$.

Proposition 3.9 $\mathcal{X}$-grafting and λs_e-reduction commute.

Theorem 3.10 ([27]) • WN and CR of s_e: The s_e-calculus is WN and CR.

- Simulation of β-reduction: Let $a,\, b \in \Lambda$, if $a \rightarrow_\beta b$ then $a \rightarrow^*_{\lambda s_e} b$.
- CR of λs_e: The λs_e-calculus is CR on open terms.
- Soundness: Let $a,\, b \in \Lambda$, if $a \rightarrow^*_{\lambda s_e} b$ then $a \rightarrow^*_\beta b$.

The characterization of the s_e-normal forms is given by the following theorem:

Theorem 3.11 ([27]) A term $a \in \Lambda s_{op}$ is an s_e-normal form if and only if one of the following holds:

1. $a \in \mathcal{X} \cup \mathbb{N}$;
2. $a = (b\ \ c)$, where b, c are s_e-normal forms;
3. $a = \lambda.b$, where b is an s_e-normal form;
4. $a = b\sigma^j c$, where c is an s_e-normal form and b is an s_e-normal form of one of the following forms: (a) X, (b) $d\sigma^i e$, with $j < i$ or (c) $\varphi^i_k d$, with $j \leq k$;
5. $a = \varphi^i_k b$, where b is an s_e-normal form of one of the following forms: (a) X, (b) $c\sigma^j d$, with $j > k+1$ or (c) $\varphi^j_l c$, with $k < l$;

PROOF. We verify the non existence of redices from the rules of the s_e-calculus. The first three cases are obviously normalized forms. For the fourth case we should analyse, possible redices from the σ-rules (i.e., rules whose name begin with σ). Analyzing each of the corresponding subcases we have: a) no σ-rule applies; b) the sole possible redex is from the *σ-σ-transition* rule, that does not apply because of the restriction on the scripts; c) both *σ-φ-transition* rules 1 and 2 do not apply because of the restriction on scripts. For the fifth case we should analyse, possible redices from the φ-rules (i.e., rules whose name begin with φ). We have the following subcases: a) obviously we have a normal form; b) the sole possible redex is the one from the *φ-σ-transition* rule, that does not applies because of the restriction on scripts; c) both candidate rules, the *φ-φ-transition* ones do not apply because of the restriction on scripts. ∎

As corollary we obtain a characterization of λs_e-normal forms.

Corollary 3.12 (λs_e-normal forms) A term $a \in \Lambda s_{op}$ is a λs_e-normal form if and only if one of the following holds:

1. $a \in \mathcal{X} \cup \mathbb{N}$;
2. $a = (b \ \ c)$, where b, c are λs_e-normal forms and b is not an abstraction $\lambda.d$;
3. $a = \lambda.b$, where b is a λs_e-normal form excluding applications of the form $(c \ \ \mathtt{1})$ such that there exists d with $\varphi_0^2 d =_{s_e} c$;
4. $a = b\sigma^j c$, where c is a λs_e-normal form and b is an λs_e-normal form of one of the following forms: (a) X, (b) $d\sigma^i e$, with $j < i$ or (c) $\varphi_k^i d$, with $j \leq k$;
5. $a = \varphi_k^i b$, where b is a λs_e-normal form of one of the following forms: (a) X, (b) $c\sigma^j d$, with $j > k + 1$ or (c) $\varphi_l^j c$, with $k < l$;

PROOF. Items 2 and 3 result from adapting the proof of Theorem 3.10 to avoid redices of the *σ-generation* and *Eta* rules of the λs_e-calculus. ∎

3.4 Typed λ-calculi

We recall that environments in de Bruijn setting are simply lists of types and in the case of the $\lambda\sigma$-calculus, substitutions receive environments as types. We introduce the following notation concerning environments. If Γ is the environment $\Gamma_1.\Gamma_2.\ldots.\Gamma_n.nil$, then $\Gamma_{\geq i}$ denotes the environment $\Gamma_i.\Gamma_{i+1}.\ldots.\Gamma_n.nil$; analogously, $\Gamma_{\leq i}$ stands for $\Gamma_1.\ldots.\Gamma_i$, etc. The rewrite rules of the corresponding typed calculi are exactly the same (except that rules involving abstractions are now typed). In all these calculi, we assume types and environments built by TYPES $A \mid A \to B$ and ENVIRS Γ $nil \mid A.\Gamma$.

Here are the typing rules for the simply typed λ-calculus in de Bruijn notation:

Definition 3.13 The syntax of **simply typed λ-calculus in de Bruijn notation** is defined by:

The set of terms TERMS $a ::== \mathtt{n} \mid (a \ \ b) \mid \lambda_A.a$

The **typing system**, called **L1**, and given by the following rules:

$$(\mathbf{L1}\text{-}var) \quad A.\Gamma \vdash \mathtt{1} : A \qquad (\mathbf{L1}\text{-}\lambda) \quad \frac{A.\Gamma \vdash b : B}{\Gamma \vdash \lambda_A.b : A \to B}$$

$$(\mathbf{L1}\text{-}varn) \quad \frac{\Gamma \vdash \mathtt{n} : B}{A, \Gamma \vdash \mathtt{n+1} : B} \qquad (\mathbf{L1}\text{-}app) \quad \frac{\Gamma \vdash b : A \to B \quad \Gamma \vdash a : A}{\Gamma \vdash (b \ \ a) : B}$$

Observe that typing and grafting are not compatible.

Example 3.14 Consider the environment $\Gamma = A.(A \to A) \to (B \to A) \to A.nil$ and a variable X of type A. Let us show that $\Gamma \vdash ((2\ \ \lambda_A.X)\ \ \lambda_B.X) : A$. Firstly, $\mathtt{2}$ is typed:

$$(varn)\ \frac{(var)(A \to A) \to (B \to A) \to A.nil \vdash \mathtt{1} : (A \to A) \to (B \to A) \to A}{A.(A \to A) \to (B \to A) \to A.nil \vdash \mathtt{2} : (A \to A) \to (B \to A) \to A}$$

Afterwards, the necessary abstractions are typed:

$$(\lambda)\ \frac{A.\Gamma \vdash X : A}{\Gamma \vdash \lambda_A.X : A \to A} \qquad (\lambda)\ \frac{B.\Gamma \vdash X : A}{\Gamma \vdash \lambda_B.X : B \to A}$$

$$\text{Finally,}\ (app)\ \cfrac{\cfrac{\Gamma \vdash \mathtt{2} : (A \to A) \to (B \to A) \to A \quad \Gamma \vdash \lambda_A.X : A \to A}{\Gamma \vdash (\mathtt{2}\ \ \lambda_A.X) : (B \to A) \to A \quad \Gamma \vdash \lambda_B.X : B \to A}}{\Gamma \vdash ((\mathtt{2}\ \ \lambda_A.X)\ \ \lambda_B.X) : A}\,(app)$$

Observe that applying the grafting $\{X/\mathtt{1}\}$ to the term $((2\ \ \lambda_A.X)\ \ \lambda_B.X)$ we obtain the term $((2\ \ \lambda_A.\mathtt{1})\ \ \lambda_B.\mathtt{1})$, which is not well-typed.

The next proposition establishes compatibility between substitution and typing.

Proposition 3.15 ([16]) Take a variable X of type B and an environment Γ. If $\Gamma \vdash a : A$ and $\Gamma \vdash b : B$ then $\Gamma \vdash \{X/b\}a : A$.

Proof. By induction on the structure of terms. Firstly, if $a = X$, then $A = B$. Secondly, if $a = \mathtt{n}$ then $\{X/b\}a = \mathtt{n}$. Thirdly, if $a = (a_1\ \ a_2)$ then we have that $\Gamma \vdash a_2 : A_1$ and $\Gamma \vdash a_1 : A_1 \to A$ and by the *app* typing rule $\Gamma \vdash (a_1\ \ a_2) : A$; by induction hypothesis $\Gamma \vdash \{X/b\}a_2 : A_1$ and $\Gamma \vdash \{X/b\}a_1 : A_1 \to A$ which implies $\Gamma \vdash \{X/b\}(a_1\ \ a_2) : A$. Finally, if a is an abstraction of the form $\lambda_C.a_1$ then A should be of the form $C \to D$ and $C.\Gamma \vdash a_1 : D$. By definition of substitution $\{X/b\}\lambda_C.a_1 = \lambda_C.\{X/b^+\}a_1$. Observe that if we can prove that $C.\Gamma \vdash b^+ : B$ then, since $C.\Gamma \vdash X : B$, we can suppose inductively that $C.\Gamma \vdash \{X/b^+\}a_1 : D$ and subsequently, by applying the λ typing rule, we can conclude that $\Gamma \vdash \lambda_C.\{X/b^+\}a_1 : C \to D$. To prove that if $\Gamma \vdash b : B$ then $C.\Gamma \vdash b^+ : B$ we prove by induction on the structure of terms that the following more general affirmation holds:

if $C_i.\dots.C_1.\Gamma \vdash b : B$ then $C_i.\dots.C_1.C.\Gamma \vdash b^{+i} : B$

Firstly, if $b = X$ then $b^{+i} = X$. Secondly, if $b = \mathtt{n}$ then if $n > i$ then $\mathtt{n}^{+i} = \mathtt{n}+\mathtt{1}$ and the type of $\mathtt{n}$ and of $\mathtt{n}+\mathtt{1}$ in the environments $C_i.\dots.C_1.\Gamma$ and $C_i.\dots.C_1.C.\Gamma$, respectively, coincide (in fact, it is the one of $\mathtt{n}-\mathtt{i}$ in the environment Γ). Else, if $n \leq i$ then $\mathtt{n}^{+i} = \mathtt{n}$. Thirdly, if b is an application of the form $(b_1\ \ b_2)$ then $(b_1\ \ b_2)^{+i} = (b_1^{+i}\ \ b_2^{+i})$ and by the induction hypothesis we have $C_i.\dots.C_1.C.\Gamma \vdash b_1^{+i}E \to B$ and $C_i.\dots.C_1.C.\Gamma \vdash b_2^{+i}E$, for some type E, which enables us to conclude that $C_i.\dots.C_1.C.\Gamma \vdash (b_1^{+i}\ \ b_2^{+i}) : B$. Finally, if b is an abstraction of the form $\lambda_E.b_1$ then $B = E \to F$ and $E.C_i.\dots.C_1.\Gamma \vdash b_1 : F$. By induction hypothesis $E.C_i.\dots.C_1.C.\Gamma \vdash b_1^{+i} : F$ and consequently $C_i.\dots.C_1.C.\Gamma \vdash \lambda_E.b_1^{+i} : E \to F$. ∎

We recall now the typing rules for λs and λs_e.

Definition 3.16 The syntax of **simply typed λs- and λs_e-calculus** is given by:

The set of terms TERMS a $::==$ $\mathtt{n} \mid X \mid (a\ \ b) \mid \lambda_A.a \mid a\sigma^i b \mid \varphi^i_k a, \quad \forall n, k \geq 0,\ \forall i \geq 1$

The **typing system Ls1**, given by the rules **Ls1**-*var*, **Ls1**-*varn*, **Ls1**-λ and **Ls1**-*app* which are exactly the same as **L1**-*var*, **L1**-*varn*, **L1**-λ and **L1**-*app*, respectively, and the new rules:

$$(\mathbf{Ls1}\text{-}\sigma) \qquad \frac{\Gamma_{\geq i} \vdash b : B \qquad \Gamma_{<i}.B.\Gamma_{\geq i} \vdash a : A}{\Gamma \vdash a\,\sigma^i b : A}$$

$$(\mathbf{Ls1}\text{-}\mathit{Mtv}) \qquad \Gamma_X \vdash X : A_X$$

$$(\mathbf{Ls1}\text{-}\varphi) \qquad \frac{\Gamma_{\leq k}.\Gamma_{\geq k+i} \vdash a : A}{\Gamma \vdash \varphi^i_k a : A}$$

(**Ls1**-*Mtv*) is added to type open terms and is taken to mean: for every metavariable X, there exists an environment Γ_X and a type A_X such that the rule holds. In order to obtain compatibility between typing and grafting, to each meta-variable X we associate a unique type A_X and a unique environment Γ_X. We assume for each pair (Γ, A) an infinite set of variables X such that $\Gamma_X = \Gamma$ and $A_X = A$.

Now we present the simply typed $\lambda\sigma$-calculus.

