Second- and Third-Order Matching via Explicit Substitutions

FLÁVIO LEONARDO CAVALCANTI DE MOURA, *Departamento de Ciência da Computação, Universidade de Brasília, Brasília DF, Brasil. E-mail: flavio@cic.unb.br*

FAIROUZ KAMAREDDINE, *School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, Scotland. E-mail: fairouz@macs.hw.ac.uk*

MAURICIO AYALA-RINCÓN, *Departamento de Matemática, Universidade de Brasília, Brasília DF, Brasil. E-mail: ayala@unb.br*

Abstract

The past few years have established the benefits of using explicit substitutions in the treatment of problems such as unification and matching related to higher-order automated deduction and the implementation of programming languages. Matching is a basic operation extensively used in computation and deduction. Second and third-order matching, in particular, provide an adequate environment for expressing pattern recognition and program transformation or simplification.

We present a practical algorithm which solves a class of second-order matching problems in the language of the $\lambda\sigma$-calculus of explicit substitutions. This class includes all the problems originated in the language of the simply typed λ-calculus. In addition, we show how Dowek's third-order matching decision procedure can be adapted to the whole language of the simply typed $\lambda\sigma$-calculus and as a consequence we get the decidability of third-(and consequently second-)order matching for the whole language of the simply typed $\lambda\sigma$-calculus.

Keywords: Higher-Order Unification, Higher-Order Matching, Explicit Substitutions.

1 Introduction

Matching is a basic operation which is extensively used in the implementation of automated deduction and programming environments. The decidability of higher-order matching has been an open question for about thirty years [Hue75]. Progress has been made where second-, third- and fourth-order matching have been shown to be decidable [Hue75, Dow94, Pad00]. However, matching solutions are not necessarily unique and most general unifiers do not necessarily exist. Moreover, in [Loa03], the undecidability of fifth-order β-matching is given, but the proof does not deal with η-conversion. Despite all these developments, for the general higher orders case, it remains unknown whether matching is decidable.

Calculi of explicit substitutions are refinements of the λ-calculus by internalising the substitution operation. This is done by extending the language of the λ-calculus and decomposing the substitution operation into smaller steps. By studying matching using explicit substitutions, we can ensure a more accurate theoretical description of implementations based on the λ-calculus. This is useful when we consider low-level implementations in which matching algorithms are to be implemented at the level of the language itself. The advantages of this approach include being closer to low-level implementations based on the λ-calculus, where substitution has to be explicit.

Two main contributions are given:

- The development of a second-order matching algorithm via the $\lambda\sigma$-style of explicit substitutions [ACCL91]. For doing this we introduce a convenient notation for matching problems in the $\lambda\sigma$-calculus and then we adapt the higher-order unification method introduced in [DHK00]. We characterise a class of second-order matching problems for which we prove the termination of this procedure. This class includes all second-order problems originated in the language of the pure λ-calculus. Interesting examples illustrate how the algorithm works.
- A proof of the decidability of third-order (and consequently second-order) matching for the whole language of the $\lambda\sigma$-calculus. This is done by extending the techniques for the λ-calculus introduced in [Dow94] to the language of the simply typed $\lambda\sigma$-calculus: firstly, matching problems are translated to interpolation problems in the language of the $\lambda\sigma$-calculus. Afterwards, it is proved that whenever an interpolation problem has a solution it is possible to find a solution whose size depends only on a measure of the structure of the initial matching problem. This is reached by introducing the notion of the $\lambda\sigma$-Böhm tree which allows us to define an adequate measure. Finally, this measure which is based on the input problem provides the necessary bound for reducing the search space of possible matching solutions guaranteeing in this way the decidability.

After presenting the necessary background in Section 2, Sections 3 and 4 present the main contributions: the second-order matching algorithm via the $\lambda\sigma$-calculus of explicit substitutions and the decidability of third-order matching for the whole language of this calculus. Then Section 5 concludes and presents future work.

2 Background

We start this section with a brief presentation of the simply typed λ- and $\lambda\sigma$-calculus and some basic definitions used throughout the paper. We use de Bruijn indexes [dB72] instead of variable names. This is because de Bruijn's notation is more adequate for implementations of the λ-calculus since α-conversion is no longer needed.

We define types and simply typed λ-terms in de Bruijn notation as usual:

Types	$A ::=$	$K \mid A \to A$, where K is an atomic type.
Contexts	$\Gamma ::=$	$nil \mid A \cdot \Gamma$
Terms	$a ::=$	$\underline{n} \mid X \mid a\ a \mid \lambda_A.a$, where $n \in \mathbb{N} = \{1, 2, \ldots\}$ and $X \in \mathcal{X}$, the set of meta-variables.

As usual, parenthesis are used to avoid ambiguities and we assume that applications are left associative, i.e., $(a_1\ a_2 \ldots a_n)$ means $((\ldots(a_1\ a_2)\ldots)\ a_n)$ and, abstractions are right associative, i.e., $\lambda_{x_1}\lambda_{x_2}\ldots\lambda_{x_n}.a$ is interpreted as $\lambda_{x_1}.(\lambda_{x_2}.(\ldots(\lambda_{x_n}.a)\ldots))$. Moreover, an application has higher priority then an abstraction. In this way, $\lambda_x.a_1\ a_2$ means $\lambda_x.(a_1\ a_2)$.

The set of λ-terms built with the grammar above is usually denoted by $\Lambda_{dB}(\mathcal{X})$ and its system of simple types is given by the following typing rules:

(var) $$\frac{}{A \cdot \Gamma \vdash \underline{1} : A}$$ (var n) $$\frac{\Gamma \vdash \underline{n} : B}{A \cdot \Gamma \vdash \underline{n+1} : B}$$

(app) $$\frac{\Gamma \vdash a : A \to B \quad \Gamma \vdash b : A}{\Gamma \vdash \ a\, b : B}$$ (lambda) $$\frac{A \cdot \Gamma \vdash a : B}{\Gamma \vdash \lambda_A.a : A \to B}$$

The type judgement $\Gamma \vdash a : A$ can also be written a^{Γ}_{A} where the term a is decorated with the context Γ and the type A.

We assume that for each type there exists an enumerable set of meta-variables and add the following typing rule for meta-variables:

(Metavar) $\Gamma \vdash X : A$, where Γ is any context.

β- and η-contraction are defined as usual and $=_{\beta\eta}$ denotes $\beta\eta$-conversion. In this paper we work with $\beta\eta$-conversion which means that terms are assumed to be in η-long normal form. The η-long normal form of a $\Lambda_{dB}(\mathcal{X})$-term to be defined is an adaptation from [DHK00] which uses the updating functions defined below.

Definition 2.1 (Updating functions U^i_k)
The *updating functions* $U^i_k : \Lambda_{dB}(\mathcal{X}) \to \Lambda_{dB}(\mathcal{X})$, for $k \geq 0$ and $i \geq 1$ are defined inductively by:

(a) $U^i_k(X) = X$, for $X \in \mathcal{X}$

(b) $U^i_k(a\, b) = U^i_k(a)\, U^i_k(b)$

(c) $U^i_k(\lambda_A.a) = \lambda_A.U^i_{k+1}(a)$

(d) $U^i_k(\underline{n}) = \begin{cases} \underline{n+i-1}, & \text{if } n > k \\ \underline{n}, & \text{if } n \leq k \end{cases}$

Definition 2.2 (η-long normal forms)
Let a be a $\Lambda_{dB}(\mathcal{X})$-term in β-normal form (β-nf, for short) of type $A_1 \to \ldots \to A_m \to B$, where $m \geq 0$ and B is atomic, in context Γ. The *η-long normal form* (η-lnf, for short) a' of a, is inductively defined by:

(a) if $a = \lambda_A.b$ then $a' = \lambda_A.b'$.

(b) if $a = \underline{n}\, b_1 \ldots b_q$, with $q \geq 0$, then $a' = \lambda_{A_1} \ldots \lambda_{A_m}.\underline{n+m}\, c_1 \ldots c_q\, \underline{m}' \ldots \underline{1}'$, where $c_1, \ldots, c_q$ are the η-lnfs of the β-nfs of $U^{m+1}_0(b_1), \ldots, U^{m+1}_0(b_q)$, respectively.

(c) if $a = X\, b_1 \ldots b_q$, with $q \geq 0$, then $a' = \lambda_{A_1} \ldots \lambda_{A_m}.X\, c_1 \ldots c_q\, \underline{m}' \ldots \underline{1}'$, where $c_1, \ldots, c_q$ are the η-lnfs of the β-nfs of $U^{m+1}_0(b_1), \ldots, U^{m+1}_0(b_q)$, respectively.

Definition 2.3 (Order of types and terms)
The order of a $\Lambda_{dB}(\mathcal{X})$-term a is the order of its type, say A whose order, written as $|A|$, is inductively defined by:

(a) If A is atomic then $|A| = 1$;

(b) If $A = B \to C$ then $|A| = max\{1 + |B|, |C|\}$.

Unification problems deal with *unification equations* which are defined by:

Definition 2.4 (Unification equation: flexible-flexible, rigid-rigid, flexible-rigid)
A *unification equation* is an equation of the form $a =^? b$ where a and b are well-typed $\Lambda_{dB}(\mathcal{X})$-terms of the same type and under the same context. The *order* of a unification equation is the highest order of the meta-variables occurring in it. A unification equation is called *flexible-flexible* or *rigid-rigid* if the left and right-hand sides of the equation are both flexible or rigid terms, respectively. If one term is rigid and the other is flexible (independently of the order) the equation is called *flexible-rigid*. A unification equation is called *trivial* if it has the form $a =^? a$.

Example 2.5
Let $\Gamma = A \cdot A \to A \cdot A \to B \cdot B \cdot nil$ and X and Y be meta-variables such that $\Gamma \vdash X : A \to A$ and $\Gamma \vdash Y : (A \to A) \to B$. The unification equation $(\underline{3}(\underline{2}(X\ \underline{1}))) =^? \underline{4}$ is well typed in Γ and has order 2 (since X has order 2); the unification equation $(Y\ X) =^? \underline{4}$ is also well typed in Γ and has order 3.

Definition 2.6 (Unifier/Matcher, Unification/Matching problem)

- A *unifier* for the unification equation $a =^? b$ is a substitution σ where $a\sigma =_{\beta\eta} b\sigma$.
- A *unification problem* P is a finite conjunction of unification equations:

$$\bigwedge_i (a_i =^? b_i)$$

  The *order* of a unification problem is given by the highest order amongst its unification equations. A solution to the unification problem P is a substitution which is a unifier for all equations in P.
- A *higher-order matching equation* is an equation of the form $a \ll^? b$, where a and b are λ-terms of the same type which are well typed in the same context and, such that b does not contain meta-variables.
- A solution or a *matcher* for the matching equation $a \ll^? b$, is a substitution σ such that $a\sigma =_{\beta\eta} b$.
- A *higher-order matching problem* is a finite conjunction of matching equations:

$$\bigwedge_i (a_i \ll^? b_i)$$

  The order of a matching problem is given by the highest order of its meta-variables.

The definition of a matcher corresponds to the notion of "filtering", which comes from the assumption that the term to be matched have disjoint variable sets or they can be renamed as usual in rewriting systems and pattern matching. The alternative notion of "semi-unification" $(\exists\sigma, a\sigma =_{\beta\eta} b\sigma =_{\beta\eta} b)$ is not treated here [Bur89].

2.1 The $\lambda\sigma$-calculus

The $\lambda\sigma$-calculus of explicit substitutions extends the λ-calculus with explicit operators to simulate the substitution (meta-)operation of the λ-calculus. The rewriting system of the $\lambda\sigma$-calculus is given in Table 1. Its syntax is given as follows:

Types	A	$::=$	$K \mid A \to A$	
Contexts	Γ	$::=$	$nil \mid A \cdot \Gamma$	
Terms	a	$::=$	$\underline{1} \mid X \mid a\ a \mid \lambda_A.a \mid a[s]$	where $X \in \mathcal{X}$
Substitutions	s	$::=$	$id \mid \uparrow \mid a \cdot s \mid s \circ s$	

The set of $\lambda\sigma$-terms is written as $\Lambda_{\lambda\sigma}(\mathcal{X})$. Substitutions are lists of terms in the $\lambda\sigma$-calculus and hence the type of a substitution must be a list of types, i.e., a context. If s is a substitution and Γ and Δ are contexts then we write $\Gamma \vdash s \triangleright \Delta$ to represent that the substitution s has type Δ in context Γ. The $\lambda\sigma$-typing rules are given by:

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

TABLE 1. The $\lambda\sigma$-Rewriting System

(var) $$\frac{}{A \cdot \Gamma \vdash \underline{1} : A}$$

(lambda) $$\frac{A \cdot \Gamma \vdash a : B}{\Gamma \vdash \lambda_A.a : A \to B}$$

(app) $$\frac{\Gamma \vdash a : A \to B \quad \Gamma \vdash b : A}{\Gamma \vdash \ a\, b : B}$$

(clos) $$\frac{\Gamma \vdash s \triangleright \Gamma' \quad \Gamma' \vdash a : A}{\Gamma \vdash a[s] : A}$$

(id) $$\frac{}{\Gamma \vdash id \triangleright \Gamma}$$

(shift) $$\frac{}{A \cdot \Gamma \vdash \uparrow \triangleright \Gamma}$$

(cons) $$\frac{\Gamma \vdash a : A \quad \Gamma \vdash s \triangleright \Gamma'}{\Gamma \vdash a \cdot s \triangleright A.\Gamma'}$$

(comp) $$\frac{\Gamma \vdash s'' \triangleright \Gamma'' \quad \Gamma'' \vdash s' \triangleright \Gamma'}{\Gamma \vdash s' \circ s'' \triangleright \Gamma'}$$

In addition, to each meta-variable X we associate a unique type T_X and a unique context Γ_X. We assume that for each pair (Γ, A) there is an enumerable set of meta-variables, such that for each meta-variable X in this set, we have that $\Gamma_X = \Gamma$ and $T_X = A$. We add the following typing rule for meta-variables:

(Metavar) $$\frac{}{\Gamma_X \vdash X : T_X}$$

We write a_A^Γ to say that the $\lambda\sigma$-term a has type A in context Γ.

