Third Order Matching is Decidable

Gilles Dowek
INRIA-Rocquencourt,
B.P. 105, 78153 Le Chesnay Cedex, France
Gilles.Dowek@inria.fr

Abstract

The higher order matching *problem is the problem of determining whether a term is an instance of another in the simply typed λ-calculus, i.e. to solve the equation $a = b$ where a and b are simply typed λ-terms and b is ground. The decidability of this problem is still open. We prove the decidability of the particular case in which the variables occurring in the problem are at most third order.*

Introduction

The *higher order matching* problem is the problem of determining whether a term is an instance of another in the simply typed λ-calculus i.e. to solve the equation $a = b$ where a and b are simply typed λ-terms and b is ground.

Pattern matching algorithms are used to check if a proposition can be deduced from another by elimination of universal quantifiers or by introduction of existential quantifiers. In automated theorem proving, elimination of universal quantifiers and introduction of existential quantifiers are mixed and full unification is required, but in proof-checking and semi-automated theorem proving, these rules can be applied separately and thus pattern matching can be used instead of unification.

Higher order matching is conjectured decidable in [6] and the problem is still open. In [5] [6] [7] Huet has given a semi-decision algorithm and shown that in the particular case in which the variables occurring in the term a are at most second order this algorithm terminates, and thus that second order matching is decidable. In [10] Statman has reduced the conjecture to the λ-definability conjecture and in [11] Wolfram has given an always terminating algorithm whose completeness is conjectured.

We prove in this paper that third order matching is decidable i.e. we give an algorithm that decides if a matching problem, in which all the variables are at most third order, has a solution. The main idea is that if the problem $a = b$ has a solution then it also has a solution whose depth is bounded by some integer s depending only on the problem $a = b$, so a simple enumeration of the substitutions whose depth is bounded by s gives a decision algorithm. This result can also be used to bound the depth of the search tree in Huet's semi-decision algorithm and thus turn it into a always-terminating decision algorithm. It can also be used to design an algorithm which enumerates a complete set of solutions to a third order matching problem and either terminates if the problem has a finite complete set of solutions or keeps enumerating solutions forever if it the problem admits no such set. At last we discuss the problems that occur when we try to generalize the proof given here to higher order matching.

1 Trees and Terms

1.1 Trees

Definitions 1 (Following [3]) An *occurrence* is a list of strictly positive integers $\alpha =< s_1, ..., s_n >$. The number n is called the *length* of the occurrence α. A *tree domain* D is a non empty finite set of occurrences such that if $\alpha < n >\in D$ then $\alpha \in D$ and if also $n \neq 1$ then $\alpha < n-1 >\in D$. A *tree* is a function from a tree domain D to a set L, called the set of labels of the tree.

If T is a tree and D its domain, the occurrence $< \ >$ is called the *root* of T and the occurrence $\alpha < n >$ 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 $\alpha < n >\in D$. A *leaf* is an occurrence that has no sons.

Let T be a tree and let $\alpha =< s_1, ..., s_n >$ be an occurrence in this tree, the *path* of α is the set of occurrences $\{< s_1, ..., s_p > \ | \ p \leq n\}$. The number of elements of this path is the length of α plus one.

The *depth* of the tree T is the length of the longest occurrence in D. This occurrence is, of course, a leaf.

If T is a tree of domain D and α is an occurrence of D, the *subtree* T/α is the tree T' whose domain is $D' = \{\beta \mid \alpha\beta \in D\}$ and such that

$$T'(\beta) = T(\alpha\beta)$$

By an abuse of language, if $\alpha < n >$ is an occurrence of a tree T, the subtree $T/\alpha < n >$ is also called the n^{th} *son* of the occurrence α.