Definition 3.17 The syntax of **simply typed $\lambda\sigma$-calculus** is given by the sets of terms and substitutions: TERMS a $::==$ $\mathtt{1} \mid X \mid (a\ \ b) \mid \lambda_A.a \mid a[s]$ *and* SUBS s $::==$ $id \mid \uparrow \mid a.s \mid s \circ s$ and the rules **Lσ1**-*var*, **Lσ1**-λ and **Lσ1**-*app* which are exactly the same as **L1**-*var*, **L1**-λ and **L1**-*app*, respectively, together with the new rules:

$$(\mathbf{L\sigma 1}\text{-}\mathit{clos}) \quad \frac{\Gamma \vdash s \triangleright \Gamma' \qquad \Gamma' \vdash a : A}{\Gamma \vdash a[s] : A} \qquad (\mathbf{L\sigma 1}\text{-}\mathit{id}) \quad E \vdash id \triangleright \Gamma$$

$$(\mathbf{L\sigma 1}\text{-}\mathit{cons}) \quad \frac{\Gamma \vdash a : A \qquad \Gamma \vdash s \triangleright \Gamma'}{\Gamma \vdash a : A \cdot s \triangleright A, \Gamma'} \qquad (\mathbf{L\sigma 1}\text{-}\mathit{shift}) \quad A.\Gamma \vdash \uparrow \triangleright \Gamma$$

$$(\mathbf{L\sigma 1}\text{-}\mathit{comp}) \quad \frac{\Gamma \vdash s'' \triangleright \Gamma'' \qquad \Gamma'' \vdash s' \triangleright \Gamma'}{\Gamma \vdash s' \circ s'' \triangleright \Gamma'} \qquad (\mathbf{L\sigma 1}\text{-}\mathit{Mtv}) \quad \Gamma_X \vdash X : A_X$$

The reduction rules of both the typed $\lambda\sigma$-calculus and the typed λs_e-calculus are defined by adding to the rules in $\lambda\sigma$ and in λs_e the necessary typing information.

Definition 3.18 (Typed $\lambda\sigma$-calculus) The **typed $\lambda\sigma$-calculus** is defined by the rewrite rules of the rewriting system $\lambda\sigma$ (Table 1) changing the rules that involve abstractions as in Table 4. The resulting rewriting system is also called $\lambda\sigma$.

TABLE 4. The *Beta, Abs* and *Eta* rules of the typed $\lambda\sigma$-calculus

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

The typed version of σ has the same properties as the untyped one.

Proposition 3.19 (Grafting and typing are compatible [16]) If X is a variable and b a term such that $\Gamma_X \vdash b : A_x$ then:
1. $\forall$ environment Δ and term a such that $\Delta \vdash a : A$, we have: $\Delta \vdash a\{X/b\} : A$.
2. $\forall$ environments Δ, Δ' and substitution s such that $\Delta \vdash s \rhd \Delta'$, we have: $\Delta \vdash s\{X/b\} \rhd \Delta'$.

Proof. By simultaneous induction on the structure of typing derivation of $\Delta \vdash a : A$ and $\Delta \vdash s \rhd \Delta'$. ∎

Since a unique environment and a unique type are associated to every meta-variable, terms as $((Y\ \ X)\ \ X[\uparrow])$ and $((Y\ \ \lambda_A.X)\ \ X)$ cannot be typed in any environment.

Proposition 3.20 ([40]) The typed $\lambda\sigma$-calculus is WN and CR.

Definition 3.21 (η-long normal forms in $\lambda\sigma$) Let a be a $\lambda\sigma$-term of type $A_1 \to \ldots \to A_n \to B$ in the environment Γ and in $\lambda\sigma$-normal form. The **η-long normal form** of a, written a', is defined by:

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

Definition 3.21 has been proven correct and well-founded in [16]. The **long normal form** of a $\lambda\sigma$-term is defined as the η-long normal form of its $\beta\eta$-normal form. Two terms are $\beta\eta$-equivalent if and only if they have the same long normal form [16].

Definition 3.22 (Typed λs_e-calculus) The **typed λs_e-calculus** is defined by the rewrite rules of the rewriting system λs_e (rules in Tables 2 and 3) changing the rules that involve abstractions as in Table 5. The resulting rewriting system is also called λs_e.

Table 5. The *generation, transition* and *Eta* rules of the typed λs_e-calculus

(σ-generation)	$(\lambda_A.a\ \ b)$	$\longrightarrow$	$a\,\sigma^1\, b$		
(σ-λ-transition)	$(\lambda_A.a)\,\sigma^i b$	$\longrightarrow$	$\lambda_A.(a\,\sigma^{i+1}\, b)$		
(φ-λ-transition)	$\varphi^i_k(\lambda_A.a)$	$\longrightarrow$	$\lambda_A.(\varphi^i_{k+1}\, a)$		
(Eta)	$\lambda_A.(a\ \ \mathtt{1})$	$\longrightarrow$	b	if	$a =_{s_e} \varphi^2_0 b$

We recall now the main results concerning typed λs and λs_e:

Theorem 3.23 ([26]) 1. **Subject Reduction of λs:** If $\Gamma \vdash a : A$ and $a \to_{\lambda s} b$ then $\Gamma \vdash b : A$.

2. **SN of λs:** Every well typed term is SN in the simply typed λs-calculus.
3. **Subject Reduction of λs_e:** If $\Gamma \vdash a : A$ and $a \to_{\lambda s_e} b$ then $\Gamma \vdash b : A$.

The characterization of η-long normal forms in λs_e, to be introduced, is necessary to simplify our set of unification rules. Essentially this is the way to guarantee that meta-variables of functional type $A \to B$ are instantiated with typed λs_e-terms of the form $\lambda_A.a$.

Definition 3.24 (η-long normal form in λs_e) Let a be a λs_e-term of type $A_1 \to \ldots \to A_n \to B$ in the environment Γ and in λs_e-normal form. The **η-long normal form** of a, written a', is defined by:

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

Lemma 3.25 Definition 3.24 of η-long normal form is correct and well-founded.

PROOF. In the first case the number of occurrences of meta-variables is preserved and the size of the term is strictly decreasing. In the second case, if $p = 0$ the type is strictly decreasing and if $p \neq 0$ the number of occurrences of meta-variables is decreasing and the size of the term is strictly decreasing. In case 3 the number of occurrences of meta-variables is strictly decreasing. ∎

Definition 3.26 The **long normal form** of a λs_e-term is the η-long normal form of its $\beta\eta$-normal form.

In $\lambda\sigma$ the reduction of an η-redex may create σ-redices as in $X[\lambda.(2\ \ 1).\uparrow] \longrightarrow_{Eta} X[1.\uparrow] \longrightarrow_{VarShift} X[id] \longrightarrow_{Id} X$. Hence, to compute the long normal form, all redices, including the η-redices, should be reduced before expanding the term. That is not the case for the λs_e-calculus. In fact, by checking rule by rule, one can easily verify that no η-reduction may generate new s_e-redices. Then one could replace in the previous definition $\beta\eta$ with β. But the use of η-reduction, being it unessential at all, makes the unification process more efficient as it is explained after introduction of the λs_e-unification rules (Definition 5.2).

As in the $\lambda\sigma$-calculus, two λs_e-terms are $\beta\eta$-equivalent if and only if they have the same long normal form. Subsequently we present characterizations of λs_e-normal terms whose main operators are either σ or φ (i.e., of type 3. and 4. in Corollary 3.12). This is essential in order to simplify our presentation of the unification rules and of *Flex-Flex* equations. Our characterization is similar to that of [27] and is obtained by observing when arithmetic restrictions for the application of the transition rules of the λs_e-calculus do not hold. For instance, in order to apply a *φ-φ-transition* rule to reduce a term of the form $\varphi_k^i(\varphi_l^j a)$, we need either $l + j \leq k$ or $l \leq k < l + j$ as such a rule does not apply if $l + j > k$ and ($l > k$ or $k < l + j$) or, equivalently, if $l > k$. Note that there are no other rules to reduce, at root position, a term of this form.

Observe firstly that by the λs_e rewrite rules left arguments of the σ operator or arguments of φ operators at λs_e-normal terms are neither applications nor abstractions nor de Bruijn indices. For instance, $\varphi_i^j(a\ \ b) \to (\varphi_k^i a\ \ \varphi_k^i b)$, $(a\ \ b)\sigma^i c \to (a\sigma^i c\ \ b\sigma^i c)$. Then the sole possibility is to have as a left argument a meta-variable. Thus one has to consider terms alternating sequences of operators φ and σ whose left innermost argument is a meta variable; consider, for instance, the term $((\varphi^{j_3}_{i_3}((\varphi^{j_1}_{i_1}X)\sigma^{i_2}\cdot))\sigma^{i_4}\cdot)\sigma^{i_5}\cdot$, where right arguments of the operator σ are denoted by "$\cdot$".

Definition 3.27 Consider a λs_e-normal term t whose leading operator is either σ or φ and whose left innermost meta-variable is X. Denote by $\psi^{j_k}_{i_k}$ the operator at position k following the sequence of operators φ and σ, considering only left arguments of the σ operators, in the order innermost outermost. Additionally, if $\psi^{j_k}_{i_k}$ corresponds to an operator φ then j_k and i_k denote its super and subscripts, respectively and if $\psi^{j_k}_{i_k}$ corresponds to an operator σ then $j_k = 0$ and i_k denotes its superscript. Let a_k denote the corresponding right argument of the k^{th} operator if $\psi^{j_k}_{i_k} = \sigma^{i_k}$ and the empty argument if $\psi^{j_k}_{i_k} = \varphi^{j_k}_{i_k}$. The **skeleton** of t, written $sk(t)$, is defined as: $\psi^{j_p}_{i_p}\ldots\psi^{j_1}_{i_1}(X, a_1, \ldots, a_p)$.

Example 3.28 Let $((\varphi^{j_3}_{i_3}((\varphi^{j_1}_{i_1}X)\sigma^{i_2}a))\sigma^{i_4}b)\sigma^{i_5}c$ be a λs_e-normal term. Then its representation as a skeleton is given by $\psi^0_{i_5}\psi^0_{i_4}\psi^{j_3}_{i_3}\psi^0_{i_2}\psi^{j_1}_{i_1}(X, a, b, c)$.

In the sequel, for a λs_e-normal term whose leading operator is either φ or σ we will eventually abuse its skeleton representation $sk(t)$. Thus, for instance, for a λs_e-term a we can write $a \to^* sk(t)$ representing $a \to^* t$ or $a =_{s_e} sk(t)$ representing $a =_{s_e} t$.

Lemma 3.29 Let t be a λs_e-normal term whose leading operator is either σ or φ and whose skeleton is $\psi^{j_p}_{i_p}\ldots\psi^{j_1}_{i_1}(X, a_1, \ldots, a_p)$. Successive scripts i_k and i_{k+1} satisfy the following:

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

PROOF. By simple analysis of the arithmetic constraints at the λs_e rewrite rules. ∎

4 Unification in the $\lambda\sigma$-calculus

In this section we recall higher order unification in the $\lambda\sigma$-calculus as originally introduced in [16]. Another approach of higher order unification by explicit substitution was presented by Lescanne, Benaissa and Briaud in [31] and based on the $\lambda\upsilon$-calculus of [7]. The $\lambda\upsilon$-calculus preserves strong normalization but its confluence is restricted to closed terms. [31] informally suggests to close terms before unification is realized.

The problem to be considered is how to solve equational systems on typed $\lambda\sigma$-terms (i.e., in $\Lambda(\mathcal{X}, \mathcal{Y})$) modulo the equational theory of $\lambda\sigma$. Equational systems are restricted to be on substitution-closed terms, because of the fact that $\lambda\sigma$ is CR on terms without substitution variables, but non CR on open terms (i.e., when substitution variables are admitted). Since the main goal is to provide a mechanism to solve unification problems in the λ-calculus this restriction is not relevant.

Definition 4.1 Let $\mathcal{T}(\mathcal{F}, \mathcal{X})$ be a term algebra over a set of function symbols $\mathcal{F}$ and a countable set of variables $\mathcal{X}$ and let $\mathcal{A}$ be an $\mathcal{F}$-algebra. An **$\langle\mathcal{F}, \mathcal{X}, \mathcal{A}\rangle$-unification problem**, for short **unification problem**, is a first order formula without universal quantifier nor negation whose atoms are of the form $\mathbb{F}, \mathbb{T}$ and $s =^?_{\mathcal{A}} t$, where both

$s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. Unification problems will be written as disjunctions of existentially quantified conjunctions of atomic equational unification problems

$$D = \bigvee_{j \in J} \exists \vec{w}_j \bigwedge_{i \in I_j} s_i =^?_{\mathcal{A}} t_i$$

When there is a sole disjunctor, the unification problem is called a **unification system**. The variables in the set $\vec{w}$ in a unification system $P = \exists \vec{w} \bigwedge_{i \in I} s_i =^?_{\mathcal{A}} t_i$ are called **bound** and denoted $\mathcal{B}var(P)$, while those occurring in s_i's and t_i's are called **free** and denoted $\mathcal{F}var(P)$. $\mathbb{T}$ and $\mathbb{F}$ stand for the empty conjunction and disjunction, respectively. The empty disjunction, corresponds to an unsatisfiable problem.