The simply typed $\lambda\sigma$-calculus is weakly terminating and, hence every typed term has a normal form. The $\lambda\sigma$-normal form ($\lambda\sigma$-nf for short) of an expression of sort term or substitution is essential in this work and is given by the following:

PROPOSITION 2.7 ([Rio93])
Any $\lambda\sigma$-term in $\lambda\sigma$-*nf* has one of the following forms:

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

Definition 2.8 (η-lnf [DHK00])
Let a be a $\lambda\sigma$-term of type $A_1 \to \ldots \to A_n \to B$ in context Γ and in $\lambda\sigma$-nf. The *η-lnf* of a, written as a', is defined by:

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

From now on terms in $\lambda\sigma$-nf are assumed to be in η-lnf. The unification rules of the $\lambda\sigma$-calculus, introduced in [DHK00], are given in Table 2.

Definition 2.9 ([DHK00])
A unification system P is in *$\lambda\sigma$-solved form* if it is a conjunction of nontrivial equations of the following forms:

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

3 Second-order Matching

Higher-order unification problems are reduced to first-order unification problems modulo the equational theory of the $\lambda\sigma$-calculus via the so called *precooking translation*:

Definition 3.1 (Precooking [DHK00])
Let $a \in \Lambda_{dB}(\mathcal{X})$ such that $\Gamma \vdash a : A$. To every meta-variable X of type B in the term a, we associate the type B and context Γ in the $\lambda\sigma$-calculus. The *precooking* of a from $\Lambda_{dB}(\mathcal{X})$ to $\Lambda_{\lambda\sigma}(\mathcal{X})$, is defined by $a_F = f(a, 0)$ where $f(a, n)$ is defined by:
(*a*) $f((\lambda_B.a), n) = \lambda_B(f(a, n+1))$ (*b*) $f(\underline{k}, n) = \underline{1}[\uparrow^{k-1}]$
(*c*) $f(a\ b, n) = f(a, n)\ f(b, n)$ (*d*) $f(X, n) = X[\uparrow^n]$

The precooking translation is a function that takes a term from the simply typed λ-calculus and returns an equivalent term in the language of the simply typed $\lambda\sigma$-calculus. This translation is essential to avoid variable capture since the HOU procedure in the $\lambda\sigma$-calculus uses first-order substitution (grafting). A $\lambda\sigma$-term of the form $X[\uparrow^n]$ expresses the fact that X is in the scope of n abstractors and hence, the replacement of X by a term, say a, results in the term $a[\uparrow^n]$ that will increase by n all the free de Bruijn indices that occur in a during the normalisation.

The unification problem:

$$\lambda_A.X =^?_{\lambda\sigma} \lambda_A.\underline{1} \tag{3.1}$$

is a well-formed unification problem in the language of the $\lambda\sigma$-calculus, nevertheless it is neither in the image of the precooking translation nor is derived from a problem that is in the image of the precooking translation. In fact, X is in the scope of

Dec-λ $$\frac{P \wedge \lambda_A.e_1 =^?_{\lambda\sigma} \lambda_A.e_2}{P \wedge e_1 =^?_{\lambda\sigma} e_2}$$

Dec-App $$\frac{P \wedge (\underline{n}\ e^1_1 \ldots e^1_p) =^?_{\lambda\sigma} (\underline{n}\ e^2_1 \ldots e^2_p)}{P \wedge e^1_1 =^?_{\lambda\sigma} e^2_1 \wedge \ldots \wedge e^1_p =^?_{\lambda\sigma} e^2_p}$$

Dec-Fail $$\frac{P \wedge (\underline{n}\ e^1_1 \ldots e^1_{p_1}) =^?_{\lambda\sigma} (\underline{m}\ e^2_1 \ldots e^2_{p_2})}{Fail}, \text{ if } m \neq n.$$

Exp-λ $$\frac{P}{\exists(A \cdot \Gamma \vdash Y : B), P \wedge X =^?_{\lambda\sigma} \lambda_A.Y}$$
if $(\Gamma \vdash X : A \to B) \in \mathcal{TV}ar(P)$, $Y \notin \mathcal{TV}ar(P)$,
and X is not a solved variable.

Exp-App $$\frac{P \wedge X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)}{P \wedge X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] =^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q) \wedge \bigvee_{r \in R_p \cup R_i} \exists H_1 \ldots \exists H_k : X =^?_{\lambda\sigma} (\underline{r}\ H_1 \ldots H_k)}$$
if X has an atomic type and is not solved;
where $H_1, \ldots, H_k$ are fresh variables of appropriate types,
not occurring in P, with the contexts $\Gamma_{H_i} = \Gamma_X$,
R_p is the subset of $\{1, \ldots, p\}$ such that $(\underline{r}\ H_1 \ldots H_k)$ has the right type, R_i =if $m \geq n + 1$ then $\{m - n + p\}$ else $\emptyset$.

Normalise $$\frac{P \wedge e_1 =^?_{\lambda\sigma} e_2}{P \wedge e'_1 =^?_{\lambda\sigma} e'_2} \text{ if } e_1 \text{ or } e_2 \text{ is not in } \eta\text{-lnf.}$$
where e'_1 (resp. e'_2) is the η-lnf of e_1 (resp. e_2)
if e_1 (resp. e_2) is not a solved variable and e_1 (resp. e_2) otherwise.

Replace $$\frac{P \wedge X =^?_{\lambda\sigma} t}{\{X \mapsto t\}(P) \wedge X =^?_{\lambda\sigma} t} \text{ if } X \in \mathcal{TV}ar(P), X \notin \mathcal{TV}ar(t) \text{ and}$$
if t is a meta-variable then $t \in \mathcal{TV}ar(P)$.

TABLE 2. Unification Rules for the $\lambda\sigma$-calculus (see [DHK00])

the abstractor λ_A which prohibits the replacement of X by a term containing a free occurrence of the index $\underline{1}$. Applying the rule **Dec-λ** to (3.1) we get a solved form:

$$\frac{\lambda_A.X =^?_{\lambda\sigma} \lambda_A.\underline{1}}{X =^?_{\lambda\sigma} \underline{1}} \text{Dec-}\lambda$$

The solved form $X =^?_{\lambda\sigma} \underline{1}$ is associated with the grafting $X \mapsto \underline{1}$ which clearly is not a solution to the problem (3.1). In fact, the problem (3.1) does not have solutions. Therefore, the unification procedure of [DHK00] was designed for unification problems that were originated from the simply typed λ-calculus.

3.1 An Important Class of $\lambda\sigma$-terms

In this subsection we characterise the structure of second-order terms in the language of the $\lambda\sigma$-calculus which can be derived from terms that are originally in the image of the precooking translation. Such characterisation is essential to guarantee that the problems we are trying to solve come from the simply typed λ-calculus.

The motivation for this section comes from the fact that the unification method [DHK00] does not terminate for all second-order matching problems written in the $\lambda\sigma$-style. Hence this method does not allow us to answer directly whether second-order matching in the $\lambda\sigma$-calculus is decidable. The counter-example is the following:

$$X_A^{A\to A\cdot\Gamma}[(\lambda_A.\underline{1}_A^{A\cdot\Gamma})^{\Gamma}_{A\to A}\cdot id^{\Gamma}_{\Gamma}]^{\Gamma}_A =^?_{\lambda\sigma} b^{\Gamma}_A$$

where b is a given closed term, i.e. a term without occurrences of meta-variables.

We can build the following derivation (for simplicity, decorations are removed):

$X[(\lambda_A.\underline{1})\cdot id] =^?_{\lambda\sigma} b \to^{\mathbf{Exp-App}} \quad X[(\lambda_A.\underline{1})\cdot id] =^?_{\lambda\sigma} b \wedge X =^?_{\lambda\sigma} (\underline{1}\,Y) \to^{\mathbf{Replace}}$

$(\underline{1}\,Y)[(\lambda_A.\underline{1})\cdot id] =^?_{\lambda\sigma} b \wedge X =^?_{\lambda\sigma} (\underline{1}\,Y) \to^{\mathbf{Normalise}}$

$Y[(\lambda_A.\underline{1})\cdot id] =^?_{\lambda\sigma} b \wedge X =^?_{\lambda\sigma} (\underline{1}\,Y) \to^{\mathbf{Exp-App}} \dots$

At this point we can repeat the strategy **Exp-App**, **Replace** and **Normalise** since the last problem generated (see the last line) is composed by two flexible-rigid equations, the first of which is equivalent to the original problem up to the renaming of meta-variables.

Now we present a technical lemma that guarantees the existence of a special strategy for reducing the composition of substitutions.

Lemma 3.2
Let a be a well typed term and $b_1\cdot\ldots\cdot b_r\cdot\uparrow^{r'}$ and $c_1\cdot\ldots\cdot c_s\cdot\uparrow^{s'}$ be two well typed substitutions in the language of the $\lambda\sigma$-calculus, where:

- $r, r', s, s' \geq 0$;
- $b_1, \ldots, b_r$ are atomic terms;
- $\begin{cases} c_{r'+1},\ldots,c_s \text{ are atomic (and the other } c_i\text{'s have arbitrary types), if } r' < s; \\ c_1,\ldots,c_s \text{ have arbitrary types, otherwise.}\end{cases}$

If the term $(a[b_1\cdot\ldots\cdot b_r\cdot\uparrow^{r'}])[c_1\cdot\ldots\cdot c_s\cdot\uparrow^{s'}]$ is well typed then there exists a normalisation strategy that leads to the term $a[a_1\cdot\ldots\cdot a_p\cdot\uparrow^n]$, where $a_1\cdot\ldots\cdot a_p\cdot\uparrow^n$ is a substitution in $\lambda\sigma$-normal form in which all the terms $a_1,\ldots,a_p$ have atomic types and $\begin{cases} p = r \text{ and } n = r'-s+s', & \text{if } r' \geq s; \\ p = r+s-r' \text{ and } n = s', & \text{if } r' < s.\end{cases}$

Proof. Consider the following normalisation strategy:
$(a[b_1\cdot\ldots\cdot b_r\cdot\uparrow^{r'}])[c_1\cdot\ldots\cdot c_s\cdot\uparrow^{s'}] \to_{Clos} a[(b_1\cdot\ldots\cdot b_r\cdot\uparrow^{r'})\circ(c_1\cdot\ldots\cdot c_s\cdot\uparrow^{s'})] \to^r_{Map}$
$a[b_1[c_1\cdot\ldots\cdot c_s\cdot\uparrow^{s'}]\cdot\ldots\cdot b_r[c_1\cdot\ldots\cdot c_s\cdot\uparrow^{s'}]\cdot(\uparrow^{r'}\circ(c_1\cdot\ldots\cdot c_s\cdot\uparrow^{s'}))]$.

- If $r' \geq s$ then after s applications of the rules ($Assoc$) and ($ShiftCons$), we get:

$$a[b_1[c_1\cdot\ldots\cdot c_s\cdot\uparrow^{s'}]\cdot\ldots\cdot b_r[c_1\cdot\ldots\cdot c_s\cdot\uparrow^{s'}]\cdot\uparrow^{r'-s+s'}].$$

Since $b_1, \ldots, b_r$ have atomic types, we conclude that the $\lambda\sigma$-normal form of the terms:

$$b_1[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}], \ldots, b_r[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}]$$

are also atomic terms, and that $p = r$ and $n = r' - s + s'$.

- If $r' < s$ then after r' applications of the rules $(Assoc)$ AND $(ShifCons)$, we get:

$$a[b_1[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}] \cdot \ldots \cdot b_r[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}] \cdot c_{r'+1} \cdot \ldots c_s \cdot \uparrow^{s'}].$$

Since the terms $b_1, \ldots, b_r, c_{r'+1}, \ldots, c_s$ have atomic type by hypothesis, we conclude that the $\lambda\sigma$-normal form of the terms $b_1[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}], \ldots, b_r[c_1 \cdot \ldots \cdot c_s \cdot \uparrow^{s'}], c_{r'+1}, \ldots, c_s$ also have atomic type. In this case, $p = r + s - r'$ and $n = s'$. ∎

The next definition allows us to distinguish occurrences of the same meta-variable in a given unification system, which will be essential in Proposition 3.4.

Definition 3.3 (Solved occurrence)
An occurrence of the meta-variable X is said to be *solved* if it occurs in an equation of the form $X =^?_{\lambda\sigma} a$ where a is a $\lambda\sigma$-term without occurrences of X.