If a is a label and $T_1, ..., T_n$ are trees (of domains $D_1, ..., D_n$) then the *tree of root a and sons* $T_1, ..., T_n$ is the tree T of domain $D = \{< \ >\} \cup \bigcup_i \{< i > \alpha \mid \alpha \in D_i\}$ such that

$$T(< \ >) = a$$

and

$$T(< i > \alpha) = T_i(\alpha)$$

If T is a tree of domain D, α an occurrence of D and T' a tree of domain D' then the *graft* of T' in T at the occurrence α ($T[\alpha \leftarrow T']$) is the tree T'' of domain $D'' = D - \{\alpha\beta \mid \alpha\beta \in D\} \cup \{\alpha\beta \mid \beta \in D'\}$ and such that

$$T''(\gamma) = T'(\beta) \text{ if } \gamma = \alpha\beta$$

and

$$T''(\gamma) = T(\gamma) \text{ otherwise}$$

Let T and T' be trees and let a be a label such that all the occurrences of a in T are leaves $\alpha_1, ..., \alpha_n$ then the *substitution* of T' for a in T ($T[a \leftarrow T']$) is defined as $T[\alpha_1 \leftarrow T']...[\alpha_n \leftarrow T']$. Note that since $\alpha_1, ..., \alpha_n$ are leaves, the order in which the grafts are performed is insignificant.

1.2 Types

Definition 2 (Type)

Let us consider a finite set $\mathcal{T}$. The elements of $\mathcal{T}$ are called *atomic types*. A *type* is a tree whose labels are either the elements of $\mathcal{T}$ or $\rightarrow$ and such that the occurrences labeled with an element of $\mathcal{T}$ are leaves and the ones labeled with $\rightarrow$ have two sons.

Let T be a type, if the root of T is labeled with an atomic type U then T is written U, if the root of T is labeled with $\rightarrow$ and its sons are written T_1 and T_2 then T is written $(T_1 \rightarrow T_2)$. By convention $T_1 \rightarrow T_2 \rightarrow T_3$ is an abbreviation for $(T_1 \rightarrow (T_2 \rightarrow T_3))$.

Definition 3 (Order of a Type)

If T is a type, the *order* of T is defined by

- $o(T) = 1$ if T is atomic,
- $o(T_1 \rightarrow T_2) = max\{1 + o(T_1), o(T_2)\}$.

1.3 Typed λ-terms

Definitions 4 For each type T we consider three sets $\mathcal{C}_T$, $\mathcal{I}_T$, $\mathcal{L}_T$. The elements of $\mathcal{C}_T$ are called *constants* of type T, those of $\mathcal{I}_T$ *instantiable variables* of type T and those of $\mathcal{L}_T$ *local variables* of type T.

We assume that we have in each atomic type at least a constant and that there is a finite number of constants i.e. that the set $\bigcup_T \mathcal{C}_T$ is finite [1]. We assume also that we have an infinite number of instantiable and local variables of each type.

A typed λ-term is a tree whose labels are either *App*, or $< Lam, x >$ where x is a local variable, or $< Var, x >$ where x is a constant, an instantiable variable or a local variable such that the occurrences labeled with *App* have two sons, the occurrences labeled with $< Lam, x >$ have one son and the occurrences labeled with $< Var, x >$ are leaves.

Let t be a term, if the root of t is labeled with $< Var, x >$ we write it x, if the root of t is labeled with $< Lam, x >$ and its son is written u then we write it $\lambda x : T.u$ where T is the type of x, if the root of t is labeled with *App* and its sons are written u and v then we write it $(u\ v)$. By convention $(u\ v\ w)$ is an abbreviation for $((u\ v)\ w)$.

In a term t, an occurrence α labeled with $< Var, x >$ is *bound* if there exists an occurrence β in the path of α labeled with $< Lam, x >$, it is *free* otherwise.

A term is *ground* if no occurrence is labeled with a pair $< Var, x >$ with x instantiable.

Let t and t' be terms and x be a variable, the *substitution* of t' for x in t ($t[x \leftarrow t']$) is defined as $t[< Var, x > \leftarrow t']$.