Definition 4.2 A **unifier** of an $\langle \mathcal{F}, \mathcal{X}, \mathcal{A} \rangle$-unification system $\exists \vec{w} \bigwedge_{i \in I} s_i =^?_{\mathcal{A}} t_i$ is a grafting σ where $\mathcal{A} \models \exists \vec{w} \bigwedge_{i \in I} s_i \sigma_{\setminus \vec{w}} = t_i \sigma_{\setminus \vec{w}}$ and $\sigma_{\setminus \vec{w}}$ denotes the restriction of σ to the domain $\mathcal{X} \setminus \vec{w}$.
A unifier of an $\langle \mathcal{F}, \mathcal{X}, \mathcal{A} \rangle$-unification problem, $\vee_{j \in J} \exists \vec{w}_j \wedge_{i \in I_j} s_i =^?_{\mathcal{A}} t_i$, is a grafting σ that unifies at least one of the unification systems involved.

For simplicity, all references to the term algebra $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and to the algebra $\mathcal{A}$ are omitted, when they are clear from the context. When the algebra $\mathcal{A}$ considered is the quotient algebra over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ defined by the congruence associated with a set of equations E, i.e. $\mathcal{A} = \mathcal{T}(\mathcal{F}, \mathcal{X})/E$, then we denote $=^?_{\mathcal{A}}$ by $=^?_E$. The set of unifiers of a unification problem, D, or system, P, is denoted by $\mathcal{U}_{\mathcal{A}}(D)$ or $\mathcal{U}_{\mathcal{A}}(P)$, respectively.

Definition 4.3 Let σ, θ be grafting valuations from $\mathcal{X}$ into $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and $\mathcal{A}$ be an algebra over $\mathcal{T}(\mathcal{F}, \mathcal{X})$. θ is **more general modulo** $\mathcal{A}$ than σ, denoted $\theta \leq_{\mathcal{A}} \sigma$, if $\exists$ γ such that $\mathcal{A} \models \theta\gamma = \sigma$.

$\leq_{\mathcal{A}}$ induces a quasi ordering over the set of grafting valuations. When necessary, we restrict $\theta \leq_{\mathcal{A}} \sigma$ to a set $\mathcal{Y} \subset \mathcal{X}$ writing $\theta \leq^{\mathcal{Y}}_{\mathcal{A}} \sigma$.

Definition 4.4 Let D be an $\langle \mathcal{F}, \mathcal{X}, \mathcal{A} \rangle$-unification problem. A **complete set of unifiers** of D is a set of grafting valuations, denoted by $\mathcal{CU}_{\mathcal{A}}(D)$, such that:

1. $\mathcal{CU}_{\mathcal{A}}(D) \subset \mathcal{U}_{\mathcal{A}}(D)$ (Correctness)
2. $\forall \theta \in \mathcal{U}_{\mathcal{A}}(D) \exists \sigma \in \mathcal{CU}_{\mathcal{A}}(D)$ such that $\sigma \leq^{var(D)}_{\mathcal{A}} \theta$ (Completeness)
3. $\forall \theta \in \mathcal{CU}_{\mathcal{A}}(D)$, $Ran(\theta) \cap Dom(\theta) = \emptyset$ (Idempotency)

A **complete set of most general unifiers** of D, denoted by $\mathcal{CMGU}_{\mathcal{A}}(D)$, is a complete set of unifiers that additionally satisfies:

4. $\forall \theta, \sigma \in \mathcal{CMGU}_{\mathcal{A}}(D)$ $\theta \leq^{var(D)}_{\mathcal{A}} \sigma, \theta = \sigma$ (Minimality)

[16] presents a set of rewrite rule schemata that simplify unification problems in order to obtain the set of unifiers. The simplest are the boolean simplification rules.

Definition 4.5 The boolean **simplification rules** for unification problems are those of Table 6 modulo associativity and commutativity of the boolean conjunction and disjunction. In that table P, Q, R stand for unification problems, e for an equation and s, t for terms.

Basic decomposition rules for unification (to be defined after specializing unification notions to $\lambda\sigma$-terms) should be applied modulo boolean simplification rules.

TABLE 6. The Boolean simplification rules for unification problems

(Trivial)	$P \wedge s =^? s$	$\rightarrow$	P
(AndIdem)	$P \wedge e \wedge e$	$\rightarrow$	$P \wedge e$
(OrIdem)	$P \vee e \vee e$	$\rightarrow$	$P \vee e$
(SimpAndT)	$P \wedge \mathbb{T}$	$\rightarrow$	P
(SimpAndF)	$P \vee \mathbb{F}$	$\rightarrow$	$\mathbb{F}$
(SimpOrT)	$P \vee \mathbb{T}$	$\rightarrow$	$\mathbb{T}$
(SimpOrF)	$P \vee \mathbb{F}$	$\rightarrow$	P
(Distrib)	$P \wedge (Q \vee R)$	$\rightarrow$	$(P \wedge Q) \vee (P \wedge R)$
(Propag)	$\exists \vec{z}(P \vee Q)$	$\rightarrow$	$\exists \vec{z} P \vee \exists \vec{z} Q$
(ElimQE)	$\exists z P$	$\rightarrow$	P, if $z \notin var(P)$
(ElimBV)	$\exists z \quad z =^? t \wedge P$	$\rightarrow$	P, if $z \notin var(P) \cup var(t)$

Definition 4.6 A **$\lambda\sigma$-unification problem** P is a unification problem in the algebra $\mathcal{T}_{\lambda\sigma}(\mathcal{X})$ modulo the equational theory of $\lambda\sigma$. An **equation** of such a problem is denoted $a =^?_{\lambda\sigma} b$, where a and b are substitution-closed $\lambda\sigma$-terms of the same sort. An equation of the form $a =^?_{\lambda\sigma} a$ is called trivial. For a unification problem P, $\mathcal{T}var(P)$ denotes the set of variables of sort TERM and $\mathcal{U}_{\lambda\sigma}(P)$ denotes the set of all unifiers of P.

Definition 4.7 The **$\lambda\sigma$-unification rules** for typed $\lambda\sigma$-unification problems are given in Table 7.

Since $\lambda\sigma$ is CR and WN, the search can be restricted to η-long normal solutions that are graftings of the form $\{X/\lambda.a\}$ or $\{X/(\mathtt{n}\, a_1 \ldots a_p)\}$ and $\{X/(Z[s]a_1 \ldots a_p)\}$, when the type of X is functional respectively atomic. The rules *Normalize* and *Dec-λ*, use the fact that $\lambda\sigma$ is CR and WN to normalize equations of the form $\lambda.a =^?_{\lambda\sigma} \lambda.b$ into equations of the form $a' =^?_{\lambda\sigma} b'$. The rule *Exp-$\lambda$* generates the grafting $\{X/\lambda.Y\}$ for a variable X of type $A \rightarrow B$, where Y is a new variable of type B.

Example 4.8 Consider the unification problem $(X \;\; \mathtt{1}) =^?_{\lambda\sigma} \mathtt{1}$, where X has type $A \rightarrow A$. The rule *Exp-λ* takes a new variable Y of type A and by the grafting $\{X/\lambda_A.Y\}$ the problem is transformed into $(\lambda.Y \;\; 1) =^?_{\lambda\sigma} 1$ that β-reduces to $Y[\mathtt{1}.id] =^?_{\lambda\sigma} 1$.

Since Y has an atomic type A, a normal solution can only be a grafting of the form $\{Y/(\mathtt{n}\, a_1 \ldots a_p)\}$ or $\{Y/(Z[s]a_1 \ldots a_p)\}$. Grafting valuations of the second form are not solutions because normal forms of terms of the form $(Z[s]a_1 \ldots a_p)[\mathtt{1}.id]$ cannot be 1. Then all solutions should be of the first form. Performing the corresponding grafting $\{Y/(\mathtt{n}\; Y_1 \ldots Y_p)\}$, where $Y_1, \ldots, Y_p$ are new variables. Observe that $\mathtt{n}$ can only be $\mathtt{1}$ or $\mathtt{2}$ (equivalently, $\mathtt{1}[\uparrow]$), because in the other case the head in the reduction of $(\mathtt{n}\, a_1 \ldots a_p)[\mathtt{1}.id]$ is $\mathtt{n}-1$. For terms with heads $\mathtt{1}$ and $\mathtt{2}$ we have: $(1\; a_1 \ldots a_p)[\mathtt{1}.id] \rightarrow^* (1\; a_1[\mathtt{1}.id] \ldots a_p[\mathtt{1}.id])$ and $(\mathtt{1}[\uparrow]\; a_1 \ldots a_p)[\mathtt{1}.id] \rightarrow^* (\mathtt{1}[\uparrow][\mathtt{1}.id]\; a_1[\mathtt{1}.id] \ldots a_p[\mathtt{1}.id]) \rightarrow (\mathtt{1}[\uparrow \circ (\mathtt{1}.id)]\; a_1[\mathtt{1}.id] \ldots a_p[\mathtt{1}.id]) \rightarrow^* (1\; a_1[\mathtt{1}.id] \ldots a_p[\mathtt{1}.id])$.

For an equation of the form $X[a_1 \ldots a_p. \uparrow^n] =^?_{\lambda\sigma} (\mathtt{m}\; b_1 \ldots b_q)$, where X has an atomic type A, solutions can only be grafting valuations of the form $\{X/(\mathtt{r}\; c_1 \ldots c_k)\}$, where $\mathtt{r} \in \{1, \ldots, p\} \cup \{m - n + p\}$. *Exp-App* advances in direction towards this solution.

TABLE 7. The $\lambda\sigma$-unification rules

(Dec-λ)	$P \wedge \lambda_A.a =^?_{\lambda\sigma} \lambda_A.b \quad \to \quad P \wedge a =^?_{\lambda\sigma} b$
(Dec-App)	$P \wedge (\mathtt{n}\; a_1 \dots a_p) =^?_{\lambda\sigma} (\mathtt{n}\; b_1 \dots b_p) \quad \to \quad P \bigwedge_{i=1..p} a_i =^?_{\lambda\sigma} b_i$
(App-Fail)	$P \wedge (\mathtt{n}\; a_1 \dots a_p) =^?_{\lambda\sigma} (\mathtt{m}\; b_1 \dots b_q) \quad \to \quad \mathbb{F} \qquad \text{if } \mathtt{n} \neq \mathtt{m}$
(Exp-λ)	$P \quad \to \quad \exists(Y \text{ where } A.\Gamma \vdash Y : B), P \wedge X =^?_{\lambda\sigma} \lambda_A.Y$ if $(\Gamma \vdash X : A \to B) \in \mathcal{T}var(P), Y \notin \mathcal{T}var(P)$, and X is an unsolved variable
(Exp-App)	$P \wedge X[a_1 \dots a_p . \uparrow^n] =^?_{\lambda\sigma} (\mathtt{m}\; b_1 \dots b_q) \quad \to$ $P \wedge X[a_1 \dots a_p . \uparrow^n] =^?_{\lambda\sigma} (\mathtt{m}\; b_1 \dots b_q) \;\wedge$ $\bigvee_{r \in R_p \cup R_i} \exists H_1, \dots, H_k, X =^?_{\lambda\sigma} (\mathtt{r}\; H_1 \dots H_k)$ • if X has an atomic type and is not solved • $H_1, \dots, H_k$ are variables of appropriate types, not occurring in P and the environments $\Gamma_{H_i} = \Gamma_X$ • $R_p \subseteq \{1, \dots, p\}$ such that $(\mathtt{r}\; H_1 \dots H_k)$ has the right type • $R_i = \{m - n + p\}$ if $m \geq n + 1$ else $\emptyset$
(Replace)	$P \wedge X =^?_{\lambda\sigma} a \quad \to \quad \{X/a\}P \wedge X =^?_{\lambda\sigma} a$ if $X \in \mathcal{T}var(P), X \notin \mathcal{T}var(a)$ and $a \in \mathcal{X} \Rightarrow a \in \mathcal{T}var(P)$
(Normalize)	$P \wedge a =^?_{\lambda\sigma} b \quad \to \quad P \wedge a' =^?_{\lambda\sigma} b'$ if a or b is not in long normal form, $a' = \begin{cases} \text{the long normal form of } a & \text{if } a \text{ is an unsolved variable} \\ a & \text{otherwise} \end{cases}$ b' is defined from b similarly to a' from a.

During the unification process the rule *Replace* simply propagates, to the current unification problem, the grafting $\{X/a\}$ corresponding to equations $X =^?_{\lambda\sigma} a$ previously added by the application of the other unification rules.