Definition 3.3 allows the same meta-variable to have simultaneously solved and non-solved occurrences in the same unification system. E.g., in the unification problem

$$\lambda_A.X[\uparrow] =^?_{\lambda\sigma} \lambda_A.\underline{2} \ \wedge \ X =^?_{\lambda\sigma} \underline{1} \tag{3.2}$$

the occurrence of X in the first equation is non-solved while in the second equation its occurrence in solved. According to Definition 2.9 the meta-variable X in (3.2) is non-solved as well as the two equations of the problem.

For a given unification system in the simply typed $\lambda\sigma$-calculus, we call *fresh equations* all the new equations introduced by applications of the rules **Exp-λ** or **Exp-App**. The notion of fresh equation is important because it allows us to keep a distinction between the equations that come from the original problem and the auxiliary ones introduced during the unification process.

Proposition 3.4 (Characterisation of a special subclass of $\lambda\sigma$-terms)
Let P be a second-order unification system in the simply typed $\lambda\sigma$-calculus and X be a meta-variable occurring in P. If P can be obtained from a unification system that is in the image of the precooking translation using the rules given in Table 2, then each non-solved occurrence of the meta-variable X in non-fresh equations in P has the form $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ where $p, n \geq 0$ and the terms $a_1, \ldots, a_p$ are atomic.

Proof. By induction on the size of the derivation that generated P.

If P is in the image of the precooking translation then each occurrence of a meta-variable X in P has the form $X[\uparrow^n]$ $(n \geq 0)$ and the result follows where $p = 0$.

For the induction step suppose that P is obtained from a unification system that is in the image of the precooking translation using the rules of Table 2. Let P' be a system derived in one step from P after an application of the rule:

- **Dec-λ** or **Dec-App**: In this case any non-solved occurrence of the form $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ in P' was already an occurrence in P since this rule do not change the structure of flexible terms and the proposition follows by IH.

- **Exp-λ**: In this case the unification system P' is obtained from P by adding a fresh equation of the form $X =^?_{\lambda\sigma} \lambda_A.Y$ and the result follows by IH.
- **Exp-App**: In this case a new unification problem is generated and each new unification system results from P by adding a fresh equation of the form $X =^?_{\lambda\sigma} (\underline{r}\ H_1 \ldots H_k)$. The result follows for each generated unification system by IH.
- **Replace**: In this case, P contains an equation of the form $X =^?_{\lambda\sigma} a$ and the result follows by IH because the replacement of X by a preserves the type of the terms. In particular the types of the terms inside the substitutions remain unchanged.
- **Normalise**: From Table 1 we infer that terms are introduced inside substitutions only by applications of the rules (*Beta*) or (*Abs*). In both cases, the type of the term introduced in the substitution is equal to the type associated[1] to the abstractor of the applied rule. We can separate the abstractors into two distinct non-overlapping sets: the abstractors that were already in the original problem, and the ones introduced by applications of the rule **Exp-λ**. The abstractors in the first set may have arbitrary types while those generated by applications of **Exp-λ** must have atomic types because meta-variables are at most second-order. From the definition of the precooking translation we have that if X is a meta-variable that is in the scope of n abstractors then it is translated to the language of the $\lambda\sigma$-calculus as $X[\uparrow^n]$. Suppose some of these n abstractors form a β-redex:

$$(\lambda_{A_1}.\cdots X[\uparrow^n]\cdots)\ b \tag{3.3}$$

  An application of the rule (*Beta*) generates a new sub-term of the form $(\cdots X[\uparrow^n] \cdots)[b\cdot id]$ and the propagation of the substitution $b\cdot id$ results in a sub-term of the form $X[\uparrow^n][\underline{1}\cdot\ldots\cdot\underline{n'}\cdot b[\uparrow^{n'}]\cdot\uparrow^{n'}]$, where $n' \leq n$ is the number of abstractors between λ_{A_1} and $X[\uparrow^n]$. In fact, this propagation is obtained by successive applications of (*Abs*) and (*App*). Considering (the semantics of) the rule (*ShiftCons*) we conclude that all the terms introduced in substitutions whose associated abstractors were already in the original problem (and hence may have arbitrary types) are removed by applications of the rule (*ShiftCons*). Therefore every sub-term of the form $X[a_1\cdot\ldots\cdot a_P\cdot\uparrow^n]$ is such that the terms $a_1,\ldots a_p$ come from λ's originated by applications of the rule **Exp-λ**, and since meta-variables are at most second-order, we conclude that $a_1,\ldots,a_p$ are atomic. ∎

The given characterisation allows us to conclude that all the possible "projections" over the terms $a_1,\ldots,a_p$ do not introduce new meta-variables because all of these terms have atomic types. In this way we can optimise the rule **Exp-App** for this particular case. In the next subsection we define an adequate notation for dealing with *higher-order matching*. Afterwards, we present the second-order matching algorithm.

3.2 Unification by Transformation Notation

Matching problems are characterised by the fact that terms in the right-hand side of matching equations cannot be instantiated. Therefore, the first difficulty to use the general rules of [DHK00] is related to applications of the rule **Exp-λ** because it introduces a flexible-flexible equation whose right-hand side needs to be instantiated.

[1] For example, the type A is associated to the abstractor λ_A.

As an example, let $a \ll^?_{\lambda\sigma} b$ be a second-order matching problem such that the term a has an occurrence of the meta-variable X of functional type, say $A \to A$. An application of a rule like **Exp-**λ introduces a new equation of the form $X =^?_{\lambda\sigma} \lambda_A.Y$, where the fresh meta-variable Y needs to be instantiated, i.e., this new equation is not a matching equation and hence, the resulting problem is no longer a matching problem. To solve this drawback we use a notation based on the so called "unification by transformation" approach [Nip93]. According to this approach, a matching problem will be represented by a pair of the form $\langle \sigma, M \rangle$, where σ is a grafting, and M is a matching problem. The advantage of this notation is that we can define matching rules that do not introduce terms that need to be instantiated in the right-hand side of the equations because graftings and matching equations are kept in different places. Using this notation the rule **Exp-**λ can be adapted so that a problem obtained after an application of it is still a matching problem. In the case of the above example, the **Exp-**λ rule can be designed in such a way that from the matching problem $\langle \{\}, a \ll^?_{\lambda\sigma} b \rangle$ we get the equivalent matching problem $\langle \{X \mapsto \lambda_A.Y\}, \{a \ll^?_{\lambda\sigma} b\} \rangle$.

This notation is independent from the matching rules and, hence we can characterise solved forms without knowing explicitly the matching rules.

Definition 3.5 (Solved form)
A *solved form* is a pair of the form $\langle \theta, \{\} \rangle$, where the first element of the pair is a grafting and the second element is the trivial conjunction.

In the next subsection we give a second-order matching algorithm able to solve second-order matching problems that come from the simply typed λ-calculus, i.e., the problems that matters in practice. This algorithm is an adaptation of the unification algorithm given in [DHK00].

3.3 The Second-Order Matching Algorithm

The second-order matching rules are given in Table 3. The rules **Dec**$_m$**-**λ, **Dec**$_m$**-App**, **Dec**$_m$**-Fail** and **Normalise**$_m$ correspond respectively to **Dec-**λ, **Dec-App**, **Dec-Fail** and **Normalise** written in the unification by transformation notation. The rule **Exp**$_m$**-**λ is the matching version of **Exp-**λ. The difference between them, in addition to the notation, is that **Exp**$_m$**-**λ always replaces a meta-variable of functional type by an abstraction whose body is a fresh meta-variable of atomic type and also applies the generated grafting to the current matching problem. This sole step corresponds to several applications of **Exp-**λ and **Replace**. Note that, if no replacement is done, the rule **Exp-**λ could be applied *ad infinitum.* To avoid such infinite reductions, [DHK00] defined the notion of *fair strategy.* The definition of **Exp**$_m$**-**λ avoids the necessity of defining any strategy because the rules in Table 3 cannot be applied to a given second-order matching problem forever. In fact, for a given equation each rule can be applied only once. The rules **Imit** and **Proj** generate grafting for flexible-rigid equations when the head of the flexible term is a meta-variable of atomic type. The main difference between **Imit** and **Proj** is that the latter does not introduce fresh meta-variables. In addition, while **Proj** may generate several different graftings, for **Imit** we have at most one grafting. Moreover, in the rule **Imit**, the head of the term which replaces X is a de Bruijn index of at most third order. This is because the newly introduced meta-variables have at most second-order.

Dec$_m$-λ $$\frac{\langle \sigma, M \wedge \lambda_A.a \ll^?_{\lambda\sigma} \lambda_A.b \rangle}{\langle \sigma, M \wedge a \ll^?_{\lambda\sigma} b \rangle}$$

Dec$_m$-App $$\frac{\langle \sigma, M \wedge (\underline{n}\, a_1 \ldots a_p) \ll^?_{\lambda\sigma} (\underline{n}\, b_1 \ldots b_p) \rangle}{\langle \sigma, M \wedge a_1 \ll^?_{\lambda\sigma} b_1 \wedge \ldots \wedge a_p \ll^?_{\lambda\sigma} b_p \rangle}$$

Dec$_m$-Fail $$\frac{\langle \sigma, M \wedge (\underline{n}\, a_1 \ldots a_p) \ll^?_{\lambda\sigma} (\underline{m}\, b_1 \ldots b_q) \rangle}{Fail}, \text{ if } n \neq m.$$

Exp$_m$-λ $$\frac{\langle \sigma, M \rangle}{\exists Y : (A_k \cdot \ldots \cdot A_1 \cdot \Gamma \vdash Y : B), \langle \sigma', \{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\}M \rangle}$$
if $(\Gamma \vdash X : A_1 \to \ldots \to A_k \to B) \in \mathcal{TV}ar(M)$, $Y \notin \mathcal{TV}ar(M)$,
and X is not a solved variable.
where $\sigma' = \{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\}\sigma$

Imit $$\frac{\langle \sigma, M \wedge X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \ll^?_{\lambda\sigma} (\underline{m}\, b_1 \ldots b_q) \rangle}{\langle \sigma', \sigma' M \wedge (\underline{m-n+p}\, H_1 \ldots H_q)[\sigma' a_1 \cdot \ldots \cdot \sigma' a_p \cdot \uparrow^n] \ll^?_{\lambda\sigma} b' \rangle}$$
if X has atomic type and $m > n$.
where $b' = (\underline{m}\, b_1 \ldots b_q)$, $\sigma' = \{X \mapsto (\underline{m-n+p}\, H_1 \ldots H_q)\}\sigma$,
$H_1, \ldots, H_q$ are fresh meta-variables with appropriate type and with contexts $\Gamma_{H_i} = \Gamma_X (1 \leq i \leq q)$.

Proj $$\frac{\langle \sigma, M \wedge X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \ll^?_{\lambda\sigma} (\underline{m}\, b_1 \ldots b_q) \rangle}{\langle \{X \mapsto \underline{j}\}\sigma, \{X \mapsto \underline{j}\}M \wedge \{X \mapsto \underline{j}\}a_j \ll^?_{\lambda\sigma} (\underline{m}\, b_1 \ldots b_q) \rangle}$$
if X has atomic type, and the j-th element $(1 \leq j \leq p)$
in the list $a_1 \cdot \ldots \cdot a_p$ has the same type of X.

Normalise$_m$ $$\frac{\langle \sigma, M \wedge a \ll^?_{\lambda\sigma} b \rangle}{\langle \sigma', M \wedge a' \ll^?_{\lambda\sigma} b' \rangle} \text{ if } a \text{ or } b \text{ is not in } \eta\text{-lnf.}$$
where a' (resp. b') is the η-lnf of a (resp. b),
and σ' is obtained from σ by normalising all its terms.
if a (resp. b) is not a solved variable and a (resp. b) otherwise.

Table 3. Second-Order Matching Rules

To prove that the rules of Table 3 always terminate for second-order matching problems whose terms belong to the class characterised by Proposition 3.4, we need to define an adequate measure. We start by giving the length of a $\lambda\sigma$-term:

Definition 3.6 (Length of a $\lambda\sigma$-term)
Let $a \in \Lambda_{\lambda\sigma}(\mathcal{X})$. We inductively define $|a|$, the length of a, by:

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

Definition 3.7
Let M be a matching problem of the form: $a_1 \ll^?_{\lambda\sigma} b_1 \wedge \ldots \wedge a_n \ll^?_{\lambda\sigma} b_n$. We define $\mu(M) = (\xi, \xi', \xi'')$ where

- $\xi = \Sigma^n_{i=1}|b_i|$
- ξ' = the number of meta-variables occurring in M
- ξ'' = the sum of the order of the type of all meta-variables occurring in M.

Now denote by $<$ the usual lexicographic order over triples.

Proposition 3.8
Applications of the rules of Table 3 to second-order matching problems whose terms belong to the class characterised by Proposition 3.4 always terminate.