Definition 5 (Type of a Term)

A term t is said to have the type T if either:

- t is a constant, an instantiable variable or a local variable of type T.
- $t = (u\ v)$ and u has type $U \rightarrow T$ and v type U for some type U,
- $t = \lambda x : U.u$, the term u has type V and $T = U \rightarrow V$.

A term t is said to be *well-typed* if there exists a type T such that t has type T. In this case T is unique and is called *the type of* t.

[1] This technical restriction is in fact superfluous, because a matching problem expressed in a language with an infinite number of constants can always be reduced to one expressed in the language with a finite number of constants obtained by considering only the constants occurring in the problem and one constant in each atomic type.

Definition 6 ($\beta\eta$-reduction)

The $\beta\eta$-reduction relation, written $\triangleright$, is defined as the smallest transitive relation compatible with term structure such that

$$(\lambda x : T.t\ u) \triangleright t[x \leftarrow u]$$

$$\lambda x : T.(t\ x) \triangleright t \quad \text{if } x \text{ is not free in } t$$

We adopt the usual convention of considering terms up to α-conversion (i.e. bound variable renaming) and we consider that bound variables are renamed to avoid capture during substitutions. A rigorous presentation would use, for instance, de Bruijn indices [2].

Obviously, if t is a term of type T, x is a variable of type U and u a term of type U then the term $t[x \leftarrow u]$ has type T. In the same way if a term t has type T and t reduces to u then u has type T.

Proposition 1 The $\beta\eta$-reduction relation is strongly normalizable and confluent on typed terms, and thus each term has a unique normal form.
Proof See, for instance, [4].

Proposition 2 Let t be a normal well-typed term of type $U_1 \to \ldots \to U_n \to U$ (U atomic), the term t has the form

$$t = \lambda y_1 : U_1. \ldots \lambda y_m : U_m.(x\ u_1\ \ldots\ u_p)$$

where $m \leq n$ and x is a constant, an instantiable variable or a local variable.
Proof The term t can be written in a unique way $t = \lambda y_1 : V_1. \ldots \lambda y_m : V_m.u$ where u is not an abstraction. The term u can be written in a unique way $u = (v\ u_1\ \ldots\ u_p)$ where v is not an application. The term v is not an application by definition, it is not an abstraction (if $p = 0$ because u is not an abstraction and if $p \neq 0$ because t is normal), it is therefore a constant, an instantiable variable or a local variable. Then since t has type $U_1 \to \ldots \to U_n \to U$, we have $m \leq n$ and for all i, $V_i = U_i$.

Definition 7 (Head of a Term, Atomic Term)

Let $t = \lambda y_1 : T_1. \ldots \lambda y_m : T_m.(x\ u_1\ \ldots\ u_p)$ be a normal term. The symbol x is called the *head* of the term. If $m = 0$ then t is said to be *atomic*, it is an abstraction otherwise.

Definition 8 (η-long Form)

If $t = \lambda y_1 : U_1. \ldots \lambda y_m : U_m.(x\ u_1\ \ldots\ u_p)$ is a term of type $T = U_1 \to \ldots \to U_n \to U$ (U atomic) ($m \leq n$) which is in $\beta\eta$-normal form then we define its *β-normal η-long* form as the term

$$t' = \lambda y_1 : U_1. \ldots \lambda y_m : U_m.\lambda y_{m+1} : U_{m+1}. \ldots \lambda y_n : U_n.(x\ u'_1\ \ldots\ u'_p\ y'_{m+1}\ \ldots\ y'_n)$$

where u'_i is the β-normal η-long form of u_i and y'_i is the β-normal η-long form of y_i.

This definition is by induction on the pair $< c_1, c_2 >$ where c_1 is the number of occurrences in t and c_2 the number of occurrences in T

In the following all the terms are assumed to be in β-normal η-long form.