Definition 4.9 A unification system P is a **$\lambda\sigma$-solved form** if all its meta-variables are of atomic type and it is a conjunction of non trivial equations of the following forms:

(*Solved*) $X =^?_{\lambda\sigma} a$, where the variable X does not occur anywhere else in P and a is in long normal form. Both X and $X =^?_{\lambda\sigma} a$ are said to be **solved** in P.
(*Flex-Flex*) non solved equations of the form $X[a_1 \dots a_p . \uparrow^n] =^?_{\lambda\sigma} Y[a'_1 \dots a'_{p'} . \uparrow^{n'}]$, where $X[a_1 \dots a_p . \uparrow^n]$ and $Y[a'_1 \dots a'_{p'} . \uparrow^{n'}]$ are long normal terms with X and Y of atomic type.

In the previous definition some of the scripts p, p', n, n' may be zero.

Example 4.10 Consider the equation $X =^?_{\lambda\sigma} Y[X . \uparrow]$. This is a *flex-flex* equation, but the variable X is unsolved since it occurs in the right-hand side of the equation. Observe that the left-hand side can be written as $X[id]$. The same holds for $X[\uparrow^3] =^?_{\lambda\sigma} Y[\mathtt{1} . \uparrow]$.

Since solved forms appearing in a system P define straightforwardly the binding between the variables that do not appear anywhere else in P and the terms (in long normal form), proving that *flex-flex* equations have unifiers one obtains that any $\lambda\sigma$-solved form has $\lambda\sigma$-unifiers.

[16] showed that: deduction by the $\lambda\sigma$-unification rules of a well typed equation gives rise only to well typed equations, $\mathbb{T}$ and $\mathbb{F}$; solved problems are normalized for the $\lambda\sigma$-unification rules; a conjunction of equations irreducible by the $\lambda\sigma$-unification rules is a solved system; and the $\lambda\sigma$-unification rules are correct and complete.

5 Unification in the λs_e-calculus

The main characteristics of the typed $\lambda\sigma$-calculus needed in the development of the unification method of the previous section are its weak normalization and confluence (Proposition 3.20) and its characterization of normal forms (Definition 3.21).

Definition 5.1 A **λs_e-unification problem** P is a unification problem in the algebra $\mathcal{T}_{\lambda s_e}(\mathcal{X})$ modulo the equational theory presented by λs_e. An **equation** of such a problem is denoted $a =^?_{\lambda s_e} b$, where a and b are two λs_e-terms of the same sort. An equation is called trivial when of the form $a =^?_{\lambda s_e} a$. The set of meta-variables in a unification problem P is denoted $\mathcal{T}var(P)$. The set of all unifiers of a problem P is denoted $\mathcal{U}_{\lambda s_e}(P)$.

Definition 5.2 The **λs_e-unification rules** for typed λs_e-unification problems are given in Table 8.

Here is how the λs_e-unification rules of Table 8 simplify λs_e-unification problems:

Since λs_e is CR and WN, the search can be restricted to η-long normal solutions that are graftings binding functional variables into η-long normal terms of the form $\lambda.a$ and atomic variables into η-long normal terms of the form $(\mathtt{k}\ b_1 \dots b_p)$ or $a\sigma^i b$ or $\varphi^i_k a$, where in the first case $\mathtt{k}$ could be omitted and p could be zero. Use of the η rule is important to reduce the number of cases (or unification rules) to be considered when defining the unification algorithm, but as for the $\lambda\sigma$-calculus, one can develop a HOU method based on the β-conversion alone [16]. This is not surprising since the original Huet's algorithm was developed only for the β-conversion. The rules *Normalize* and *Dec-λ*, use the fact that λs_e is CR and WN to normalize equations of the form $\lambda.a =^?_{\lambda s_e} \lambda.b$ in equations of the form $a' =^?_{\lambda s_e} b'$.

During the unification process the rule *Replace* simply propagates, to the current problem, the grafting $\{X/a\}$ corresponding to equations $X =^?_{\lambda s_e} a$ previously added by the application of the other unification rules.

The rule *Exp-λ* generates the grafting $\{X/\lambda.Y\}$ for a variable X of type $A \to B$, where Y is a new variable of type B.

Equations of the form $(\mathtt{n}\ a_1 \dots a_p) =^?_{\lambda s_e} (\mathtt{m}\ b_1 \dots b_q)$ are transformed by the rules *Dec-App* and *App-Fail* into the empty disjunction when $\mathtt{n} \neq \mathtt{m}$ (as there are no solution), or into the conjunction $\bigwedge_{i=1..p} a_i =^?_{\lambda s_e} b_i$, when $\mathtt{n} = \mathtt{m}$ (note that terms of the form $(\mathtt{n}\ a_1 \dots a_p)$ include those where $\mathtt{n}$ is omitted or $p = 0$).

Example 5.3 Consider the unification problem $(\lambda.(\lambda.(X\ \ \mathtt{2})\ \ \mathtt{1})\ \ Y) =^?_{\lambda s_e} (\lambda.(Z\ \ \mathtt{1})\ \ U)$ where X, Y, Z and U are meta-variables.

Then $(\lambda.(\lambda.(X\ \ \mathtt{2})\ \ \mathtt{1})\ \ \ Y) \to^* (\lambda.(X\sigma^1\mathtt{1}\ \ \mathtt{2}\sigma^1\mathtt{1})\ \ \ Y) \to^* (\lambda.(X\sigma^1\mathtt{1}\ \ \mathtt{1})\ \ \ Y) \to^*$ $((X\sigma^1\mathtt{1})\sigma^1 Y\ \ \mathtt{1}\sigma^1 Y) \to^* ((X\sigma^2 Y)\sigma^1(\mathtt{1}\sigma^1 Y)\ \ \varphi^1_0 Y) \to ((X\sigma^2 Y)\sigma^1(\varphi^1_0 Y)\ \ \varphi^1_0 Y)$ and

TABLE 8. The λs_e-unification rules

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

$(\lambda.(Z\ \ \mathtt{1})\ \ U) \rightarrow^* (Z\sigma^1 U\ \ \mathtt{1}\sigma^1 U) \rightarrow (Z\sigma^1 U\ \ \varphi^1_0 U)$. Hence:

$$\begin{array}{lll}
(\lambda.(\lambda.(X\ \ \mathtt{2})\ \ \mathtt{1})\ \ Y) =^?_{\lambda s_e} (\lambda.(Z\ \ \mathtt{1})\ \ U) & & \rightarrow_{Normalize} \\
((X\sigma^2 Y)\sigma^1(\varphi^1_0 Y)\ \ \varphi^1_0 Y) =^?_{\lambda s_e} (Z\sigma^1 U\ \ \varphi^1_0 U) & & \rightarrow_{Dec-App} \\
(X\sigma^2 Y)\sigma^1(\varphi^1_0 Y) =^?_{\lambda s_e} Z\sigma^1 U\ \wedge\ \varphi^1_0 Y =^?_{\lambda s_e} \varphi^1_0 U & & \rightarrow_{Dec-\varphi} \\
(X\sigma^2 Y)\sigma^1(\varphi^1_0 Y) =^?_{\lambda s_e} Z\sigma^1 U\ \wedge\ Y =^?_{\lambda s_e} U & &
\end{array}$$

Observe that solutions of $Y =^?_{\lambda s_e} U$ are graftings of the form $\{Y/V, U/V\}$. Additionally, a variety of solutions can be given for $(X\sigma^2 Y)\sigma^1(\varphi^1_0 Y) =^?_{\lambda s_e} Z\sigma^1 Y$: take $\{X/\mathtt{n}\}$; thus if $n \geq 2$, $\{Z/\mathtt{n-1}\}$.

Note that the equations of this example correspond to those of $\lambda\sigma$-unification: $(\lambda.(\lambda.(X\ \ \mathtt{2})\ \ \mathtt{1})\ \ Y) =^?_{\lambda\sigma} (\lambda.(Z\ \ \mathtt{1})\ \ U) \rightarrow_{Normalize} (X[Y.Y.id]\ \ Y) =^?_{\lambda\sigma} (Z[U.id]\ \ U)$, can be decomposed into $X[Y.Y.id] =^?_{\lambda\sigma} Z[U.id]\ \wedge\ Y =^?_{\lambda\sigma} U$. Note that since X and Z are meta-variables of functional type, $X[Y.Y.id] =^?_{\lambda\sigma} Z[U.id]$ is not *flex-flex*.

In the $\lambda\sigma$-calculus, the unification rule *Exp-App* advances in direction towards solutions for equations of the form $X[a_1 \ldots a_p. \uparrow^n] =^?_{\lambda s_e} (\mathtt{m}\ b_1 \ldots b_q)$, where X is an unsolved variable of an atomic type, say A. Solutions are grafting valuations of the form $\{X/(\mathtt{r}\ c_1 \dots c_k)\}$, where $\mathtt{r} \in \{1, \dots, p\} \cup \{m-n+p\}$. The λs_e-unification rule *Exp-App* develops the analogous role for unification problems over the λs_e-calculus.

It is important to note that explicit use of λs_e-normal forms in the unification rule *Exp-App* is not essential. It is done however, with the sole objective of simplifying the case analysis presented in the definition of the rule and thus in its completeness proof. In fact, this can be dropped from the general presentation of the λs_e-unification procedure and be subsequently incorporated as an efficient unification strategy, where before applying the *Exp-App* rule, unification problems are normalized. Normalization before applying other unification rules is usually proposed in any unification strategy including the $\lambda\sigma$-unification approach in [16]. This is a consequence of the fundamental commutation theorem between substitution and reduction in λ-calculus.

Now, we give important properties for the λs_e-unification rules (Definition 5.2).

Lemma 5.4 (Well-typedness) Deduction by the λs_e-unification rules of a well typed equation gives rise only to well typed equations, $\mathbb{T}$ and $\mathbb{F}$.

PROOF. By analyzing, rule by rule, the types of the resulting transformed equation.

- *Normalize*: this is consequence of the fact that the rewriting system λs_e is well typed and of the correctness of the definition of long normal forms.
- *Dec-λ*: if $\lambda_A.a =^?_{\lambda s_e} \lambda_A.b$ is well typed then the types of a and b coincide.
- *App-Fail*: obvious.
- *Dec-App, Dec-φ*: if $(\mathtt{n}\ a_1 \dots a_p) =^?_{\lambda s_e} (\mathtt{n}\ b_1 \dots b_p)$ is well typed then so is $(a_1 \dots a_p) =^?_{\lambda s_e} (b_1 \dots b_p)$. Without loss of generality we can suppose that a_i and b_i are normalized concluding that the equations $a_i =^?_{\lambda s_e} b_i$ are well typed. Well typing of the unification rule Dec-φ is proved similarly.
- *Exp-λ*: by definition of the rule, since $X\ :\ \Gamma \vdash A \to B$ and $Y\ :\ A.\Gamma \vdash B$ then by the typing rule **L1**-λ, $\lambda_A.Y\ :\ \Gamma \vdash A \to B$. Hence, $X =^?_{\lambda s_e} \lambda_A.Y$ is well typed.
- *Exp-App*: we present a simple sketch of the inductive proof on q and p. We omit the case in which $\mathtt{m}$ occurs and consider a simple equation of the form $X\sigma^i b =^?_{\lambda s_e} (b_1\ \ b_2)$ where X is an atomic meta-variable. We have $\Gamma_{<i}.B.\Gamma_{\geq i} \vdash X\ :\ A$ and $\Gamma_{\geq i} \vdash a\ :\ B$ and by the typing rule **Ls1**-σ, for some environment Γ and types A and B, $\Gamma \vdash X\sigma^i b\ :\ A$. By assumption $X\sigma^i b =^?_{\lambda s_e} (b_1\ \ b_2)$ is well typed thus $\Gamma \vdash (b_1\ \ b_2)\ :\ A$. By the typing rule **Ls1**-*App* we have some type C such that $\Gamma \vdash b_1\ :\ C \to A$ and $\Gamma \vdash b_2\ :\ C$. Then by assumption of the *Exp-App* unification rule, variables H_1, H_2 can be appropriately selected such that $\Gamma \vdash H_1\ :\ C \to A$ and $\Gamma \vdash H_2\ :\ C$. Hence the equation $X =^?_{\lambda s_e} (H_1\ \ H_2)$ is well typed. The proof is finished by analyzing combined cases of successive σ and φ operators and by completing in the straightforward form the inductive reasoning.
- *Replace*: well typing of equations is preserved because, as $P\ \wedge\ X =^?_{\lambda s_e} a$ well typed, by replacing X with a in P, types are not changed. This is a consequence of the compatibility between grafting and typing. ∎

Example 5.5 We present three different unification problems and corresponding equations, to be treated with the *Exp-App* $\lambda\sigma$- and λs_e-unification rules, which result from the application of both unification methods. The reader is invited to complete the computations.