Proof. We prove that if M' is a second-order matching problem derived from M, then $\mu(M') < \mu(M)$. The proof is by cases, according to the rules that generated M':

- **Dec**$_m$**-**λ: An application of this rule decreases the length of the terms on both sides of the equation in which it is applied and, hence $\mu(M') < \mu(M)$.
- **Dec**$_m$**-App**: An application of this rule decreases the length of the terms on both sides of the equation in which it is applied and, hence $\mu(M') < \mu(M)$.
- **Exp**$_m$**-**λ: An application of this rule replaces a meta-variable of functional type by a fresh one with atomic type. In this way, clearly ξ remains unchanged, and ξ' also remains unchanged because the new introduced fresh meta-variable replaces all the occurrences of the one it is substituted for. Finally, ξ'' decreases because the new introduced meta-variable has a type with smaller order.
- **Imit**: An application of this rule decreases ξ.
- **Proj**: If M' is any of the (many finite) new matching problems generated after an application of **Proj** to M then we have that ξ remains unchanged and ξ' decreases because no new meta-variable is introduced.
- **Normalise**$_m$: Since the $\lambda\sigma$-calculus is weakly-normalizing, we have that an equation of the form $a \ll^?_{\lambda\sigma} b$ is transformed into an equation of the form $a' \ll^?_{\lambda\sigma} b$ by an application of this rule, where a' is the η-lnf of a. In this way, we have that ξ remains unchanged, but ξ' and ξ'' cannot increase because none of the rules of the $\lambda\sigma$-calculus introduce new meta-variables. From this, we conclude that $\mu(M') \leq \mu(M)$, but since normalize can only be applied once at a time (intercalated with other rules) we have that, if the resulting problem is in solved form then the reduction stops. Otherwise, at least one of the other rules can be applied and in this case the measure decreases. ∎

Example 3.9
Let M be the second-order matching problem given by the equation $\Gamma \vdash \lambda_A.(X\ \underline{3}) \ll^? \lambda_A.(\underline{2}(\underline{4}\ \underline{3})) : A \to B$, whose context is given by $\Gamma = A \to B \cdot A \cdot A \to A \cdot nil$, where A and B are atomic types and $\Gamma \vdash X : A \to B$. After the precooking translation, we get the equation $\lambda_A.(X[\uparrow]\ \underline{3}) \ll^?_{\lambda\sigma} \lambda_A.(\underline{2}(\underline{4}\ \underline{3}))$. The algorithm generates the reduction:

$$\langle\{\},\{\lambda_A.(X[\uparrow]\,\underline{3}) \ll^?_{\lambda\sigma} \lambda_A.(\underline{2}(\underline{4}\,\underline{3}))\}\rangle$$

$$\Big\downarrow\ \mathbf{Dec}_m\textbf{-}\lambda$$

$$\langle\{\},\{(X[\uparrow]\,\underline{3}) \ll^?_{\lambda\sigma} (\underline{2}(\underline{4}\,\underline{3}))\}\rangle$$

$$\Big\downarrow\ \mathbf{Exp}_m\textbf{-}\lambda$$

$$\langle\{X \mapsto \lambda_A.Y\},\{((\lambda_A.Y)[\uparrow]\,\underline{3}) \ll^?_{\lambda\sigma} (\underline{2}(\underline{4}\,\underline{3}))\}\rangle$$

$$\Big\downarrow\ \mathbf{Normalise}_m$$

$$\langle\{X \mapsto \lambda_A.Y\},\{Y[\underline{3}\cdot\uparrow] \ll^?_{\lambda\sigma} (\underline{2}(\underline{4}\,\underline{3}))\}\rangle$$

$$\Big\downarrow\ \mathbf{Imit}$$

$$\langle\{Y \mapsto (\underline{2}\,H_1), X \mapsto (\lambda_A.(\underline{2}\,H_1))\},\{(\underline{2}\,H_1)[\underline{3}\cdot\uparrow] \ll^?_{\lambda\sigma} (\underline{2}(\underline{4}\,\underline{3}))\}\rangle$$

$$\Big\downarrow\ \mathbf{Normalise}_m$$

$$\langle\{Y \mapsto (\underline{2}\,H_1), X \mapsto (\lambda_A.(\underline{2}\,H_1))\},\{(\underline{2}\,H_1[\underline{3}\cdot\uparrow]) \ll^?_{\lambda\sigma} (\underline{2}(\underline{4}\,\underline{3}))\}\rangle$$

$$\Big\downarrow\ \mathbf{Dec}_m\textbf{-App}$$

$$\langle\{Y \mapsto (\underline{2}H_1), X \mapsto (\lambda_A.(\underline{2}H_1))\},\{H_1[\underline{3}\cdot\uparrow] =^?_{\lambda\sigma} (\underline{4}\,\underline{3})\}\rangle$$

$$\mathbf{Imit}\ \swarrow \qquad \searrow\ \mathbf{Proj}$$

$$T \qquad\qquad T'$$

where T' is given by:

$$\langle\{H_1 \mapsto \underline{1}, Y \mapsto (\underline{2}\,\underline{1}), X \mapsto (\lambda_A.(\underline{2}\,\underline{1}))\},\{\underline{1}[\underline{3}\cdot\uparrow] =^?_{\lambda\sigma} (\underline{4}\,\underline{3})\}\rangle$$

$$\Big\downarrow\ \mathbf{Normalise}_m$$

$$\langle\{H_1 \mapsto \underline{1}, Y \mapsto (\underline{2}\,\underline{1}), X \mapsto \lambda_A.(\underline{2}\,\underline{1})\},\{\underline{3} \ll^?_{\lambda\sigma} (\underline{4}\,\underline{3})\}\rangle$$

$$\Big\downarrow\ \mathbf{Dec}_m\textbf{-Fail}$$

$$Fail$$

and T is given by: $\langle\{H_1 \mapsto (\underline{4}\ \ H_2), Y \mapsto (\underline{2}(\underline{4}\ \ H_2)), X \mapsto \lambda_A.(\underline{2}(\underline{4}\ \ H_2))\},\{(\underline{4}\ \ H_2)[\underline{3}\cdot\ \uparrow] \ll^?_{\lambda\sigma} (\underline{4}\ \ \underline{3})\}\rangle$

$$\Big\downarrow\ \mathbf{Normalise}_m$$

$$\langle\{H_1 \mapsto (\underline{4}H_2), Y \mapsto (\underline{2}(\underline{4}H_2)), X \mapsto \lambda_A.(\underline{2}(\underline{4}H_2))\},\{(\underline{4}H_2[\underline{3}\cdot\uparrow]) \ll^?_{\lambda\sigma} (\underline{4}\,\underline{3})\}\rangle$$

$$\Big\downarrow\ \mathbf{Dec}_m\textbf{-App}$$

$$\langle\{H_1 \mapsto (\underline{4}H_2), Y \mapsto (\underline{2}(\underline{4}H_2)), X \mapsto \lambda_A.(\underline{2}(\underline{4}H_2))\},\{H_2[\underline{3}\cdot\uparrow] \ll^?_{\lambda\sigma} \underline{3}\}\rangle$$

$$\mathbf{Imit}\ \swarrow \qquad \searrow\ \mathbf{Proj}$$

$$T'' \qquad\qquad T'''$$

where T'' and T''' are, respectively, given by:

$$\langle\{H_2 \mapsto \underline{3}, H_1 \mapsto (\underline{4}\ \underline{3}), Y \mapsto (\underline{2}(\underline{4}\ \underline{3})), X \mapsto \lambda_A.(\underline{2}(\underline{4}\ \underline{3}))\}, \{\}\rangle$$

and

$$\langle\{H_2 \mapsto \underline{1}, H_1 \mapsto (\underline{4}\ \underline{1}), Y \mapsto (\underline{2}(\underline{4}\ \underline{1})), X \mapsto \lambda_A.(\underline{2}(\underline{4}\ \underline{1}))\}, \{\}\rangle$$

The higher-order substitution obtained from the grafting $X \mapsto \lambda_A.(\underline{2}(\underline{4}\ \underline{1}))$ is $X/\lambda_A.(\underline{2}(\underline{4}\ \underline{1}))$.

Remark 3.10
In the following we present a proof of correctness and completeness of the matching rules of Table 3. In these proofs, we identify the matching problem $\langle\sigma, M\rangle$ with the unification problem P built in the following way:

- $X =^?_{\lambda\sigma} a$ is an equation in P whenever $X \mapsto a$ is in σ;
- $a =^?_{\lambda\sigma} b$ is an equation in P whenever $a \ll^?_{\lambda\sigma} b$ is an equation in M.

Proposition 3.11 (Correctness)
The rules given in Table 3 are correct, i.e., if $\langle\sigma', M'\rangle$ can be derived from $\langle\sigma, M\rangle$ after an application of one of these rules then $\mathcal{M}_{\lambda\sigma}(\langle\sigma', M'\rangle) \subseteq \mathcal{M}_{\lambda\sigma}(\langle\sigma, M\rangle)$.

Proof.

1. **Dec$_m$-λ**: Let γ be a solution to the unification problem $P' \wedge a =^?_{\lambda\sigma} b$ that corresponds to the codification of the matching problem $\langle\sigma, M' \wedge a \ll^?_{\lambda\sigma} b\rangle$ as in Remark 3.10. Since for each grafting θ we have that $\theta\lambda_A.a =_{\lambda\sigma} \lambda_A.\theta a$, we conclude that γ is also a solution to $P' \wedge \lambda_A.a =^?_{\lambda\sigma} \lambda_A.b$, and hence a solution to $\langle\sigma, M\rangle$.
2. **Dec$_m$-App**: Let γ be a solution to $P' \wedge a_1 =^?_{\lambda\sigma} b_1 \wedge \ldots \wedge a_r =^?_{\lambda\sigma} b_r$ that corresponds to the codification of the matching problem $\langle\sigma, M' \wedge a_1 \ll^?_{\lambda\sigma} b_1 \wedge \ldots \wedge a_r \ll^?_{\lambda\sigma} b_r\rangle$ according to Remark 3.10. Since a grafting can be propagated over de Bruijn indexes i.e., $\gamma(\underline{n}\ a_1 \ldots a_r) =_{\lambda\sigma} (\underline{n}\ \gamma a_1 \ldots \gamma a_r)$, we conclude that γ is also a solution to $P' \wedge (\underline{n}\ a_1 \ldots a_r) =^?_{\lambda\sigma} (\underline{n}\ b_1 \ldots b_r)$ and hence a solution to $\langle\sigma, M\rangle$.
3. **Exp$_m$-λ**: Let P be the unification problem that codifies the matching problem $\langle\sigma, M\rangle$. We suppose that M contains at least one occurrence of the meta-variable X of type $A_1 \to \ldots \to A_k \to A$ (A atomic), for some $k \geq 0$. Let γ be a solution to the problem $P' \wedge X =^?_{\lambda\sigma} \lambda_{A_1} \ldots \lambda_{A_k}.Y$ that codifies the matching problem:

   $$\langle\{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\}\sigma, \{X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.Y\}M\rangle$$

   obtained from $\langle\sigma, M\rangle$ after an application of **Exp$_m$-λ**. Notice that P' can be obtained from P by replacing all occurrences of X for $\lambda_{A_1} \ldots \lambda_{A_k}.Y$ and hence, if γ is a solution to $P' \wedge X =^?_{\lambda\sigma} \lambda_{A_1} \ldots \lambda_{A_k}.Y$ then it is also a solution to P because γ has a grafting of the form $X \mapsto \lambda_{A_1} \ldots \lambda_{A_k}.a$ that solves the occurrences of X in P and all the other meta-variables are solved in the same way in P and P' by γ, except Y which occurs only in P'.
4. **Imit**: Let P be the unification problem that codifies the matching problem given by $\langle\sigma, M \wedge X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow] \ll^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)\rangle$. If P' is the unification problem that codifies the matching problem $\langle\sigma', M'\rangle$ obtained from $\langle\sigma, M \wedge X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow] \ll^?_{\lambda\sigma} (\underline{m}\ b_1 \ldots b_q)\rangle$ after an application of **Imit** then P' can be obtained from P

by replacing all occurrences of X for $(\underline{m-n+p}\ H_1 \dots H_q)$. Therefore, if γ is a solution to P' then it has graftings which solve all the meta-variables occurring in P and the meta-variables $H_1, \dots, H_q$ introduced after the application of **Imit** and hence γ is also a solution to P.

5. **Proj**: Let P be the unification problem that codifies the matching problem given by $\langle \sigma, M \wedge X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow] \ll^?_{\lambda\sigma} (\underline{m}\ b_1 \dots b_q) \rangle$. If P' is the unification problem that codifies the matching problem $\langle \sigma', M' \rangle$ obtained from $\langle \sigma, M \wedge X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow] \ll^?_{\lambda\sigma} (\underline{m}\ b_1 \dots b_q) \rangle$ after an application of **Proj** then P' can be obtained from P by replacing all occurrences of X for $\underline{j}$. Therefore, if γ is a solution to P' then it has graftings which solve all the meta-variables occurring in P' and hence γ is also a solution to P.
6. **Normalise**: The proof is straightforward because the rule normalise only converts a term to its η-lnf. ∎

PROPOSITION 3.12 (Completeness)
The rules given in Table 3 are complete, i.e., if $\langle \sigma', M' \rangle$ can be derived from $\langle \sigma, M \rangle$ after an application of one of these rules then $\mathcal{M}_{\lambda\sigma}(M) \subseteq \mathcal{M}_{\lambda\sigma}(M')$.

PROOF.