1.4 Böhm Trees

Definition 9 (Böhm Tree)

A (finite) *Böhm tree* is a tree whose occurrences are labeled with pairs $< l, x >$ such that l is a list of local variables $< y_1, ..., y_n >$ and x is a constant, an instantiable variable or a local variable.

Definition 10 (Type of a Böhm Tree)

Let t be a Böhm tree whose root is labeled with the pair $<< y_1, ..., y_n >, x >$ and whose sons are $u_1, ..., u_p$. The Böhm tree t is said to have the type T if the Böhm trees $u_1, ..., u_p$ have type $U_1, ..., U_p$ the symbol x has type $U_1 \rightarrow ... \rightarrow U_p \rightarrow U$ (U atomic) and $T = T_1 \rightarrow ... \rightarrow T_n \rightarrow U$ where $T_1, ..., T_n$ are the types of the variables $y_1, ..., y_n$.

A Böhm tree t is said to be *well-typed* if there exists a type T such that t has type T. In this case T is unique and is called *the type of* t.

Definition 11 (Böhm Tree of a Normal Term)

Let $t = \lambda y_1 : T_1. \ ... \ \lambda y_n : T_n.(x \ u_1 \ ... \ u_p)$ be a λ-term in normal (η-long) form. The *Böhm tree* of t is inductively defined as the tree whose root is the pair $< l, x >$ where $l =< y_1, ..., y_n >$ is the list of the variables bound at the top of this term, x is the head symbol of t and sons are the Böhm trees of $u_1, ..., u_p$.

Remark Normal (η-long) well-typed terms and well-typed Böhm trees are in one-to-one correspondence. Moreover if t is a normal (η-long) term and $\tilde{t}$ is its Böhm tree then occurrences in t labeled with a constant, an instantiable variable or a local variable and occurrences in $\tilde{t}$ are in one-to-one correspondence. So we will use the following abuse of notation: if α is an occurrence in the Böhm tree of t we write (t/α) for the normal (η-long) term corresponding to the Böhm tree $(\tilde{t}/\alpha)$ and $t[\alpha \leftarrow u]$ for the term $t[\alpha' \leftarrow u]$ where α' is the occurrence of a variable or a constant in t corresponding to α.

Notation Let t be a term, we write $|t|$ for the depth of the Böhm tree of the normal (η-long) form of t.

Proposition 3 In each type T there is a ground term t such that $|t| = 0$.
Proof Let $T = U_1 \rightarrow ... \rightarrow U_n \rightarrow U$ with U atomic and let c be a constant of type U. The term $t = \lambda x_1 : U_1. \ ... \ \lambda x_n : U_n.c$ has type T and $|t| = 0$.

1.5 Substitution

Definition 12 (Substitution)

A *substitution* is a finite set of pairs $< x_i, t_i >$ where x_i is an instantiable variable and t_i a term of the same type in which no local variable occurs free such that if $< x, t >$ and $< x, t' >$ are both in this set then $t = t'$. The variables x_i are said to be *bound* by the substitution.

Definition 13 (Substitution applied to a Term)

If σ is a substitution and t a term then we let

$$\sigma t = t[\alpha_1^1 \leftarrow t_1]...[\alpha_1^{p_1} \leftarrow t_1]...[\alpha_n^1 \leftarrow t_n]...[\alpha_n^{p_n} \leftarrow t_n]$$

where $\alpha_i^1, ..., \alpha_i^{p_i}$ are the occurrences of x_i in t.

Note that since the α_i^j are leaves, the order in which the grafts are performed is insignificant.