1. Consider the problem $(\lambda.(\lambda.(X\ \mathtt{2})\ \mathtt{1})\ Y) =^{?} (\lambda.(\lambda.V\ \mathtt{1})\ U)$. Related equations to be treated by applying the corresponding *Exp-App* unification rules are: $(X[Y.Y.id]\ Y) =^{?}_{\lambda\sigma} V[U.U.id]$ and $((X\sigma^2 Y)\sigma^1(\varphi^1_0 Y)\ \varphi^1_0 Y) =^{?}_{\lambda s_e} (V\sigma^2 U)\sigma^1(\varphi^1_0 U)$. Solutions are reached after applying *Exp-App* unification rules with $V =^{?}_{\lambda\sigma} (V_1\ V_2)$ and $V =^{?}_{\lambda s_e} (V_1\ V_2)$, where V_1, V_2 are new variables.
2. From the problem $\lambda.(\lambda.(Y\ \mathtt{1})\ \lambda.(X\ \mathtt{1})) =^{?} \lambda.(\lambda.V\ \lambda.W)$ we reach the equations: $(Y[\lambda.(X\ \mathtt{1}).id]\ \ \lambda.(X\ \mathtt{1})) =^{?}_{\lambda\sigma} V[\lambda.W.id]$ and $(Y\sigma^1\lambda.(X\ \mathtt{1})\ \ \lambda.(\varphi^1_1\ \mathtt{1})) =^{?}_{\lambda s_e} V\sigma^1\lambda.W$ After applying the corresponding *Exp-App* rules, with $V =^{?}_{\lambda\sigma} (V_1\ V_2)$ and $V =^{?}_{\lambda s_e} (V_1\ V_2)$, new equations appear: $\lambda.(X\ \mathtt{1}) =^{?}_{\lambda\sigma} V_2[\lambda.(X\ \mathtt{1}).id]$ and $\lambda.(\varphi^1_1 X\ \mathtt{1}) =^{?}_{\lambda s_e} V_2\sigma^1\lambda.(X\ \mathtt{1})$. Solutions result by selecting the case $V_2 =^{?} \mathtt{1}$.
3. Consider the problem $(\lambda.(X\ \lambda.W)\ \mathtt{1}) =^{?} (\lambda.(Z\ \lambda.(U\ \mathtt{1}))\ V)$. Related equations to be treated by the application of *Exp-App* unification rules are: $W[\mathtt{1}.\mathtt{1}[\uparrow].\uparrow] =^{?}_{\lambda\sigma} (U[\mathtt{1}.V[\uparrow].\uparrow]\ \mathtt{1})$ and $W\sigma^2\mathtt{1} =^{?}_{\lambda s_e} (U\sigma^2 V\ \mathtt{1})$. Solutions are found after applying *Exp-App* unification rules with $W =^{?} (W_1\ W_2)$.

Before formalizing *flex-flex* equations in λs_e, we give an example where the application of *Exp-λ* and *Exp-App* is essential. Solutions are those of the *flex-flex* equations.

Example 5.6 (Continuing Example 5.3 and 5.5 1.)
Consider the problem $(\lambda.(\lambda.(X\ \mathtt{2})\ \mathtt{1})\ Y) =^{?}_{\lambda\sigma} (\lambda.(Z\ \mathtt{1})\ U)$, where now we make considerations about the types of meta-variables. Let Y and U be of type A and X and Z be of type $A \to A$. In the $\lambda\sigma$-unification setting:

$$\begin{array}{ll}
(\lambda.(\lambda.(X\ \mathtt{2})\ \mathtt{1})\ Y) =^{?}_{\lambda\sigma} (\lambda.(Z\ \mathtt{1})\ U) & \to_{Exp-\lambda, Replace} \\
(\lambda.(\lambda.(X\ \mathtt{2})\ \mathtt{1})\ Y) =^{?}_{\lambda\sigma} (\lambda.(\lambda.V\ \mathtt{1})\ U) \ \wedge\ Z =^{?}_{\lambda\sigma} \lambda.V & \to_{Normalize} \\
(X[Y.Y.id]\ Y) =^{?}_{\lambda\sigma} V[U.U.id] \ \wedge\ Z =^{?}_{\lambda\sigma} \lambda.V &
\end{array}$$

where V is a new meta-variable of type A. The interesting step in the whole process is the application of the *Exp-App* $\lambda\sigma$-unification rule to the first equation, which, by case analysis, could transform this into the search for solutions of $(X[Y.Y.id]\ Y) =^{?}_{\lambda\sigma} V[U.U.id] \ \wedge\ V =^{?}_{\lambda\sigma} (V_1\ V_2)$ that by *Replace* and *Normalize* gives $(X[Y.Y.id]\ Y) =^{?}_{\lambda\sigma} (V_1[U.U.id]\ V_2[U.U.id]) \ \wedge\ V =^{?}_{\lambda\sigma} (V_1\ V_2)$ and, finally, by decomposition and *Replace* gives the unification problem: $X[V_2[U.U.id].V_2[U.U.id].id] =^{?}_{\lambda\sigma} V_1[U.U.id] \ \wedge\ Y =^{?}_{\lambda\sigma} V_2[U.U.id] \ \wedge\ V =^{?}_{\lambda\sigma} (V_1\ V_2)$. Note that both X and V_1 are of type $A \to A$. Then by, firstly, applying twice *Exp-λ* introducing new equations $X =^{?}_{\lambda\sigma} \lambda.X'$ and $V_1 =^{?}_{\lambda\sigma} \lambda.V_1'$, where X' and V_1' are fresh atomic meta-variables; afterwards, by applying twice *Replace* and, finally, by applying *Normalize* and *Dec-λ* we obtain $X'[\mathtt{1}.V_2[U[\uparrow].U[\uparrow].\uparrow].V_2[U[\uparrow].U[\uparrow].\uparrow].\uparrow] =^{?}_{\lambda\sigma} V_1'[\mathtt{1}.U[\uparrow].U[\uparrow].\uparrow] \ \wedge\ Y =^{?}_{\lambda\sigma} V_2[U.U.id] \ \wedge\ V =^{?}_{\lambda\sigma} (V_1\ V_2) \ \wedge\ X =^{?}_{\lambda\sigma} \lambda.X' \ \wedge\ V_1 =^{?}_{\lambda\sigma} \lambda.V_1'$. The *flex-flex* equation has: $\{X/\lambda.X_1, Z/\lambda.(\lambda.X_1\ \mathtt{2}), Y/X_2, U/X_2\}$ and $\{X/\lambda.X_1, Z/\lambda.(\lambda.X_1\ \mathtt{1}), Y/X_2, U/X_2\}$ as the obvious solutions. In the λs_e-setting (taking $P_1 = Z =^{?}_{\lambda s_e} \lambda.V \wedge V =^{?}_{\lambda s_e} (V_1\ V_2)$):

$$\begin{array}{ll}
(\lambda.(\lambda.(X\ \mathtt{2})\ \mathtt{1})\ Y) =^{?}_{\lambda s_e} (\lambda.(Z\ \mathtt{1})\ U) & \to_{Exp-\lambda, Replace} \\
(\lambda.(\lambda.(X\ \mathtt{2})\ \mathtt{1})\ Y) =^{?}_{\lambda s_e} (\lambda.(\lambda.V\ \mathtt{1})\ U) \ \wedge\ Z =^{?}_{\lambda s_e} \lambda.V & \to_{Normalize} \\
((X\sigma^2 Y)\sigma^1(\varphi^1_0 Y)\ \ \varphi^1_0 Y) =^{?}_{\lambda s_e} (V\sigma^2 U)\sigma^1(\varphi^1_0 U) \ \wedge\ Z =^{?}_{\lambda s_e} \lambda.V & \to_{Exp-App} \\
((X\sigma^2 Y)\sigma^1(\varphi^1_0 Y)\ \ \varphi^1_0 Y) =^{?}_{\lambda s_e} (V\sigma^2 U)\sigma^1(\varphi^1_0 U) \ \wedge\ P_1 & \to_{Replace} \\
((X\sigma^2 Y)\sigma^1(\varphi^1_0 Y)\ \ \varphi^1_0 Y) =^{?}_{\lambda s_e} ((V_1\ V_2)\sigma^2 U)\sigma^1(\varphi^1_0 U) \ \wedge\ P_1 & \to_{Normalize} \\
((X\sigma^2 Y)\sigma^1(\varphi^1_0 Y)\ \ \varphi^1_0 Y) =^{?}_{\lambda s_e} ((V_1\sigma^2 U)\sigma^1(\varphi^1_0 U)\ \ (V_2\sigma^2 U)\sigma^1(\varphi^1_0 U)) \ \wedge\ P_1 & \to_{Dec-App} \\
(X\sigma^2 Y)\sigma^1(\varphi^1_0 Y) =^{?}_{\lambda s_e} (V_1\sigma^2 U)\sigma^1(\varphi^1_0 U) \ \wedge\ \varphi^1_0 Y =^{?}_{\lambda s_e} (V_2\sigma^2 U)\sigma^1(\varphi^1_0 U) \ \wedge\ P_1 &
\end{array}$$

As for the the $\lambda\sigma$ case, applying twice *Exp-λ* and *Replace* and then *Normalize* and *Dec-λ* we obtain

$(X'\sigma^3 Y)\sigma^2(\varphi_0^1 Y) =^?_{\lambda s_e} (V_1'\sigma^3 U)\sigma^2(\varphi_0^1 U) \ \wedge\ \varphi_0^1 Y =^?_{\lambda s_e} (V_2\sigma^2 U)\sigma^1(\varphi_0^1 U) \ \wedge$
$X =^?_{\lambda s_e} \lambda.X' \ \wedge\ V_1 =^?_{\lambda s_e} \lambda.V_1'$.

From the first flex-flex equation we obtain, by simple decomposition, the partial solution $\{X/\lambda.X_1, Z/\lambda.(\lambda.X_1\ \ V_2), Y/X_2, U/X_2\}$. To obtain a complete solution it remains to resolve the *flex-flex* equation $\varphi_0^1 X_2 =^?_{\lambda s_e} (V_2\sigma^2 X_2)\sigma^1(\varphi_0^1 X_2)$. Observe that $(\mathtt{1}\sigma^2 X_2)\sigma^1(\varphi_0^1 X_2) \to \mathtt{1}\sigma^1(\varphi_0^1 X_2) \to \varphi_0^1(\varphi_0^1 X_2) \to \varphi_0^1 X_2$ and also $(\mathtt{2}\sigma^2 X_2)\sigma^1(\varphi_0^1 X_2) \to (\varphi_0^2 X_2)\sigma^1(\varphi_0^1 X_2) \to \varphi_0^1 X_2$. This, analogously to the $\lambda\sigma$ setting, gives the solutions $\{X/\lambda.X_1, Z/\lambda.(\lambda.X_1\ \ \mathtt{1}), Y/X_2, U/X_2\}$ and $\{X/\lambda.X_1, Z/\lambda.(\lambda.X_1\ \ \mathtt{2}), Y/X_2, U/X_2\}$.

Definition 5.7 A unification system P is a λs_e**-solved form** if all its meta-variables are of atomic type and it is a conjunction of non trivial equations of the following forms:

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

Remark 5.8 Consider a λs_e-normal term c whose leading operator is either σ or φ and with skeleton $sk(c) = \psi_{i_p}^{j_p}\ldots\psi_{i_1}^{j_1}(X, a_1, \ldots, a_p)$. By binding X with $\mathtt{n}$, $n > i_1$, one obtains as a consequence of lemma 3.29, the normal form $t \to^* \mathtt{n} + \sum_{k=1}^{p} \mathtt{j}_k - \mathtt{p}$. We illustrate the situation with three simple cases of searching for solutions of *flex-flex* equations.

Firstly, $(\cdots((X\sigma^{i_1}a_1)\sigma^{i_2}a_2)\cdots)\sigma^{i_p}a_p =^?_{\lambda s_e} (\cdots((Y\sigma^{j_1}b_1)\sigma^{j_2}b_2)\cdots)\sigma^{j_q}b_q$ always has solutions since both its sides are λs_e-normal terms and hence the sequences $i_1, \ldots, i_p$ and $j_1, \ldots, j_q$ are strictly decreasing. Solutions are bindings $\{X/\mathtt{n}+\mathtt{p}\}$ and $\{Y/\mathtt{n}+\mathtt{q}\}$, with $n > i_1, j_1$.