1. **Dec-λ**: Let P be the codification of the matching problem $\langle \sigma, M \wedge \lambda_A.a \ll^?_{\lambda\sigma} \lambda_A.b \rangle$ and γ be a solution to P. Since $\gamma\lambda_A.a = \lambda_A.\gamma a$, then γ is also a solution.
2. **Exp$_m$-λ**: Let θ be a $\lambda\sigma$-unifier of $\langle \sigma, P \rangle$ and $X \in \mathcal{T}var(P)$ such that $\Gamma \vdash X : A_1 \to \ldots \to A_k \to B$. Thus $X\theta = a : A_1 \to \ldots \to A_k \to B$ and we can assume that a is of the form $\lambda_{A_1} \dots \lambda_{A_k}.b$ with $b : B$. Define θ' such that for all $Z \in Dom(\theta)$, $\theta'(Z) = \theta(Z)$ and $Y\theta = b$ for a new variable $Y \notin Dom(\theta)$ of type B. Then θ' is a $\lambda\sigma$-unifier of $\langle \{X \mapsto \lambda_{A_1} \dots \lambda_{A_k}.Y\}, P \rangle$. Consequently θ is a $\lambda\sigma$-unifier of $\exists (Y : A_1. \cdots .A_k.\Gamma \vdash B), \langle \{X \mapsto \lambda_{A_1} \dots \lambda_{A_k}.Y\}, P \rangle$.

3. **Imit** and **Proj**: Let γ be a matcher of $P \cup \{X[a_1. \cdots .a_p. \uparrow^n] \ll^?_{\lambda\sigma} \underline{m}\ b_1 \dots b_q\}$, where X has atomic type and $m > n$. Let $X \mapsto \underline{k}\ c_1 \dots c_r \in \gamma$. Then, we have $P\{X \mapsto \underline{k}\ c_1 \dots c_r\} \cup \{(\underline{k}\ c_1 \dots c_r)[a_1. \cdots .a_p. \uparrow^n] \ll^?_{\lambda\sigma} \underline{m}\ b_1 \dots b_q\} \to^*_{\lambda\sigma} P\{X \mapsto \underline{k}\ c_1 \dots c_r\} \cup \{\underline{k}[a_1. \cdots .a_p. \uparrow^n] c_1[a_1. \cdots .a_p. \uparrow^n] \dots c_r[a_1. \cdots .a_p. \uparrow^n] \ll^?_{\lambda\sigma} \underline{m}\ b_1 \dots b_q\}$.
   Now we have two options: $k \leq p$ or $k > p$. In the first case, the previous problem reduces to $P\{X \mapsto \underline{k}\ c_1 \dots c_r\} \cup \{a_k\ c_1[a_1. \cdots .a_p. \uparrow^n] \dots c_r[a_1. \cdots .a_p. \uparrow^n] \ll^?_{\lambda\sigma} \underline{m}\ b_1 \dots b_q\}$ and γ is certainly a unifier of it. If $k > p$ then the problem reduces to $P\{X \mapsto \underline{k}\ c_1 \dots c_r\} \cup \{\underline{k-p+n}\ c_1[a_1. \cdots .a_p. \uparrow^n] \dots c_r[a_1. \cdots .a_p. \uparrow^n] \ll^?_{\lambda\sigma} \underline{m}\ b_1 \dots b_q\}$ and it has a solution if and only if $k - p + n = m$ and thus $k = m - n + p$ at the condition that $k > p \Leftrightarrow m - n + p > p \Leftrightarrow m > n$, which gives the condition asserted in the rule **Imit**. ∎

4 Third-Order Matching

In this section we prove the decidability of third-order matching problems for the whole language of the $\lambda\sigma$-calculus [ACCL91]. Our result is achieved by adapting the techniques in [Dow94] to the language of the simply typed $\lambda\sigma$-calculus. As a consequence we get as well decidability of second-order matching for the whole language of the simply typed $\lambda\sigma$-calculus. We start by defining $\lambda\sigma$-Böhm trees for $\lambda\sigma$-terms and presenting relevant results related to this notion. In subsection 4.2, we show how a matching problem in the language of $\lambda\sigma$ is translated to an interpolation problem. In subsection 4.3, we show how to decide third-order interpolation problems.

4.1 Finite $\lambda\sigma$-Böhm Trees

We introduce finite $\lambda\sigma$-Böhm trees (or simply $\lambda\sigma$-Böhm trees) for simply typed $\lambda\sigma$-terms and set the ground for our main result.

Definition 4.1 (Occurrence, Tree, Occurrence Path, $\lambda\sigma$-Böhm Trees (of a $\lambda\sigma$-term))

- An *occurrence* is a list of strictly positive integers $\alpha = \langle s_1, \ldots, s_n\rangle$. The number n is called the *length* of the occurrence α, written $length(\alpha)$.

- A *tree domain* D is a non empty finite set of occurrences such that if $\alpha\langle n\rangle \in D$ then $\alpha \in D$ and if $n \neq 1$ then also $\alpha\langle n-1\rangle \in D$. A *tree* is a function from a tree domain D to a set L, called the set of labels of the tree. The occurrence $\langle\ \rangle$ is called the *root* of the tree and $\alpha\langle n\rangle$ is called the n-th son of the occurrence α. The number of sons of an occurrence α is the greatest integer n such that $\alpha\langle n\rangle \in D$. A leaf is an occurrence which has no sons. The *depth* of a tree is the length of the longest occurrence in its domain.

- Let T be a tree and $\alpha = \langle s_1, \ldots, s_n\rangle$ be an occurrence in T. The *path of* α is the set of occurrences $\{\langle s_1, \ldots, s_k\rangle | k \leq n\}$.

- A $\lambda\sigma$-Böhm tree is a tree whose nodes are labelled with pairs $\langle l, v\rangle$ such that l is a positive integer and v is a well-typed $\lambda\sigma$-term.

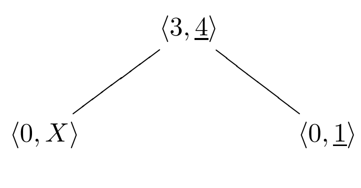
- Let $a = \lambda_{A_1} \cdots \lambda_{A_k}.(h\ b_1 \cdots b_m)$ be a well-typed term in $\lambda\sigma$-nf. The Böhm tree of a is recursively defined as the tree whose root is labelled with the pair $\langle k, h\rangle$ and whose sons are the $\lambda\sigma$-Böhm trees of:

  1. b_1,…,b_m, if h is a de Bruijn index;

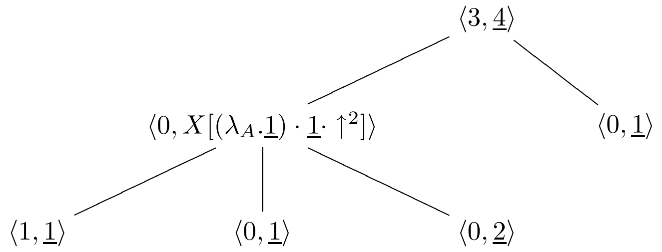
  2. $a_1, \ldots, a_p, b_1, \ldots, b_m$, if h is the closure of a meta-variable of the form $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$, where $a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n$ is a substitution in $\lambda\sigma$-nf.

  As in [Dow94], $|a|$ denotes the depth of the $\lambda\sigma$-Böhm tree of the $\lambda\sigma$-nf of term a.

Example 4.2
The $\lambda\sigma$-Böhm tree of the term $\lambda_A\lambda_A\lambda_A.(\underline{4}\ X\ \underline{1})$ is given by:



The $\lambda\sigma$-Böhm tree of the term $\lambda_A\lambda_A\lambda_A.(\underline{4}\ (X[(\lambda_A.\underline{1}) \cdot \underline{1} \cdot \uparrow^2]\ \underline{2})\ \underline{1})$ is given by:



According to the definition of $\lambda\sigma$-Böhm trees, the $\lambda\sigma$-terms $(X[\underline{2} \cdot \underline{1} \cdot \uparrow^2]\ \underline{3})$ and $(X\ \underline{2}\ \underline{1}\ \underline{3})$ have the same $\lambda\sigma$-Böhm tree, except that the labels in the root are different. The labels are essential to identify the term that originated the given $\lambda\sigma$-Böhm tree.

Definition 4.3 (Free occurrence)
A free occurrence of a de Bruijn index $\underline{i}$ in the $\lambda\sigma$-term a is defined by:

1. If $a = \underline{i}$ then $\underline{i}$ occurs free in a.
2. If $a = \lambda_A.b$ and $\underline{i}$ occurs free in b then $\underline{i+1}$ occurs free in a.
3. If $a = (b\ c)$ and $\underline{i}$ occurs free in b or c (or in both) then $\underline{i}$ occurs free in a.
4. If $a = X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ and $\underline{i}$ occurs free in a_i, for any $1 \leq i \leq p$, or $i > n$ then $\underline{i}$ occurs free in a.

Definition 4.4 (Relevant term)
A $\lambda\sigma$-term $a = \lambda_{A_1} \ldots \lambda_{A_k}.b$ is *relevant in its i-th* $(1 \leq i \leq k)$ *argument* if the index $\underline{k-i+1}$ occurs free in b.

The following lemmas allow us to establish when $|a| \leq |a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]|$ where the terms $a_1, \ldots, a_p$ are at most second-order. The motivation for that comes from the fact that if $X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ b_1 \ldots b_q \ll^?_{\lambda\sigma} b$ is a matching equation and σ is a solution to this equation with $X\sigma = \lambda_{B_1} \ldots \lambda_{B_q}.t$ then we have that

$$|t[b_q \cdot \ldots \cdot b_1 \cdot a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]| = |b|.$$

If it is always the case that $|t| \leq |t[b_q \cdot \ldots \cdot b_1 \cdot a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]|$, i.e.,

$$|t| \leq |b| \tag{4.1}$$

then an enumeration of the terms t satisfying (4.1) would give a decision procedure for third-order matching. This would not be the case if $|a_i| = 0$, for some $1 \leq i \leq p$. To solve this problem we show that if a matching problem is solvable then there is a solution which is limited by a number that only depends on the initial problem.

Lemma 4.5
Let a be a $\lambda\sigma$-nf and $a_1, \ldots, a_p$ be $\lambda\sigma$-terms of atomic type, $n \geq 0$ such that the term $a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ is well typed. For all $1 \leq j \leq p$, if the de Bruijn index $\underline{j}$ occurs free in a then the $\lambda\sigma$-nf of $a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ contains a_j as a sub-term, and hence $|a| \leq |a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]|$.

Proof. Note that all possible free occurrences of $\underline{j}$ in a are in non functional positions because they have atomic types; i.e., all these free occurrences are leaves. Therefore, the normalisation of $a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ does not include any simulation of β-reduction. Thus, if β is a free occurrence of $\underline{j}$ in a, then after a σ-normalisation, every occurrence α in the $\lambda\sigma$-Böhm tree of the $\lambda\sigma$-nf of a_j will become the occurrence $\beta\alpha$ in the $\lambda\sigma$-Böhm tree of the $\lambda\sigma$-nf of $a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$. This means that the $\lambda\sigma$-Böhm tree of a_j is a sub-tree of the $\lambda\sigma$-Böhm tree of the $\lambda\sigma$-nf of $a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$. ∎

If a substitution s is a list of de Bruijn indexes, then we have a similar result for which the order of the indexes is not relevant. This is formalised in the next lemma:

Lemma 4.6
Let a and $a[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]$ be well typed and in $\lambda\sigma$-nf with $k, n \geq 0$ and $i_k \neq n$. Then $|a| \leq |a[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]|$.

Proof. By structural induction on a:

- If $a = \underline{r}$ and $r \leq k$ then $|\underline{r}[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]| = |\underline{i_r}| = 0$. Otherwise, $|\underline{r}[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]| = |\underline{r - k + n}| = 0$
- If $a = X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^m]$ with $p, m \geq 0$, then:
  - if $p = m = 0$ then $X = X[id] = X[\uparrow^0]$ and hence, by definition of $\lambda\sigma$-Böhm trees we have that $|X| \leq |X[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]|$.
  - if $p > 0$ then for all $m \geq 0$ we have that $|X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^m]| \stackrel{def.}{=} 1 + max(|a_1|, \ldots, |a_p|)$. Moreover,

  $$a[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n] = X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^m][\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n] \to_\sigma^* \begin{cases} t_1, \text{ if } m \leq k; \\ t_2, \text{ if } m > k; \end{cases}$$

  where $t_1 = X[a_1[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n] \cdot \ldots \cdot a_p[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n] \cdot \underline{i_{m+1}} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]$ and $t_2 = X[a_1[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n] \cdot \ldots \cdot a_p[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n] \cdot \uparrow^{m-k+n}]$.
  In the first case, i.e., where $m \leq k$, we have by IH that for all $1 \leq j \leq p$,

  $$|a_j| \leq |a_j[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]| \tag{4.2}$$

  and by definition of $\lambda\sigma$-Böhm trees, we have that:
  $|X[a_1[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n] \cdot \ldots \cdot a_p[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n] \cdot \underline{i_{m+1}} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]| =$
  $1 + max(|a_1[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]|, \ldots, |a_p[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]|) \leq$
  $1 + max(|a_1|, \ldots, |a_p|) = |X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^m]|$.