Definition 14 (Composition of Substitutions)
Let σ and τ be two substitutions the substitution $\tau \circ \sigma$ is defined by

$$\tau \circ \sigma = \{< x, \tau t > \mid < x, t > \in \sigma\} \cup \{< x, t > \mid < x, t > \in \tau \text{ and } x \text{ not bound by } \sigma\}$$

Proposition 4 Let σ and τ be two substitutions and t is a term, we have

$$(\tau \circ \sigma)t = \tau(\sigma t)$$

Proof By decreasing induction on the depth of an occurrence α in t we prove that we have

$$(\tau \circ \sigma)(t/\alpha) = \tau(\sigma(t/\alpha))$$

2 Pattern Matching

Definition 15 (Matching Problem)
A *matching problem* is a set $\Phi = \{< a_1, b_1 >, ..., < a_n, b_n >\}$ of pairs of terms of the same type such that the terms $b_1, ..., b_n$ are ground. A pair $< a, b >$ is frequently written as an equation $a = b$.

Definition 16 (Third Order Matching Problem)
A *third order matching problem* is a matching problem $\Phi = \{a_1 = b_1, ..., a_n = b_n\}$ such that the types of the instantiable variables that occur in $a_1, ..., a_n$ are of order at most three.

Definition 17 (Solution)
Let $\Phi = \{a_1 = b_1, ..., a_n = b_n\}$ be a matching problem. A substitution σ is a *solution* of this problem if and only if for every i, the normal form of the terms σa_i and b_i are identical up to α-conversion.

Remark Usual unification terminology distinguishes *variables* (here instantiable variables) and *constants*. The need for local variables comes from the fact that we want to transform the problem $\lambda y : T.x = \lambda y : T.y$ (where x is an instantiable variable of type T) into the problem $x = y$ by dropping the common abstraction. The symbol y cannot be an instantiable variable (because it cannot be instantiated by a substitution), it cannot be a constant because, if it were, we would have the solution $x \leftarrow y$ to the second problem which is not a solution to the first. So we let y be a local variable and the solution $x \leftarrow y$ is now forbidden in both problems because no local variable can occur free in the terms substituted for variables in a substitution.

In Huet's unification algorithm [5] [6] these local variables are always kept in the head of the terms in common abstractions. In Miller's mixed prefixes terminology [8], constants are universal variables declared to the left hand side of the instantiable variables and local variables are universal variables declared to the right hand side of all the instantiable variables.

Remark In an alternative definition of matching problems, the terms $b_1, ..., b_n$ do not need to be ground. The method of this paper can be adapted to such problems using the standard technique of *variable freezing* [6].

Definition 18 (Ground Solution)
Let $\Phi = \{a_1 = b_1, ..., a_n = b_n\}$ be a problem and let σ be a solution to Φ. The solution σ is said to be *ground* if for each instantiable variable that has an occurrence in some a_i, the term σx is ground.

Proposition 5 If a matching problem has a solution then it has a ground solution.
Proof Let $\Phi = \{a_1 = b_1, ..., a_n = b_n\}$ be a matching problem and let σ be a solution to this problem. Let $y_1 : T_1, ..., y_n : T_n$ be the instantiable variables occurring in the term σx for some x instantiable variable occurring in some a_i. Let $u_1, ..., u_n$ be ground terms of the types $T_1, ..., T_n$. Let $\tau = \{< y_1, u_1 >, ..., < y_n, u_n >\}$, and $\sigma' = \tau \circ \sigma$. Obviously, for each instantiable variable x of a, the term $\sigma' x$ is ground and σ' is a solution to Φ.

Definition 19 (Complete Set of Solutions)
Obviously if σ is a solution to a problem Φ then for any substitution τ, $\tau \circ \sigma$ is also a solution to Φ. A set S of solutions to a problem Φ is said to be *complete* if for every substitution θ that is a solution to this problem there exists a substitution $\sigma \in S$ and a substitution τ such that $\theta = \tau \circ \sigma$.

Lemma 1 Some problems have no finite complete set of solutions.
Proof (Example 1) Consider an atomic type T and an instantiable variable $x : T \to (T \to T) \to T$ and the problem

$$\lambda a : T.(x\ a\ \lambda z : T.z) = \lambda a : T.a$$

The substitutions

$$x \leftarrow \lambda o : T.\lambda s : T \to T.(s\ ...\ (s\ o)\ ...\ )$$

are solutions to this problem and they cannot be obtained as instances of a finite number of solutions.