Secondly, since the left-side of $(\cdots((X\sigma^{i_1}a_1)\sigma^{i_2}a_2)\cdots)\sigma^{i_p}a_p =^?_{\lambda s_e} \varphi_{k_q}^{j_q}\cdots\varphi_{k_1}^{j_1}Y$ is a λs_e-normal term, the sequence $k_1, \ldots, k_q$ is strictly decreasing. Now select n, m such that $n > i_1$, $m > k_1$ and $n - p = m + \sum_{l=1}^{q} j_l - q$, and the bindings $\{X/\mathtt{n}\}$, $\{Y/\mathtt{m}\}$.

Thirdly, for $\varphi_{k_p}^{i_p}\cdots\varphi_{k_1}^{i_1}X =^?_{\lambda s_e} \varphi_{l_q}^{j_q}\cdots\varphi_{l_1}^{j_1}Y$ select, for instance, bindings $\{X/\mathtt{n}\}$, $\{Y/\mathtt{m}\}$, such that $n > k_1, m > l_1$ and $n + \sum_{r=1}^{p} i_r - p = m + \sum_{r=1}^{q} j_r - q$.

Moreover, observe that by selecting graftings of the form $\{X/(H_1\ldots H_l)\}$ (where $H_1, \ldots, H_l$ are meta-variables of appropriate types) the term c with skeleton $sk(c)$ is split into applications of terms with identical skeletons and left innermost meta-variables $H_1, \ldots, H_m$ such that: $\psi_{i_p}^{j_p}\ldots\psi_{i_1}^{j_1}((H_1\ldots H_k), a_1, \ldots, a_p) \to^*$

$$(\psi_{i_p}^{j_p}\ldots\psi_{i_1}^{j_1}(H_1, a_1, \ldots, a_p), \ldots, \psi_{i_p}^{j_p}\ldots\psi_{i_1}^{j_1}(H_k, a_1, \ldots, a_p)).$$

Let consider for instance, $((\varphi_{i_3}^{j_3}((\varphi_{i_1}^{j_1}(H_1 \ldots H_l))\sigma^{i_2}\cdot))\sigma^{i_4}\cdot)\sigma^{i_5}\cdot \to^*$

$$(((\varphi_{i_3}^{j_3}((\varphi_{i_1}^{j_1}H_1)\sigma^{i_2}\cdot))\sigma^{i_4}\cdot)\sigma^{i_5}\cdot\ \ldots\ ((\varphi_{i_3}^{j_3}((\varphi_{i_1}^{j_1}H_l)\sigma^{i_2}\cdot))\sigma^{i_4}\cdot)\sigma^{i_5}\cdot).$$

Lemma 5.9 Any λs_e-solved form has λs_e-unifiers.

Proof. For simplicity we omit the analysis of types. Since solved forms appearing in a system P define straightforwardly bindings between variables that do not appear anywhere else in P or in terms (in long normal form), it is only necessary to prove that *flex-flex* equations have unifiers.

Let P be a system in λs_e-solved form including a *flex-flex* equation of the form $\psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(X, a_1, \dots, a_p) =^?_{\lambda s_e} \psi^{l_q}_{k_q} \dots \psi^{l_1}_{k_1}(Y, b_1, \dots, b_q)$. This equation always has solutions. Select for example bindings $\{X/\mathtt{n}, Y/\mathtt{m}\}$ such that $n > i_1, m > l_1$ and $n + \sum_{r=1}^{p} j_r - p = m + \sum_{r=1}^{q} l_r - q$ (see previous Remark 5.8). ∎

Lemma 5.10 Solved problems are normalized for the λs_e-unification rules and, conversely, if a system is a conjunction of equations that cannot be reduced by the λs_e-unification rules then it is solved.

PROOF. It is easy to verify, rule by rule, that solved and *flex-flex* equations cannot be transformed by the λs_e-unification rules. So solved forms (or problems) are in normal form for the λs_e-unification rules. Conversely, suppose P is a non solved system, then P contains an equation $a =^?_{\lambda s_e} b$ that is neither solved nor *flex-flex*. If either a or b are not in long normal form then the rule *Normalize* applies . If the equation is of the form $X =^?_{\lambda s_e} b$, where X occurs in other position at P, then the rule *Replace* applies.

The remaining cases, where both a and b are long normal terms, are subsequently listed using the characterization of λs_e-normal forms at Corollary 3.12.

Observe firstly that if a is of the form $\lambda.a'$ then, since b is a long normal term, the sole possibility to have a well typed equation is if b is of the form $\lambda.b'$ in which case rule *Dec-λ* applies.

Secondly, suppose that a is of the form $(\mathtt{k}\ a_1 \dots a_p)$. Then if b is of the form $(\mathtt{l}\ b_1 \dots b_q)$ then either *Dec-App* or *App-Fail* applies (remember here that both $\mathtt{k}$ and $\mathtt{l}$ could be omitted and p and q could be zero). If b has a leading operator σ or φ then rule *Exp-App* applies.

Finally, the remaining cases of equations between terms with main operators σ and φ are either *flex-flex* or can be reduced with rule *Dec-φ*. ∎

Definition 5.11 Let P and P' be λs_e-unification problems, let "*rule*" denote the name of a λs_e-unification rule and "$\rightarrow^{rule}$" its corresponding deduction relation over unification problems. By **correctness** and **completeness** of *rule* we understand the following:

- $P \rightarrow^{rule} P'$ implies $\mathcal{U}_{\lambda s_e}(P') \subseteq \mathcal{U}_{\lambda s_e}(P)$ *(correctness)*
- $P \rightarrow^{rule} P'$ implies $\mathcal{U}_{\lambda s_e}(P) \subseteq \mathcal{U}_{\lambda s_e}(P')$ *(completeness)*

Theorem 5.12 (Correctness and Completeness) The λs_e-rules are correct and complete.

PROOF. Firstly, we verify the correctness of all rules.

- *Dec-λ*: is correct since grafting is a congruence on λs_e-terms.
- *App-Fail*: is correct because of trivial inclusion of the empty set.
- *Dec-App, Dec-φ*: are correct because grafting is a congruence on λs_e-terms.
- *Exp-λ, Exp-App*: are correct because of the properties of the λs_e rewriting system.
- *Replace*: observe that this rule corresponds to the selection of bindings in the first order unification algorithm and its correctness is similarly proved.
- *Normalize*: is correct since normalization corresponds to simplification of terms between the same equivalence class in the λs_e-calculus.

Secondly, we verify the completeness of the rules.

- *Dec-λ*: let θ be a λs_e-unifier of an equation of the form $\lambda.a =^?_{\lambda s_e} \lambda.b$. Thus $\lambda.\theta(a) =^?_{\lambda s_e} \lambda.\theta(b)$ and since no λs_e-rule could be applied at root position of these terms then $\theta(a) =^?_{\lambda s_e} \theta(b)$.
- *Dec-App*: suppose that θ is a λs_e-unifier of $(\mathtt{n}\ a_1 \dots a_p) =^?_{\lambda s_e} (\mathtt{n}\ b_1 \dots b_p)$. Thus, because of confluence and weakly terminating properties of the λs_e rewriting system we have: $\theta(\mathtt{n}\ a_1 \dots a_p) =^?_{\lambda s_e} \theta(\mathtt{n}\ b_1 \dots b_p)$ iff $(\mathtt{n}\ \theta(a_1) \dots \theta(a_p)) =^?_{\lambda s_e} (\mathtt{n}\ \theta(b_1) \dots \theta(b_p))$ iff for all $1 \leq i \leq p$, $\theta(a_i) =^?_{\lambda s_e} \theta(b_i)$. This means that θ is a unifier of $a_1 =^?_{\lambda s_e} b_1 \wedge \dots \wedge a_p =^?_{\lambda s_e} b_p$.
- *Dec-φ*: analogous to the former case.
- *App-Fail*: it follows from the sequence of logical equivalences in the proof of completeness of rule *Dec-App* that if $n \neq m$ then there are no λs_e-unifier of $(\mathtt{n}\ a_1 \dots a_p) =^?_{\lambda s_e} (\mathtt{m}\ b_1 \dots b_p)$.
- *Exp-λ*: Let θ be a λs_e-unifier of P and $X \in \mathcal{T}var(P)$ such that $X : \Gamma \vdash A \to B$. Thus $\theta(X) = a : A \to B$ and we can assume that a is of the form $\lambda_A.b$ with $b : B$. Define θ' such that for all $Z \in Dom(\theta)$, $\theta'(Z) = \theta(Z)$ and $\theta(Y) = b$ for a new variable $Y \notin Dom(\theta)$ of type B. Then θ' is a λs_e-unifier of $P \wedge X =^?_{\lambda s_e} \lambda_A.Y$. Consequently θ is a λs_e-unifier of $\exists(Y : A.\Gamma \vdash B), P \wedge X =^?_{\lambda s_e} \lambda_A.Y$.
- *Exp-App*: consider $P \wedge \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(X, a_1, \dots, a_p) =^?_{\lambda s_e} (\mathtt{m}\ b_1 \dots b_q)$ and suppose that θ is a λs_e-unifier of this unification problem.

  Then $\theta(X) = (\mathtt{r}\, c_1 \dots c_s)$ and the interesting equation in the unification problem becomes $(\mathtt{m}\ b'_1 \dots b'_q) =^?_{\lambda s_e} \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}((\mathtt{r}\, c_1 \dots c_s), a'_1, \dots, a'_p) \to^* (\mathtt{m}\ b'_1 \dots b'_q) =^?_{\lambda s_e} (\ \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(\mathtt{r}, a'_1, \dots, a'_p)\ \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(c_1, a'_1, \dots, a'_p)\ \dots\ \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(c_s, a'_1, \dots, a'_p))$.

  Since $\psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(X, a_1, \dots, a_p)$ is the skeleton of a λs_e-normal term then the sequence $i_1, \dots, i_p$ is decreasing, being possible $i_k = i_{k+1}$ only when both the k^{th} and $k+1^{th}$ ψ's correspond to φ operators. We have two simple cases to consider: either r different from all i_k such that $\psi^{j_k}_{i_k}$ corresponds to a σ operator or $r = i_k$ for some k such that $\psi^{j_k}_{i_k} = \sigma^{i_k}$.

  In the first case, let $i_0 = \infty$ and $i_{p+1} = 0$ and suppose that $i_{k+1} < r \leq i_k$ for some $0 \geq k \geq p$ such that either $k = p$ or $\psi^{j_k}_{i_k}$ corresponds to a σ operator. Then $\psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(\mathtt{r}, a'_1, \dots, a'_p) \to^* \psi^{j_p}_{i_p} \dots \psi^{j_{k+1}}_{i_{k+1}}(\mathtt{r}, a'_{k+1}, \dots, a'_p) \to^* \mathtt{r} + \sum_{l=k+1}^{p} \mathtt{j}_l - (\mathtt{p} - \mathtt{k})$. Observe that this coincides with the definition of R_i in rule *Exp-App*; in fact, if $r = m - \sum_{l=k+1}^{p} j_l + p - k$ and $i_{k+1} < m - \sum_{l=k+1}^{p} j_l + p - k \geq i - k$ for $k = p$ or i_{k+1} corresponding to a superscript of an operator σ in the skeleton then $\psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(\mathtt{r}, a'_1, \dots, a'_p) \to^* \mathtt{m}$.

  If $r = i_k$ for some $1 \leq j \leq p$ corresponding to a σ operator, then we have the following reduction: $\psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(\mathtt{r}, a'_1, \dots, a'_p) \to^* \psi^{j_p}_{i_p} \dots \psi^{j_k}_{i_k}(\mathtt{r}, a'_k, \dots, a'_p) \to \psi^{j_p}_{i_p} \dots \psi^{j_{k+1}}_{i_{k+1}}(\varphi^{i_k}_0 a'_k, \dots, a'_p) \to^* \varphi_0^{i_k - p + k + \sum_{l=k+1}^{p} j_l} a'_k$.

  Thus, in the first case the equation becomes

$$(\mathtt{m}\ b'_1 \dots b'_q) =^?_{\lambda s_e} (\mathtt{m}\ \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(c_1, a'_1, \dots, a'_p)\ \dots\ \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(c_s, a'_1, \dots, a'_p))$$

  and in the second, $(\mathtt{m}\ b'_1 \dots b'_q) =^?_{\lambda s_e}$
  $(\varphi_0^{i_k - p + k + \sum_{l=k+1}^{p} j_l} a'_k\ \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(c_1, a'_1, \dots, a'_p)\ \dots\ \psi^{j_p}_{i_p} \dots \psi^{j_1}_{i_1}(c_s, a'_1, \dots, a'_p))$

In both cases θ is clearly solution of $\exists H_1, \dots, H_k, X =^?_{\lambda s_e} (\mathtt{r}\, H_1 \dots H_k)$, selecting $H_1, \dots, H_k$ appropriately and, consequently, it is solution of the original problem and $\bigvee_{r \in R_p \cup R_i} \exists H_1, \dots, H_k, X =^?_{\lambda s_e} (\mathtt{r}\, H_1 \dots H_k)$.