  In the second case, i.e., where $m > k$, by IH, for all $1 \leq j \leq p$, we have (4.2) and then $|X[a_1[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n] \cdot \ldots \cdot a_p[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n] \cdot \uparrow^{m-k+n}]| =$
  $1 + max(|a_1[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]|, \ldots, |a_p[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]|) \leq$
  $1 + max(|a_1|, \ldots, |a_p|) = |X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^m]|$.
- If $a = \lambda_B.b$ then by IH we have that $|b| \leq |b[\underline{1} \cdot \underline{i_1}[\uparrow] \cdot \ldots \cdot \underline{i_k}[\uparrow] \cdot \uparrow^{n+1}]|$. In addition, by the definition of $\lambda\sigma$-Böhm trees, we have that $|b[\underline{1} \cdot \underline{i_1}[\uparrow] \cdot \ldots \cdot \underline{i_k}[\uparrow] \cdot \uparrow^{n+1}]| = |\lambda_B.b[\underline{1} \cdot \underline{i_1}[\uparrow] \cdot \ldots \cdot \underline{i_k}[\uparrow] \cdot \uparrow^{n+1}]| = |(\lambda_B.b)[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]|$ and the lemma holds.
- If $a = (b\ c)$ then, by IH, $|b| \leq |b[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]|$ and $|c| \leq |c[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]|$. Since a is in $\lambda\sigma$-nf then $|a| = 1 + max(|b|, |c|) \leq 1 + max(|b[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]|, |c[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]|) = |a[\underline{i_1} \cdot \ldots \cdot \underline{i_k} \cdot \uparrow^n]|$.

∎

PROPOSITION 4.7
Let a be a $\lambda\sigma$-nf and $a_1, \ldots, a_p$ be $\lambda\sigma$-terms of at most second-order, and $n \geq 0$ such that $a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ is a well typed term. For all $1 \leq j \leq p$, if the de Bruijn index $\underline{\mathrm{j}}$ occurs free in a then $|a_j| \leq |a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]|$.

PROOF. The proof is by structural induction on a:

- If $a = \underline{j}$ then $|\underline{j}[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]| = |a_j|$.
- If $a = X[b_1 \cdot \ldots \cdot b_q \cdot \uparrow^m]$ then $a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] = X[b_1 \cdot \ldots \cdot b_q \cdot \uparrow^m][a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \rightarrow^*_\sigma$
  $\begin{cases} X[b_1[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \cdot \ldots \cdot b_q[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \cdot a_{m+1} \cdot \ldots \cdot a_p \cdot \uparrow^n], & \text{if } m < p; \\ X[b_1[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \cdot \ldots \cdot b_q[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \cdot \uparrow^{m-p+n}], & \text{if } m \geq p. \end{cases}$
  By IH we have that for all $1 \leq k \leq q$, if the de Bruijn index $\underline{j}$ occurs free in b_k then $|a_j| \leq |b_k[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]| \leq |t| = |a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]|$, where t is either
  $$X[b_1[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \cdot \ldots \cdot b_q[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \cdot a_{m+1} \cdot \ldots \cdot a_p \cdot \uparrow^n]$$
  or
  $$X[b_1[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \cdot \ldots \cdot b_q[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \cdot \uparrow^{m-p+n}].$$
- If $a = \lambda_B.b$ then $a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] = (\lambda_B.b)[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \rightarrow^*_\sigma$
  $\lambda_B.b[\underline{1} \cdot a_1[\uparrow] \cdot \ldots \cdot a_p[\uparrow] \cdot \uparrow^{n+1}]$ and by IH we have that, for all $1 \leq j \leq p+1$, if the de Bruijn index $\underline{\mathrm{j}}$ occurs free in b then
  $$|b_j| \leq |b[\underline{1} \cdot a_1[\uparrow] \cdot \ldots \cdot a_p[\uparrow] \cdot \uparrow^{n+1}]| \tag{4.3}$$
  where $b_j = \begin{cases} \underline{1} & \text{, if } j = 1 \\ a_{j-1}[\uparrow] & \text{, if } 1 < j \leq p+1. \end{cases}$
  Therefore, using Lemma 4.6 and equation (4.3), we have that:
  $j > 1$: $\quad |a_{j-1}| = |a_{j-1}[\uparrow]| \leq |b[\underline{1} \cdot a_j[\uparrow] \cdot \ldots \cdot a_p[\uparrow] \cdot \uparrow^{n+1}]| =$
  $|\lambda_B.b[\underline{1} \cdot a_1[\uparrow] \cdot \ldots \cdot a_p[\uparrow] \cdot \uparrow^{n+1}]| = |(\lambda_B.b)[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]|$.

  $j = 1$: $\quad |\underline{1}| \leq |b[\underline{1} \cdot a_j[\uparrow] \cdot \ldots \cdot a_p[\uparrow] \cdot \uparrow^{n+1}]| =$
  $|\lambda_B.b[\underline{1} \cdot a_1[\uparrow] \cdot \ldots \cdot a_p[\uparrow] \cdot \uparrow^{n+1}]| = |(\lambda_B.b)[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]|$.
- If $a = (b\ c)$ then $a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] = (b\ c)[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \rightarrow_\sigma$
  $(b[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ c[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n])$ and now we have the following cases according to the existence of a free occurrence of a de Bruijn index $\underline{j}$ with $1 \leq j \leq p$ in b and/or in c:
  1. Assume $\underline{j}$ with $1 \leq j \leq p$ occurs free in b. If this occurrence is in the head then:
     $(b[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ c[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]) =$
     $((\underline{j}\ b_1 \ldots b_k)[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ c[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]) \rightarrow^*_\sigma$
     $(a_j\ b_1[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \ldots b_k[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ c[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]) =$
     $(\lambda_{B_1} \ldots \lambda_{B_{k+1}}.d\ b_1[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \ldots b_k[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ c[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]) \rightarrow^*_{\lambda\sigma}$
     $d[c[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \cdot b_k[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \cdot \ldots \cdot b_1[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \cdot id]$.
     Since a_j is, at most second-order, we conclude that the types $B_1, \ldots, B_{k+1}$ are atomic and hence $c[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$, $b_k[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$, $\ldots$, $b_1[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ have atomic types. By Lemma 4.5 $|a_j| = |\lambda_{B_1} \ldots \lambda_{B_{k+1}}.d| = |d| \leq |d[c[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ $\cdot b_k[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \cdot \ldots \cdot b_1[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] \cdot id]| = |(b[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ c[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n])|$.

If the occurrence of $\underline{j}$ is not in the head of b then the $\lambda\sigma$-nf of

$$(b[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ c[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n])$$

is of the form $(\overline{b}\ \overline{c})$, where $\overline{b}$ (resp. $\overline{c}$) is the $\lambda\sigma$-nf of $b[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ (resp. $c[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$). Therefore, $|b[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]| \leq |(b[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ c[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n])|$ and by IH, we have that $|a_j| \leq |b[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]|$ because $\underline{j}$ occurs free in b.

2. Suppose that $\underline{j}$ with $1 \leq j \leq p$ occurs free in c. By IH, we have that $|a_j| \leq |c[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]| \leq |(b[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ c[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n])|$. ∎

Proposition 4.8

Let a be a $\lambda\sigma$-normal term and $a_1, \ldots, a_p$ be $\lambda\sigma$-terms of at most second order, $n \geq 0$ and $a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ be a well typed $\lambda\sigma$-term. Let $\alpha = \langle s_1, \ldots, s_n \rangle$ be an occurrence in the $\lambda\sigma$-Böhm tree of a and $\beta_1 = \langle s_1, \ldots, s_{k_1} \rangle$, $\ldots$, $\beta_l = \langle s_1, \ldots, s_{k_l} \rangle$ be all the occurrences of free de Bruijn indexes $\underline{j}$ with $1 \leq j \leq p$ that are in the path of α. For each $1 \leq i \leq l$, if the term a_j is relevant in its r-th argument and $|a_j| \neq 0$, where r is the position of the son of β in the path of α, i.e., $r = s_{k_i+1}$ then there exists an occurrence α' in the $\lambda\sigma$-Böhm tree of the $\lambda\sigma$-nf of $a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ whose length is greater than or equal to the length of α.

Proof. By induction on the number of occurrences of free de Bruijn indexes $\underline{j}$ with $1 \leq j \leq p$ in the path of α. We consider occurrences from the lowest to the highest in the $\lambda\sigma$-Böhm tree of a. Let

$$\alpha = \beta_l \beta_{l-1} \ldots \beta_2 \beta_1 \gamma \tag{4.4}$$

be an occurrence in the $\lambda\sigma$-Böhm tree of a. If there is no free occurrence of $\underline{j}$ $(1 \leq j \leq p)$ in the path of α, i.e., $l = 0$ in (4.4), then α is also an occurrence in the $\lambda\sigma$-Böhm tree of the $\lambda\sigma$-nf of $a[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$ and then we take $\alpha' = \alpha = \gamma$.

If there exists exactly one free occurrence of $\underline{j}$ $(1 \leq j \leq p)$ in the path of α, then $\alpha = \beta_1 \langle r \rangle \gamma'$ where $\gamma = \langle r \rangle \gamma'$. Assume that a at the occurrence β_1 has the form

$$\lambda_{B_1} \ldots \lambda_{B_v}.(\underline{j+v}\ e_1 \ldots e_u). \tag{4.5}$$

In equation (4.5), we assume that $\lambda_{B_1}, \ldots, \lambda_{B_v}$ are all the abstractors binding the free occurrence of the index $\underline{j}$, corresponding to the index $\underline{j+v}$ in this equation. Note that with this assumption, we do not lose generality because the particular representation of this index is not important in the sense that if there exists more abstractors, the representation $\underline{j+v}$ will change to, say $\underline{j'}$, but the ideas that follows can be repeated exactly the same way for $\underline{j'}$.

The propagation of a substitution s inside a term changes the structure of s only when the substitution finds an abstraction (i.e., when a rule (Abs) is applied): in this case a substitution of the form $a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n$ changes to $\underline{1} \cdot a_1[\uparrow] \cdot \ldots \cdot a_p[\uparrow] \cdot \uparrow^{n+1}$. Nevertheless the relevance (cf. Def 4.4) of the terms $a_1, \ldots, a_p$ remains unchanged because its bound indexes are not affected by the introduced $\uparrow$. Therefore, according to our assumptions the propagation of the substitution $a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n$ over a until its sub-term that labels the occurrence β_1 does not affect the substitution. In this way, we get a sub-term of the form:

$$(\lambda_{B_1} \ldots \lambda_{B_v}.(\underline{j+v}\ e_1 \ldots e_u))[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]. \tag{4.6}$$

The propagation of the substitution over the λ's in (4.6) generates the new sub-term:
$\lambda_{B_1}\ldots\lambda_{B_v}.(\underline{j+v}\; e_1\ldots e_u)[\underline{1}\cdot\ldots\cdot\underline{v}\cdot a_1[\uparrow^v]\cdot\ldots\cdot a_p[\uparrow^v]\cdot\uparrow^{n+v}]\to_\sigma^*$

$$\lambda_{B_1}\ldots\lambda_{B_v}.(a_j[\uparrow^v]\; e'_1\ldots e'_u) \qquad (4.7)$$

where $e'_i = e_i[\underline{1}\cdot\ldots\cdot\underline{v}\cdot a_1[\uparrow^v]\cdot\ldots\cdot a_p[\uparrow^v]\cdot\uparrow^{n+v}]$, for all $1\le i\le u$.

By hypothesis $a_j = \lambda_{D_1}\ldots\lambda_{D_u}.d$ is relevant in the r-th argument (which corresponds to the son of β_1 which is in the path of α) and $|a_j|\neq 0$. In this way, $a_j[\uparrow^v] = \lambda_{D_1}\ldots\lambda_{D_u}.d'$ is relevant in the r-th argument as well, where d' is obtained by normalising $d[\uparrow^v]$. Hence, the term (4.7) normalises to $\lambda_{B_1}\ldots\lambda_{B_v}.d'[e'_u\cdot\ldots\cdot e'_1\cdot id]$. By hypothesis, $|a_j|\neq 0$ and hence $|d'|\neq 0$. Moreover, since the terms $a_1,\ldots,a_p$ are at most second-order, we conclude that the types $D_1,\ldots,D_u$ are atomic and the free occurrences of $\underline{j}$ in d' cannot be in the root of the term. Let δ be any of the occurrence of $\underline{j}$ in the $\lambda\sigma$-Böhm tree of d'. Since these occurrences are all leaves, take $\alpha'=\beta_1\delta\gamma$. From the fact that $length(\delta)\ge 1$, we conclude that $length(\alpha')\ge length(\alpha)$. ∎

COROLLARY 4.9
Let a be a $\lambda\sigma$-nf, $a_1,\ldots,a_p$ be $\lambda\sigma$-normal terms of at most second-order, $n\ge 0$ and $a[a_1\cdot\ldots\cdot a_p\cdot\uparrow^n]$ be a well typed term. If for all $1\le i\le p$, the term a_i is relevant in all its arguments and $|a_i|\neq 0$ then

$$|a|\le |a[a_1\cdot\ldots\cdot a_p\cdot\uparrow^n]|$$

PROOF. Let α be the longest occurrence in the $\lambda\sigma$-Böhm tree of a. By Proposition 4.8, there exists an occurrence α' in the $\lambda\sigma$-Böhm tree of the $\lambda\sigma$-nf of $a[a_1\cdot\ldots\cdot a_p\cdot\uparrow^n]$ which is, at least, as long as α. ∎

4.2 From Matching Problems to Interpolation Problems

We describe how matching problems can be converted into equivalent interpolation problems in the language of the $\lambda\sigma$-calculus. This conversion uses a new symbol $\diamond$, called the dummy symbol, which is a fresh symbol not belonging to the language of the $\lambda\sigma$-calculus (cf. [ARdMK05, Bor95]).

DEFINITION 4.10
Let $a \ll^?_{\lambda\sigma} b$ be a matching equation and σ be a ground solution to this equation, i.e., the $\lambda\sigma$-normal form of $a\sigma$ is $\beta\eta$-equivalent to b. We define the interpolation problem $\Phi(a\ll^?_{\lambda\sigma} b,\sigma)$ inductively over the number of occurrences of a as follows:

- If $a=\lambda_A.c$ then b is also an abstraction of the form $\lambda_A.d$ and then σ is also a solution to $c\ll^?_{\lambda\sigma} d$ and we let $\Phi(a\ll^?_{\lambda\sigma} b,\sigma)=\Phi(c\ll^?_{\lambda\sigma} d,\sigma)$.
- If $a=(\underline{k}\; c_1\ldots c_m)$ then b is also of the form $(\underline{k}\; d_1\ldots d_m)$ because $a\ll^?_{\lambda\sigma} b$ is solvable and we let $\Phi(a\ll^?_{\lambda\sigma} b,\sigma)=\bigcup_i \Phi(c_i\ll^?_{\lambda\sigma} d_i,\sigma)$.
- If $a=(X[a_1\cdot\ldots\cdot a_p\cdot\uparrow^n]\; b_1\ldots b_q)$ then we let

$$\Phi(a\ll^?_{\lambda\sigma} b,\sigma)=\{(X[a_1\sigma\cdot\ldots\cdot a_p\sigma\cdot\uparrow^n]\; b_1\sigma\ldots b_q\sigma)\ll^?_{\lambda\sigma} b\}\bigcup_i H_i,\ \text{where}$$

$$H_i = \begin{cases} \Phi(a_i \ll^?_{\lambda\sigma} a_i\sigma, \sigma) & \text{if } \diamond \text{ occurs in the } \lambda\sigma\text{-normal form of} \\ & (X[a_1 \cdot \ldots \cdot a_{i-1} \cdot \diamond \cdot a_{i+1} \cdot \ldots \cdot a_p \cdot \uparrow^n]\ b_1 \ldots b_q)\sigma; \\ \Phi(b_i \ll^?_{\lambda\sigma} b_i\sigma, \sigma) & \text{if } \diamond \text{ occurs in the } \lambda\sigma\text{-normal form of} \\ & (X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ b_1 \ldots b_{i-1} \diamond b_{i+1} \ldots b_q)\sigma; \\ \emptyset & \text{otherwise.} \end{cases}$$

Proposition 4.11
Let $a \ll^?_{\lambda\sigma} b$ be a matching equation and σ be a ground solution to this equation. Then the substitution σ is a solution to $\Phi(a \ll^?_{\lambda\sigma} b, \sigma)$ and, conversely, if σ' is a solution to $\Phi(a \ll^?_{\lambda\sigma} b, \sigma)$ then σ' is also a solution to the matching equation $a \ll^?_{\lambda\sigma} b$.

Proof. By induction on the structure of the terms on the left-hand side of all the equations occurring in the problem. Let σ be a ground solution of $a \ll^?_{\lambda\sigma} b$.

- If a is an abstraction of the form $\lambda_A.c$ then b is also an abstraction of the form $\lambda_A.d$ and σ is a ground solution of $c \ll^?_{\lambda\sigma} d$. By IH σ is a solution to $\Phi(c \ll^?_{\lambda\sigma} d, \sigma)$ and hence also a solution to all the equations of $\Phi(a \ll^?_{\lambda\sigma} b, \sigma)$.
- If a is an application of the form $(\underline{k}\ c_1 \ldots c_m)$ then b is also an application of the form $(\underline{k}\ d_1 \ldots d_m)$ and σ is also a ground solution to the equations $c_1 \ll^?_{\lambda\sigma} d_1, \ldots, c_m \ll^?_{\lambda\sigma} d_m$. By IH, we have that σ is a solution to $\bigcup_i \Phi(c_i \ll^?_{\lambda\sigma} d_i, \sigma)$ and hence σ is also a solution to $\Phi(a \ll^?_{\lambda\sigma} b, \sigma)$.
- If a is of the form $(X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ b_1 \ldots b_q)$ then σ is a solution to

$$X[a_1\sigma \cdot \ldots \cdot a_p\sigma \cdot \uparrow^n]\ b_1\sigma \ldots b_q\sigma \ll^?_{\lambda\sigma} b$$

  and $a_1 \ll^?_{\lambda\sigma} a_1\sigma, \ldots, a_p \ll^?_{\lambda\sigma} a_p\sigma$, $b_1 \ll^?_{\lambda\sigma} b_1\sigma, \ldots, b_q \ll^?_{\lambda\sigma} b_q\sigma$.
  By IH, σ is also a ground solution to

$$\bigcup_{i=1}^{p} \Phi(a_i \ll^?_{\lambda\sigma} a_i\sigma, \sigma) \text{ and } \bigcup_{i=1}^{q} \Phi(b_i \ll^?_{\lambda\sigma} b_i\sigma, \sigma)$$

  and therefore, σ is a solution to

$$\{X[a_1\sigma \cdot \ldots \cdot a_p\sigma \cdot \uparrow^n]\ b_1\sigma \ldots b_q\sigma \ll^?_{\lambda\sigma} b\} \bigcup_i H_i \text{ where } H_i \text{ is as in Def. 4.10.}$$

Conversely, let σ' be a ground solution to $\Phi(a \ll^?_{\lambda\sigma} b, \sigma)$:

- If a is an abstraction of the form $\lambda_A.c$ then b is also of the form $\lambda_A.d$ and $\Phi(c \ll^?_{\lambda\sigma} d, \sigma) \subseteq \Phi(a \ll^?_{\lambda\sigma} b, \sigma)$ and hence σ' is also a solution to $c \ll^?_{\lambda\sigma} d$ and therefore, a solution to $a \ll^?_{\lambda\sigma} b$.
- If a is an application of the form $(\underline{k}\ c_1 \ldots c_m)$ then b is also of the form $(\underline{k}\ d_1 \ldots d_m)$ and, since $\bigcup_i \Phi(c_i \ll^?_{\lambda\sigma} d_i, \sigma) \subseteq \Phi(a \ll^?_{\lambda\sigma} b, \sigma)$, then σ' is also a solution to all the equations $c_i \ll^?_{\lambda\sigma} d_i$ for $i = 1, \ldots, m$ and hence, σ' is also a solution to $a \ll^?_{\lambda\sigma} b$.
- If a is of the form $(X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ b_1 \ldots b_q)$ then σ' is also a solution to $(X[a_1\sigma \cdot \ldots \cdot a_p\sigma \cdot \uparrow^n]\ b_1\sigma \ldots b_q\sigma) \ll^?_{\lambda\sigma} b$ and to $\bigcup_i \Phi(b_i \ll^?_{\lambda\sigma} b_i\sigma, \sigma)$, i.e., $b_i\sigma' =_{\lambda\sigma}$

$b_i\sigma$ for all $i = 1, \ldots, q$. Therefore,
$(X\sigma'[a_1\sigma \cdot \ldots \cdot a_p\sigma \cdot \uparrow^n]\ b_1\sigma \ldots b_q\sigma) =_{\lambda\sigma} b$
$(X\sigma'[a_1\sigma \cdot \ldots \cdot a_p\sigma \cdot \uparrow^n]\ b_1\sigma' \ldots b_q\sigma') =_{\lambda\sigma} b$
$(X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ b_1 \ldots b_q)\sigma' =_{\lambda\sigma} b$
$a\sigma' =_{\lambda\sigma} b$ ∎

Definition 4.12
Let Ψ be a third-order matching problem and σ be a solution to Ψ. We let $\Phi(\Psi, \sigma)$ be the following third-order interpolation problem:

$$\Phi(\Psi, \sigma) = \bigcup_{a \ll^?_{\lambda\sigma} b \in \Psi} \Phi(a \ll^?_{\lambda\sigma} b, \sigma).$$

4.3 Third-Order Interpolation Problems

We show how interpolation problems are decided concluding the decidability of third-order matching.

Definition 4.13 (Occurrence accessible w.r.t. an equation)
Consider a matching equation of the form $(X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ b_1 \ldots b_q) \ll^?_{\lambda\sigma} b$ and the term $t = \lambda_{C_1} \ldots \lambda_{C_q}.u$ with the same type and context of X. The set of occurrences in the $\lambda\sigma$-Böhm tree of t *accessible* with respect to the equation

$$(X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\ b_1 \ldots b_q) \ll^?_{\lambda\sigma} b$$

is inductively defined as:

- the root of the $\lambda\sigma$-Böhm tree of t is accessible.
- if α is an accessible occurrence labelled with a de Bruijn index $\underline{j}$ with $1 \leq j \leq p+q$ and d_j is relevant in its r-th argument then the occurrence $\alpha\langle r\rangle$ is accessible, where: $d_j = \begin{cases} a_j & \text{if } q < j \leq p+q; \\ b_{q-i+1} & \text{if } 1 \leq j \leq q. \end{cases}$
- if α is an accessible occurrence labelled with a de Bruijn index greater than $p+q$ or with a meta-variable then all the sons of α are accessible.

Definition 4.14 (Occurrence accessible w.r.t. an interpolation problem [Dow94])
An occurrence is accessible with respect to an interpolation problem if it is accessible with respect to one of the equations of this problem.

Definition 4.15 ($\lambda\sigma$-term accessible w.r.t. to an interpolation problem)
A $\lambda\sigma$-term is accessible with respect to an interpolation problem if all occurrences of its $\lambda\sigma$-Böhm tree which are not leaves are accessible with respect to this problem.

Definition 4.16 (Accessible solution built from a solution)
Let Φ be an interpolation problem and let σ be a ground solution to this problem. For each meta-variable X occurring in the equations of Φ consider the $\lambda\sigma$-term t such that $\{X \mapsto t\} \subseteq \sigma$. In the $\lambda\sigma$-Böhm tree of t, we prune all occurrences non accessible (that are not leaves) with respect to the equations of Φ in which X has an occurrence and put $\lambda\sigma$-Böhm trees of ground terms of depth 0 of the expected type as leaves. Call t' the term whose $\lambda\sigma$-Böhm tree is obtained in this way and $\widehat{\sigma}$ the resulting substitution.

PROPOSITION 4.17
Let Φ be an interpolation problem and let σ be a ground solution to Φ. Then the accessible solution $\widehat{\sigma}$, built from σ, is a solution to Φ.

PROOF. Let

$$(X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\, b_1 \ldots b_q) \ll^?_{\lambda\sigma} b \tag{4.8}$$

be an equation of Φ, σ be a ground solution to Φ, $t = X\sigma = \lambda_{B_1} \ldots \lambda_{B_q}.d$ and $t' = X\widehat{\sigma} = \lambda_{B_1} \ldots \lambda_{B_q}.d'$. It is enough to prove that

$$d[b_q \cdot \ldots \cdot b_1 \cdot a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n] =_{\lambda\sigma} d'[b_q \cdot \ldots \cdot b_1 \cdot a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]$$

because the left-hand side of the above equation reduces to b and the right-hand side is $\lambda\sigma$-equivalent to $(X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\, b_1 \ldots b_q)\widehat{\sigma}$.

By construction, the $\lambda\sigma$-Böhm trees of d and d' differ only on the non accessible occurrences. So, let $\beta\langle r\rangle$ be a non accessible occurrence in a path in the $\lambda\sigma$-Böhm tree of t such that the occurrence β is accessible w.r.t the equation (4.8). From the definition of accessible occurrence, we conclude that β is labelled with a de Bruijn index $\underline{j}$ $(1 \leq j \leq p+q)$ and d_j is not relevant in its r-th argument, where

$$d_j = \begin{cases} a_{j-q}, & \text{if } q < j \leq p+q; \\ b_{q-j+1}, & \text{if } 1 \leq j \leq q. \end{cases}$$

During the normalisation step, $\underline{j}$ will be replaced by d_j and since d_j is not relevant in its r-th argument, the label of $\beta\langle r\rangle$ will disappear during the normalisation. ∎

PROPOSITION 4.18
Let Φ be an interpolation problem, σ a solution to Φ and $\widehat{\sigma}$ the accessible solution built from σ. If h is the maximum depth in the $\lambda\sigma$-Böhm tree of the right-hand side of the equations of Φ and $t = X\widehat{\sigma} = \lambda_{C_1} \ldots \lambda_{C_q}.u$ (u atomic) then there are at most $h+1$ occurrences of de Bruijn indexes $\underline{k}$ with $k > q$ on a path in the $\lambda\sigma$-Böhm tree of t.