- Replace: its completeness is similarly proved to the first order case.
- Normalize: as for correctness its completeness is consequence of the fact that normalization corresponds to simplification of terms between the same equivalence class in the λs_e-calculus. ∎

6 Arithmetic properties of the λs_e-unification rules

The arithmetic constraint that naturally has resulted when defining the *Exp-App* λs_e-unification rule is more expressive than the one of the $\lambda\sigma$ HOU setting. This, jointly with an efficient arithmetic deductive method, speed up the verification of possible splittings and the search for solutions in the corresponding case analysis.

For the $\lambda\sigma$-calculus, the equation $X[a_1 \dots a_p. \uparrow^n] =^?_{\lambda\sigma} (\mathtt{m}\, b_1 \dots b_q)$ has for solutions $(\mathtt{r}\, H_1 \dots H_k)$, where $r - p + n = m$. In fact, $\mathtt{1}[\uparrow^{r-1}][a_1 \dots a_p. \uparrow^n] \to^* \mathtt{1}[\uparrow^{r-1-p+n}]$.

In [16] the $\lambda\sigma$-calculus is presented using only the de Bruijn index $\mathtt{1}$. Thus the detection of the previous kind of solutions is very inefficient. In fact, observe that since $\uparrow^n$ abbreviates $(n-1)$-compositions of $\uparrow$, finding the first component $\mathtt{1}[\uparrow^{r-1}]$ of these possible solutions can be done only after realizing a process of enumeration of the p a_i components and the $(n-1)$ $\uparrow$ of $\mathtt{n} \equiv \mathtt{1}[\uparrow^{n-1}]$. Since λs_e-terms are written using all the natural indices, one can state that searching for redices of the unification rules and determining solved and *flex-flex* equations in our unification setting are more efficient than in the language of the $\lambda\sigma$-calculus.

We show that the first numeric components of bindings for a meta-variable X of solutions of equations of the form $\psi^{j_p}_{k_p} \dots \psi^{j_1}_{k_1}(X, a_1, \dots, a_p) =^?_{\lambda s_e} (\mathtt{m}\, b_1 \dots b_q)$ are determined in a unique way.

Lemma 6.1 Let $n > k_1$, $m \le k_p$ and take a skeleton $\psi^{j_p}_{k_p} \dots \psi^{j_1}_{k_1}(X, a_1, \dots, a_p)$ of a λs_e-normal term. $\psi^{j_p}_{k_p} \dots \psi^{j_1}_{k_1}(n, a_1, \dots, a_p) \to^* n - p + \sum_{r=1}^{p} j_r > n - (k_1 - k_p + 1) \ge m$.

PROOF. Firstly, observe that since $k_1, \dots, k_p$ is a decreasing sequence, we have $n > k_1 \ge \dots \ge k_p \ge m$ and thus $k_1 - k_p < n - m$ which implies $m \le n - (k_1 - k_p + 1)$.

Secondly, observe that $\sum_{r=1}^{p} j_r \ge 0$. Thus the sole possibility to have $n - p + \sum_{r=1}^{p} j_r \le n - (k_1 - k_p + 1)$ is being $p - 1 \ge k_1 - k_p$. We consider two cases:

If $p - 1 = k_1 - k_p$ then $\psi^{j_p}_{k_p} \dots \psi^{j_1}_{k_1}(n, a_1, \dots, a_p) \to^* n - p + \sum_{r=1}^{p} j_r \ge n - p = n - (k_1 - k_p + 1) \ge m$. Moreover, observe that if there exists some operator φ, say $\psi^{j_i}_{k_i}$ in the sequence of the skeleton, then $\sum_{r=1}^{p} j_r \ge j_i > 0$ which implies $n - p + \sum_{r=1}^{p} j_r > m$. If the sequence consists only of σ operators, then $m < k_p$ and also $n - p + \sum_{r=1}^{p} j_r > m$.

If $p - 1 > k_1 - k_p$ then there exists at least one $1 \le i < p$ such that $\psi^{j_i}_{k_i} = \varphi^{j_i}_{k_i}$ and $\psi^{j_{i+1}}_{k_{i+1}} = \sigma^{k_{i+1}}$ being $k_i = k_{i+1}$. Thus $\psi^{j_{i+1}}_{k_{i+1}} \psi^{j_i}_{k_i}(n, a_i, a_{i+1}) \to \psi^{j_{i+1}}_{k_{i+1}}(n + j_i - 1, a_{i+1}) \to n + j_i + j_{i+1} - 2 \ge n - (k_i - k_{i+1})$. For each of these subsequences we have the analogous situation, obtaining for the whole sequence $n - p + \sum_{r=1}^{p} j_r > n - (k_1 - k_p + 1) \ge m$. ∎

Lemma 6.2 (Unicity) Take a skeleton $\psi^{j_p}_{k_p} \dots \psi^{j_1}_{k_1}(X, a_1, \dots, a_p)$ of a λs_e-normal term and the equation $\psi^{j_p}_{k_p} \dots \psi^{j_1}_{k_1}(X, a_1, \dots, a_p) =^?_{\lambda s_e} (\mathtt{m}\, b_1 \dots b_q)$. The first numerical component of bindings for the meta-variable X of solutions of this equations are unique.

PROOF. Observe firstly the possible cases for bindings $\{X/(\mathtt{n}\dots)\}$:

$n \le k_p$: $\psi^{j_p}_{k_p}\dots\psi^{j_1}_{k_1}(n, a_1, \dots, a_p) \to^* \psi^{j_p}_{k_p}(n, a_p)$. Since case $n = k_p$ thus $\psi^{j_p}_{k_p} = \varphi^{j_p}_{k_p}$, we have $\psi^{j_p}_{k_p}(n, a_p) \to n$.

$k_{i+1} < n \le k_i$: $\psi^{j_p}_{k_p}\dots\psi^{j_1}_{k_1}(n, a_1, \dots, a_p) \to^* \psi^{j_p}_{k_p}\dots\psi^{j_i}_{k_i}(n, a_i, \dots, a_p)$. Since case $n = k_i$ we have $\psi^{j_i}_{k_i} = \varphi^{j_i}_{k_i}$, then in both cases: $n = k_i$ and $n < k_i$, $\psi^{j_p}_{k_p}\dots\psi^{j_i}_{k_i}(n, a_i, \dots, a_p) \to \psi^{j_p}_{k_p}\dots\psi^{j_{i+1}}_{k_{i+1}}(n, a_{i+1}, \dots, a_p) \to^* n - (p-i) + \sum_{r=i+1}^{p} j_r$.

$k_1 < n$: $\psi^{j_p}_{k_p}\dots\psi^{j_1}_{k_1}(n, a_1, \dots, a_p) \to^* n - p + \sum_{r=1}^{p} j_r$.

We analyse the more general case of naturals between subscripts k. Select $k_{i+1} < n_1 \le k_i$ and $k_{l+1} < n_2 \le k_l$, for $i > l$. Then $\psi^{j_p}_{k_p}\dots\psi^{j_1}_{k_1}(n_1, a_1, \dots, a_p) \to^* n_1 - (p - i) + \sum_{r=i+1}^{p} j_r$ and $\psi^{j_p}_{k_p}\dots\psi^{j_1}_{k_1}(n_2, a_1, \dots, a_p) \to^* n_2 - (p-l) + \sum_{r=l+1}^{p} j_r$.

Since $k_1, \dots, k_p$ is a decreasing sequence we have $n_1 < n_2$. By previous lemma: $\psi^{j_i}_{k_i}\dots\psi^{j_{l+1}}_{k_{l+1}}\dots\psi^{j_1}_{k_1}(n_2, a_1, \dots, a_i) \to^* \psi^{j_i}_{k_i}\dots\psi^{j_{l+1}}_{k_{l+1}}(n_2, a_{l+1}, \dots, a_i) \to^* n_2 - (i-l) + \sum_{r=l+1}^{i} j_r > n_1$. Then $n_2 - (p-l) + \sum_{r=l+1}^{p} j_r > n_1 - (p-i) + \sum_{r=i+1}^{p} j_r$. ■

When searching for solutions of $\psi^{j_p}_{k_p}\dots\psi^{j_1}_{k_1}(X, a_1, \dots, a_p) =^?_{\lambda s_e} (\mathtt{m}\ b_1 \dots b_q)$, one should select a binding for X to an application whose first component is a natural number n such that for some i $k_{i+1} < n \le k_i$ and $n - (p-i) + \sum_{r=i+1}^{p} j_r = m$. This corresponds to searching for solutions of an integer linear problem.

By analyzing the intrinsic implementation techniques involved in our method and that of the $\lambda\sigma$-HOU of [10], we have observed that pre-cooked λ-terms in the λs_e-calculus have linear decorations on the size of the λ-terms and the magnitude of their de Bruijn indices, while in $\lambda\sigma$ these decorations are quadratic. We don't make any consideration about use of efficient data structures. For a reasonable implementation of the $\lambda\sigma$-HOU approach, a variation of the $\lambda\sigma$-calculus which includes all de Bruijn indices, as the $\lambda\sigma_{dB}$ one, should be used. Additionally, variations of this calculus, as the left-linear $\lambda_{\mathcal{L}}$-calculus [33], are adequate for implementations since they encode the infinite rules of the $\lambda\sigma_{dB}$ in a first-class substitution. From the theoretical point of view, our approach is the first one to treat this problem in a natural way, because of the simple syntax of the λs_e-calculus, where all de Bruijn indices are included. But it is not the sole use of all de Bruijn indices that makes the λs_e approach more efficient. Another problem in the decoration of substitution objects of the $\lambda\sigma$-calculus is that they are decorated with two environments that are lists of types. While the main marks in the decoration of a term object are a sole environment and its type. This makes decorations of λs_e-terms smaller than those of $\lambda\sigma$-terms. Moreover, the size of decorated λ-terms enlarges in an inadequate way when normalizing via the $\lambda\sigma$-calculus, because there exist rules in the $\lambda\sigma$-calculus, that are expensive as they enlarge the number of substitution objects to be marked in decorated terms.

7 Future Work and Conclusion

As pointed out in [16], the use of explicit substitution enables one to translate higher order unification problems into first order ones. This leads to simpler development and analysis of HOU methods. The proposed unification method and its further developments are relevant because of the necessity of analyzing, developing and implementing HOU procedures to improve the performance and expressiveness of current higher order deductive systems. Moreover, we think that our work is important due to the necessity of comparing the advantages and appropriateness of both the λs_e- and $\lambda\sigma$-style of explicit substitution in a practical and relevant setting incrementing in this way the theoretic knowledge about the properties of the involved calculi.

Advantages of the here proposed unification method, with respect to the one formulated by Dowek, Hardin and Kirchner in [16], are mainly consequences of the inherent differences between the $\lambda\sigma$- and λs_e-styles of explicit substitution.

1. In λs_e-unification we remain close to λ-calculus as we don't use more than one kind of objects: term objects. We don't use substitution objects as is done in $\lambda\sigma$-unification. From this point of view, we think that our approach is semantically clear because the principal intention of any unification via explicit substitution in some version of λ-calculus is, of course, to solve unification problems in pure λ-calculus.

2. Because of the fact that for both methods, the *Normalize* unification rule depends on the subjacent properties of the λs_e and $\lambda\sigma$ rewrite rules, correspondingly, and that the underlying reduction processes based on the λs_e- and $\lambda\sigma$-calculi are incomparable (see for instance [28]), one cannot say that λs_e-unification is more (or less) efficient than the unification setting proposed in [16]. But at least one can state that searching for redices of the unification rules (and determining solved and *flex-flex* equations) is more efficient, since λs_e terms are written using natural indices. Of course, in the praxis, this problem can be easily solved in the $\lambda\sigma$ setting by overloading the notation $\mathtt{n}$ to represent the corresponding $\lambda\sigma$-term $(\mathtt{1}[\uparrow^{n-1}])$ incorporating to the unification mechanism the necessary built-in linear arithmetic deductive method.

3. We think that the arithmetic constraint that naturally results when defining the *Exp-App* unification rule in the λs_e setting is more expressive than the one of the $\lambda\sigma$. This, jointly with an efficient arithmetic deductive method, speed up the verification of possible splittings and the search for solutions in the corresponding case analysis.

In order to obtain a HOU procedure useful in practice, an efficient and complete unification strategy was developed in [2]. In [16] the rules for unification of $\lambda\sigma$-terms are related to HOU on the pure λ-calculus by the *pre-cooking* and *back* translations. This was also done for the λs_e-calculus in [2].

In the sequel we present in an informal way one example of how to apply our unification method to HOU problems in the λ-calculus.