PROOF. Notice that the occurrences in the $\lambda\sigma$-Böhm tree of t and those in the $\lambda\sigma$-Böhm tree of u are the same, except that the labels at the root differ. Suppose that there exists an occurrence α in the $\lambda\sigma$-Böhm tree of u such that there exists more than $h+1$ occurrences of de Bruijn indexes $\underline{k}$ with $k > q$ in the path of α. Let β be the $h+1$-th occurrence of such index. The occurrence β is not a leaf and hence it is accessible w.r.t. an equation of Φ, say $(X[a_1 \cdot \ldots \cdot a_p \cdot \uparrow^n]\, b_1 \ldots b_q) \ll^?_{\lambda\sigma} b$.

For each occurrence $\gamma = \langle s_1, \ldots, s_l\rangle$, in the path of β, labeled with a de Bruijn index $\underline{j}$ with $1 \leq j \leq q$, we have that b_{q-j+1} is relevant in its r-th argument, where $r = s_{l+1}$. By Proposition 4.8, there exists an occurrence β' in the $\lambda\sigma$-Böhm tree of $u[b_q \cdot \ldots \cdot b_1 \cdot \uparrow^n] =_{\lambda\sigma} b$ such that β' contains at least $h+1$ occurrences and thus the length of β' is at least h. Since the type of β and β' are the same then β' is not a leaf. In this case, the depth of b is greater than h which contradicts the hypothesis. ∎

DEFINITION 4.19 (Compact $\lambda\sigma$-term)
A $\lambda\sigma$-term $t = \lambda_{C_1} \ldots \lambda_{C_q}.u$ (u atomic) is *compact* w.r.t. an interpolation problem Φ if no de Bruijn index $\underline{j}$ with $1 \leq j \leq q$ appears free in a path of the $\lambda\sigma$-Böhm tree of u more than $h+1$ times, where h is the maximum depth in the $\lambda\sigma$-Böhm tree of the right-hand side of the equations of Φ.

PROPOSITION 4.20
Let Φ be an interpolation problem, $\widehat{\sigma}$ be an accessible solution to Φ, h be the maximum depth in the $\lambda\sigma$-Böhm tree of the right-hand side of the equations of Φ and $t = X\widehat{\sigma} = \lambda_{C_1} \dots \lambda_{C_q}.u$ (u atomic). If α is an occurrence in the $\lambda\sigma$-Böhm tree of u that contains more than $h+1$ free occurrences of the de Bruijn index $\underline{j}$ ($1 \leq j \leq q$) in its path, then for all equations, say $(X[a_1 \cdot \dots \cdot a_p \cdot \uparrow^n]\ b_1 \dots b_q) \ll^?_{\lambda\sigma} b$ of Φ where the $(h+2)$-th occurrence labeled with $\underline{j}$ is accessible w.r.t. this equation, the term b_{q-j+1} is a projection, i.e., there exists an integer $1 \leq r \leq p$ such that $b_{q-j+1} = \lambda_{B_1} \dots \lambda_{B_p}.\underline{r}$.

PROOF. Suppose that the $(h+2)$-th free occurrence of $\underline{j}$ in the path of α is accessible w.r.t. the equation $(X[a_1 \cdot \dots \cdot a_p \cdot \uparrow^n]\ b_1 \dots b_q)$. In this case, b_{q-j+1} is relevant in its r-th argument. If the head of b_{q-j+1} is not atomic then $|b_{q-j+1}| \neq 0$ and, by Proposition 4.8, there exists an occurrence α' as long as the occurrence α in the $\lambda\sigma$-Böhm tree of $u[b_q \cdot \dots \cdot b_1 \cdot a_1 \cdot \dots \cdot a_p \cdot \uparrow^n]$, where $a_1, \dots, a_p, b_1, \dots, b_q$ are $\lambda\sigma$-terms of at most second order and hence the depth of the $\lambda\sigma$-Böhm tree of $u[b_q \cdot \dots \cdot b_1 \cdot a_1 \cdot \dots \cdot a_p \cdot \uparrow^n]$, which reduces to b, is at least $h+1$ which is contradictory. Therefore, b_{q-j+1} must be a $\lambda\sigma$-term with depth equals to 0 and hence $b_{q-j+1} = \lambda_{B_1} \dots \lambda_{B_p}.\underline{r}$. ∎

DEFINITION 4.21 (Compact Accessible Solution built from an Accessible Solution)
Let Φ be an interpolation problem, $\widehat{\sigma}$ be an accessible solution to this problem and h be the maximum depth in the $\lambda\sigma$-Böhm tree of the right-hand side of the equations of Φ. The grafting $\widehat{\sigma}$ is a *compact accessible solution built from an accessible solution* to Φ if, for all meta-variable X occurring in Φ, the term $t = X\widehat{\sigma} = \lambda_{B_1} \dots \lambda_{B_q}.u$ (u atomic) is such that there is no path in the $\lambda\sigma$-Böhm tree of u containing more than $h+1$ occurrences labelled with the de Bruijn index $\underline{j}$ ($1 \leq j \leq q$). If there exists a path in the $\lambda\sigma$-Böhm tree of u that has more than $h+1$ free occurrences of the de Bruijn index $\underline{j}$ ($1 \leq j \leq q$) then the compact accessible solution is built as follows: By Proposition 4.20, if these occurrences are accessible w.r.t. the equation $(X[a_1 \cdot \dots \cdot a_p \cdot \uparrow^n]\ b_1 \dots b_q)$, we have that b_{q-j+1} is of the form $\lambda_{C_1} \dots \lambda_{C_s}.\underline{r}$. In this case, we replace all these occurrences of $\underline{j}$ by $\lambda_{C_1} \dots \lambda_{C_s}.\underline{r}$. The compact accessible solution built from the accessible solution σ is denoted by σ'.

PROPOSITION 4.22
Let Φ be an interpolation problem, σ a solution to Φ, $\widehat{\sigma}$ be the accessible solution built from σ and σ' be the compact accessible solution built from $\widehat{\sigma}$. Then σ' is also a solution to Φ.

PROOF. The proof is similar to that of Proposition 4.17. Note that in the construction of the compact accessible solution, the replacement of $\underline{j}$ for b_{q-j+1} occurs during the normalisation step of $((\lambda_{B_1} \dots \lambda_{B_q}.u)[a_1 \cdot \dots \cdot a_p \cdot \uparrow^n]\ b_1 \dots b_q)$, where $\underline{j}$ is free in u. ∎

PROPOSITION 4.23
Let Φ be an interpolation problem, σ be a solution to Φ, $\widehat{\sigma}$ be the accessible solution built from σ and σ' be the compact accessible solution built from $\widehat{\sigma}$. If h is the maximum depth in the $\lambda\sigma$-Böhm tree of the right-hand side of the equations of Φ, then for every meta-variable X of arity q, the depth of the $\lambda\sigma$-Böhm tree of $X\sigma' = \lambda_{B_1} \dots \lambda_{B_q}.u'$ is less than or equal to $(q+1)(h+1)-1$.

PROOF. By Def. 4.21 and Proposition 4.18, we know that a path in the $\lambda\sigma$-Böhm tree of u' may have at most $h+1$ occurrences of a free de Bruijn index $\underline{j}$. Therefore, since X has arity q, a path in the $\lambda\sigma$-Böhm tree of u' may have at most $(q+1)(h+1)$ symbols, i.e., its depth is at most $(q+1)(h+1)-1$. ∎

Corollary 4.24
Let Φ be a third-order interpolation problem, σ be a solution to Φ, $\widehat{\sigma}$ be the accessible solution built from σ and σ' be the compact accessible solution built from $\widehat{\sigma}$. If h is the maximum depth in the $\lambda\sigma$-Böhm tree of the right-hand side of the equations of Φ, then for every meta-variable X of arity q, the depth of the $\lambda\sigma$-Böhm tree of $X\sigma' = \lambda_{B_1} \ldots \lambda_{B_q}.u'$ is less than or equal to $(q+1)(h+1)-1$.

Proposition 4.11 allows us to use the boundary of the solution of the interpolation problem in Corollary 4.24 for third-order matching problems. In this way we can build a decision procedure as follows:

Theorem 4.25 (Decision procedure)
The class of third-order matching problems in the $\lambda\sigma$-calculus is decidable.

Proof. Let Ψ be a third-order matching problem in the $\lambda\sigma$-calculus. Enumerate all ground substitutions for the meta-variables occurring in the equations of the form $(X[a_1 \cdot \ldots \cdot a_p\cdot \uparrow^n]\ b_1 \ldots b_q) \ll^?_{\lambda\sigma} b$ of Ψ, such that the terms to be substituted for X have depth less than or equal to $(q+1)(h+1)-1$, where h is the depth of the $\lambda\sigma$-Böhm tree of b. If none of these substitutions is a solution Φ then Φ is not solvable. Otherwise, it is solvable. ∎

5 Conclusions and Future Work

We studied second- and third-order matching problems in the $\lambda\sigma$-calculus of explicit substitutions. Both matching and explicit substitutions are important tools in the implementation of higher-order deductive and computational environments. Studies like the one carried out in this paper are relevant to improve the design and implementation of such environments based on the λ-calculus and, more precisely, on variants (e.g., the $\lambda\sigma$-calculus) of the λ-calculus which are closer to implementations.

Our main contributions are the development of an algorithm for resolving second-order matching problems in the language of the $\lambda\sigma$-calculus and the proof of the decidability of the third-order matching problem in the whole language of this calculus of explicit substitutions. For achieving this we did the following:

- We characterised an important subclass of $\lambda\sigma$-terms that includes all the flexible terms that may appear in second-order matching equations.
- This subclass allowed us to develop a practical algorithm for all second-order matching problems that originate in the simply typed λ-calculus. This algorithm was obtained by specialising the unification procedure introduced in [DHK00] for treating problems expressed in the language of this subclass. The adaptation of the procedure in [DHK00] is non-trivial because, being a semi-decision procedure for higher-order unification, termination is not guaranteed as we illustrate for a second-order matching problem at the beginning of Section 3.
- For proving the decidability of third-order matching in the language of the $\lambda\sigma$-calculus, we adapted the proof of decidability of third-order matching in the λ-calculus presented in [Dow94]. In this way, initially we introduced the notion of

finite $\lambda\sigma$-Böhm tree. This notion helped us to prove that the largest depth of the $\lambda\sigma$-Böhm tree of the terms occurring in the right hand side of the matching equations bounds the size of at least one admissible matching solution.

- This boundary cannot be established directly, and so firstly $\lambda\sigma$-third-order matching problems are translated into equivalent interpolation problems (which are simple because each interpolation equation contains a unique meta-variable occurrence). Secondly, the decidability of third-order interpolation problems is obtained by proving that whenever an interpolation problem is solvable one can find a solution that is bounded by a number that depends only on the initial problem: in fact, it depends on the largest depth of the $\lambda\sigma$-Böhm tree of the terms occurring in the right hand side of the equations and on the arity of the meta-variables occurring in the problem.
- Finally, the decidability of third-order interpolation problems is naturally extended to third-order matching problems because the conversion from matching to interpolation problems preserves solutions.

Future work includes an explicit and efficient presentation of the third-order matching algorithm for the $\lambda\sigma$-calculus related to our proof of decidability as well as the analysis and comparison of second- and third-order matching solutions in different styles of explicit substitutions.

References

[ACCL91] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit Substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[ARdMK05] M. Ayala-Rincón, F.L.C. de Moura, and F. Kamareddine. Comparing and implementing calculi of explicit substitutions with eta-reduction. *APAL*, 134:5–41, 2005.

[Bor95] P. Borovanský. Implementation of Higher-Order Unification Based on Calculus of Explicit Substitutions. In M. Bartošek, J. Staudek, and J. Wiedermann editors, *Proc. of the SOFSEM'95: Theory and Practice of Informatics*, vol. 1012 *LNCS*, p. 363–368. Springer Verlag, 1995.

[Bur89] H.J. Burckert. Matching: a special case of unification? *J. of Symb. Comp.*, 8:523–536, 1989.

[dB72] N.G. de Bruijn. Lambda-Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem. *Indag. Mat.*, 34(5):381–392, 1972.

[Hue75] G. Huet. A Unification Algorithm for Typed λ-Calculus. *TCS*, 1:27–57, 1975.

[Loa03] R. Loader. Higher order β matching is undecidable. *Logic Journal of the Interest Group in Pure and Applied Logics*, 11(1):51–68, 2003.

[MKA05] F.L.C. de Moura, F. Kamareddine, and M. Ayala-Rincón. Second order matching via explicit substitutions. In F. Baader and A. Voronkov, editors, *11th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'04)*, volume 3452 of *LNAI*, pages 433–448. Springer-Verlag, 2005.

[Dow94] G. Dowek. Third order matching is decidable. *APAL*, 69:135–155, 1994.

[DHK00] G. Dowek, T. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions. *Information and Computation*, 157:183–235, 2000.

[Nip93] T. Nipkow. Functional unification of higher-order patterns. In *Proc. 8th IEEE Symp. Logic in Computer Science*, pages 64–74, 1993.

[Pad00] V. Padovani. Decidability of fourth-order matching. *Mathematical Structures in Computer Science*, 10(3):361–372, 2000.

[Rio93] A. Ríos. *Contributions à l'étude de λ-calculs avec des substitutions explicites.* Thèse de doctorat, Université Paris VII, 1993.

Received October 15, 2006