Observe that unifying two terms a and b in the λ-calculus consists in finding a *substitution* θ such that $\theta(a) =_{\beta\eta} \theta(b)$. But in λ-calculus, $\lambda\sigma$ and λs_e, substitution is different from the first order one or grafting, as was shown in Section 2. Thus using the notation of substitution in Definitions 2.11 and 2.14 a unifier in λ-calculus of the problem $\lambda.X =^?_{\beta\eta} \lambda.2$ is not a term $t = \theta X$ such that $\lambda.t =^?_{\beta\eta} \lambda.2$ but a term $t = \theta X$ such that $\theta(\lambda.X) = \lambda.\theta^+(X) = \lambda.2$ as $\{X/t\}\lambda.X = \lambda.\{X/t^+\}X = \lambda.t^+$ and not $\lambda.t$. This observation can be extended to any unifier and by translating appropriately λ-terms $a, b \in \Lambda_{dB}(\mathcal{X})$, the HOU problem $a =^?_{\beta\eta} b$ can be reduced to equational unification. In [16] a translation called *pre-cooking* from $\Lambda_{dB}(\mathcal{X})$ terms into the language of $\lambda\sigma$ is given such that searching for solutions of the corresponding $\lambda\sigma$-unification problem corresponds to searching for solutions of the higher order problem $a =^?_{\beta\eta} b$. In the next example, we illustrate informally the analogous situation in λs_e.

Example 7.1 Consider the higher order unification problem $\lambda.(X\ \ 2) =^?_{\beta\eta} \lambda.2$, where 2 and X are of type A and $A \to A$, respectively. Observe that applying a substitution solution θ to the $\Lambda_{dB}(\mathcal{X})$-term $\lambda.(X\ \ 2)$ gives $\theta(\lambda.(X\ \ 2)) = \lambda.(\theta^+(X)\ \ 2)$. Then in the λs_e-calculus we are searching for a grafting θ' such that $\theta'(\lambda.(\varphi^2_0(X)\ \ 2)) =_{\lambda s_e} \lambda.2$. Correspondingly, in the $\lambda\sigma$-calculus the term $\lambda.(X\ \ 2)$ is translated or pre-cooked into $\lambda.(X[\uparrow]\ \ 2)$. Then we should search for unifiers for the problem $\lambda.(\varphi^2_0(X)\ \ 2) =^?_{\lambda s_e} \lambda.2$.

Now we apply the λs_e-unification rules to $\lambda.(\varphi^2_0(X)\ \ 2) =^?_{\lambda s_e} \lambda.2$.

$(\varphi^2_0(X)\ \ 2) =^?_{\lambda s_e} 2$ $\quad\to_{Dec-\lambda}$

$\exists Y(\varphi^2_0(X)\ \ 2) =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y$ $\quad\to_{Exp-\lambda}$

$\exists Y(\varphi^2_0(\lambda.Y)\ \ 2) =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y$ $\quad\to_{Replace}$

$\exists Y(\varphi^2_1 Y)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y$ $\quad\to_{Norm.}$

$(\exists Y(\varphi^2_1 Y)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.Y) \quad\wedge\quad (Y =^?_{\lambda s_e} 1 \vee Y =^?_{\lambda s_e} 2)$ $\quad\to_{Exp-app}$

$((\varphi^2_1 1)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.1) \quad\vee\quad ((\varphi^2_1 2)\sigma^1 2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.2)$ $\to_{Replace}$

$(2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.1) \vee (2 =^?_{\lambda s_e} 2 \wedge X =^?_{\lambda s_e} \lambda.2)$ $\quad\to_{Norm.}$

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

To complete the analysis observe that by definition of substitution (Definitions 2.11, 2.14) and β-reduction in $\Lambda_{dB}(\mathcal{X})$: $\{X/\lambda.1\}(\lambda.(X\ \ 2)) = \lambda.(\{X/(\lambda.1)^+\}(X)\ \ 2) = \lambda.(\lambda.1^{+1}\ \ 2) = \lambda.(\lambda.1\ \ 2) =_\beta \lambda.2$ and $\{X/\lambda.2\}(\lambda.(X\ \ 2)) = \lambda.(\{X/(\lambda.2)^+\}(X)\ \ 2) = \lambda.(\lambda.2^{+1}\ \ 2) = \lambda.(\lambda.3\ \ 2) =_\beta \lambda.\{1/2\}(3) = \lambda.2$.

In general, before unification, a λ-term a should be translated into the λs_e-term a' resulting by simultaneously replacing each occurrence of a meta-variable X at position i in a with $\varphi^{k+1}_0 X$, where k is the number of abstractors between the root position of a, ε, and position i. If $k = 0$ then the occurrence of X remains unchanged.

Example 7.2 We turn back to the HOU problem given in the introduction: $F(f(a)) =^? f(F(a))$. In the language of $\Lambda_{dB}(\mathcal{X})$ this problem can be seen as $(X\ \ (2\ \ 1)) =^?_{\beta\eta} (2\ \ (X\ \ 1))$, where both X and 2 are of type $A \to A$ and 1 is of type A. Observe that since there are no λs in the problem, the equation remains unchanged: $(X\ \ (2\ \ 1)) =^?_{\lambda s_e} (2\ \ (X\ \ 1))$.

For simplicity we omit existential quantifiers. Here are the λs_e-unification steps on this problem (Y is of type A):

$(X\ \ (2\ \ 1)) =^?_{\lambda s_e} (2\ \ (X\ \ 1)) \wedge X =^?_{\lambda s_e} \lambda.Y$ $\quad\to_{Exp-\lambda}$

$(\lambda.Y\ \ (2\ \ 1)) =^?_{\lambda s_e} (2\ \ (\lambda.Y\ \ 1)) \wedge X =^?_{\lambda s_e} \lambda.Y$ $\quad\to_{Replace}$

$Y\sigma^1(2\ \ 1) =^?_{\lambda s_e} (2\ \ Y\sigma^1 1) \wedge X =^?_{\lambda s_e} \lambda.Y$ $\quad\to_{Norm.}$

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

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

Other solutions can be obtained from the equational system $(2\ \ H_1\sigma^1(2\ \ 1)) =^?_{\lambda s_e} (2\ \ (2\ \ H_1\sigma^1 1)) \wedge X =^?_{\lambda s_e} \lambda.(3\ \ H_1)$. In fact, by *Dec-App* and *Exp-App* we obtain:

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

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

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

In [10] it was shown that for an efficient implementation of $\lambda\sigma$-HOU, the use of terms decorated with their corresponding types and environments is useful. For instance, observe that for applying unification rules such as *Exp-App* and *Exp-*λ, it is necessary to know the types and the environments of subterms of the current unification problem. In relation with that implementation, where repeated execution of a type-checking algorithm is avoided by decorating terms, λs_e-HOU has the clear advantage of having less expensive decorations than those of $\lambda\sigma$-HOU. This is the case because decorations of substitution objects are more expensive than those of term objects.

References

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

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

[3] M. Ayala-Rincón and F. Kamareddine. Unification via λs_e-Style of Explicit Substitution. In *Second International Conference on Principles and Practice of Declarative Programming*, pages 163–174, Montreal, Canada, 2000.

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

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

[6] Z.-el-A. Benaissa. *Les calculs de substitutions explicites comme fondement de l'implantation des langages fonctionnels.* PhD thesis, Univ. Henri Poincare, Nancy, 1997.

[7] Z.-el-A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. λv, a Calculus of Explicit Substitutions which Preserves Strong Normalization. *Journal of Functional Programming*, 6(5):699–722, 1996.

[8] Z.-el-A. Benaissa, P. Lescanne, and K. H. Rose. Modeling Sharing and Recursion for Weak Reduction Strategies using Explicit Substitution. In *Programming Languages: Implementations, Logics and Programs PLILP'96*, volume 1140 of *Lecture Notes on Computer Science*, pages 393–407. Springer, 1996.

[9] R. Bloo. *Preservation of Termination for Explicit Substitution.* PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1997.

[10] P. Borovanský. Implementation of Higher-Order Unification Based on Calculus of Explicit Substitutions. In M. Bartošek, J. Staudek, and J. Wiedermann, editors, *Proceedings of the SOFSEM'95: Theory and Practice of Informatics*, volume 1012 of *Lecture Notes on Computer Science*, pages 363–368. Springer Verlag, 1995.

[11] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence Properties of Weak and Strong Calculi of Explicit Substitutions. Technical Report RR 1617, INRIA, Rocquencourt, 1992.

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

[13] 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.

[14] N. G. de Bruijn. A Namefree Lambda Calculus with Facilities for Internal Definition of Expressions and Segments. Technical Report TH-Report 78-WSK-03, Department of Mathematics, Eindhoven University of Technology, 1978.

[15] N. G. de Bruijn. Lambda-Calculus Notation with Namefree Formulas Involving Symbols that Represent Reference Transforming Mappings. *Indag. Mat.*, 40:348–356, 1978.

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

[17] G. Dowek, T. Hardin, C. Kirchner, and F. Pfenning. Unification via Explicit Substitutions: The Case of Higher-Order Patterns. *Rapport de Recherche* 3591, INRIA, December 1998.

[18] W. Farmer. A Unification Algorithm for Second-Order Monadic Terms. *Annals of Pure and Applied Logic*, 39:131–174, 1988.

[19] M. C. F. Ferreira, D. Kesner, and L. Puel. Lambda-Calculi with Explicit Substitutions and Composition which Preserve Beta-Strong Normalisation. In *Algebraic and Logic Programming, ALP'96*, volume 1139 of *LNCS*, pages 284–298. Springer, 1996.

[20] W. Goldfarb. The Undecidability of the Second-Order Unification Problem. *Theoretical Computer Science*, 13(2):225–230, 1981.

[21] B. Guillaume. *Un calcul des substitutions avec etiquettes.* PhD thesis, Université de Savoie, Chambéry, 1999.

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

[23] D. Jensen and T. Pietrzykowski. Mechanizing ω-order type theory through unification. Technical Report CS-73-16, Dept. of Applied Analysis and Computer Science, University of Waterloo, 1973.

[24] F. Kamareddine and R. P. Nederpelt. On stepwise explicit substitution. *International Journal of Foundations of Computer Science*, 4(3):197–240, 1993.

[25] F. Kamareddine and R. P. Nederpelt. A useful λ-notation. *Theoretical Computer Science*, 155:85–109, 1996.

[26] 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.

[27] 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.

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

[29] F. Kamareddine, A. Ríos, and J.B. Wells. Calculi of Generalised β-reduction and explicit substitution: Type Free and Simply Typed Versions. *Journal of Functional and Logic Programming*, 1998(Article 5):1–44, 1998.

[30] C. Kirchner and C. Ringeissen. Higher-order Equational Unification via Explicit Substitutions. In *Proc. Algebraic and Logic Programming*, volume 1298 of *LNCS*, pages 61–75. Springer, 1997.

[31] P. Lescanne, Z.-el-A. Benaissa, and D. Briaud. Calculi of Explicit Substitutions: New Results. Presented at the International School on Type Theory and Term Rewriting, Glasgow University, 1996.

[32] L. Magnusson. *The implementation of ALF - a proof editor based on Martin Löf's Type Theory with explicit substitutions.* PhD thesis, Chalmers, 1995.

[33] C. Muñoz. A left-linear variant of $\lambda\sigma$. In *Proc. International Conference PLILP/ALP/HOA'97*, volume 1298 of *LNCS*, pages 224–234, Southampton (England), September 1997. Springer.

[34] C. Muñoz. *Un calcul de substitutions pour la représentation de preuves partielles en théorie de types*. PhD thesis, Université Paris 7, 1997. English version in *Rapport de recherche* INRIA RR-3309, 1997.

[35] G. Nadathur and D. S. Wilson. A representation of lambda terms suitable for operations on their intentions. *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 341–348, 1990.

[36] C. Okasaki. FUNCTIONAL PEARL Even Higher-Order Functions for Parsing or Why Would Anyone Ever Want to Use a Sixth-Order Function? *Journal of Functional Programming*, 8(2):195–199, March 1999.

[37] L. Paulson. Isabelle: The next 700 Theorem Provers. *Logic and Computer Science*, pages 361–386, 1990.

[38] T. Pietrzykowski. A complete mechanization of second order logic. *Journal of the ACM*, 20(2):333–364, 1971.

[39] C. Prehofer. Progress in Theoretical Computer Science. In R. V. Book, editor, *Solving Higher-Order Equations: From Logic to Programming*. Birkhäuser, 1997.

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

[41] J. A. Robinson. A Machine-oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1):23–41, January 1965.

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

[43] R. Vestergaard and J. B. Wells. Cut rules and explicit substitutions. *In Second Int'l Workshop on Explicit Substitutions: Theory and Applications to Programs and Proofs*, pages 14–27, 1999. Trento, Italy.

Received 30 June 2000, revised 3 May 2001