Remark In [6] [12], the similar examples $(x\ \lambda z : T.z) = a$ and $(x\ \lambda z : T.z) = b(a)$ are considered.

So in contrast with second order matching [6] [7] there is no (always terminating) algorithm that enumerates a complete set of solutions to a third order matching problem.

We consider now algorithms that take as an input a matching problem and either give *one* solution to the problem or fail if it does not have any.

3 A Bound on the Depth of Solutions

All the problems considered in the rest of the paper are third order.

To prove the decidability of third order matching we are going to prove that the depth of the term t substituted to a variable x by a solution σ to a problem Φ can be bounded by an integer s depending only on the problem Φ. Of course the previous example shows that a matching problem may have solutions of arbitrary depth, but to design a decision algorithm we do not need to prove that *all* the solutions are bounded by s but only that *at least one* is. To show this result we take a problem Φ that has a solution σ (by proposition 5, we can consider without loss of generality that this solution is ground) and we build another solution σ' whose depth is bounded by an integer s depending only on the problem Φ.

The proof is divided into two parts. In the first part,we focus on a particular case in which the problem Φ is a an *interpolation problem* i.e. set of equations of the form $(x\ c_1\ ...\ c_n) = b$ such that x is an instantiable variable and $c_1, ..., c_n$ and b are ground terms. Then, in the second part, we reduce the general case to this particular case.

Consider now an equation $(x\ c_1\ ...\ c_n) = b$ and a substitution σ solution to this equation. Let us write $t = \sigma x = \lambda y_1 : T_1.\ ...\ \lambda y_n : T_n.u$ (u atomic). We have

$$\sigma(x\ c_1\ ...\ c_n) = (\lambda y_1 : T_1.\ ...\ \lambda y_n : T_n.u\ c_1\ ...\ c_n)$$

This term reduces to $u[y_1 \leftarrow c_1, ..., y_n \leftarrow c_n]$ whose normal form is b.

The terms c_i are at most second order. In the key lemma, we prove that, *in the general case*, when we substitute a second order term c to a variable y in a term u and we normalize the term $u[y \leftarrow c]$, we get a term with a depth larger than or equal to the one of u. If this were true in all the cases, we would know that the depth of t (the solution) has to be less than or equal to the depth of b (the right hand side of the equation). A simple enumeration of the terms t whose depth is less than or equal to $|b|$ would give a decision procedure.

Actually, the key lemma shows that the depth of the normal form of $u[y \leftarrow c]$ can be less than the depth of u in two cases : when c is a non relevant term and when $|c| = 0$. When such cases happen, solutions may have an arbitrary depth. In these cases, we show that if the problem Φ has a solution σ then it has also another solution σ' whose depth is bounded by some integer s depending only on the problem Φ.

3.1 Interpolation Problems

Definition 20 (Interpolation Problem)

An *interpolation problem* is a set of equations of the form $(x\ c_1\ ...\ c_n) = b$ such that x is an instantiable variable and $c_1, ..., c_n$ and b are ground terms.

3.1.1 Key Lemma

Definition 21 (Relevant Term)

Let $c = \lambda z_1 : U_1.\ ...\ \lambda z_p : U_p.d$ (d atomic) be a normal term and i an integer, $i \leq p$. We say that c is *relevant* in its i^{th} argument if z_i has an occurrence in the term d.

Lemma 2 (Key Lemma) Let us consider a normal term u, a variable y of type T of order at most two and a normal ground term c of type T.

(1) If y has an occurrence in u then $|c| \leq |u[y \leftarrow c]|$.

(2) If α is an occurrence in the Böhm tree of u such that no occurrence in the path of α is labeled with y, then α is also an occurrence in the normal form of $u[y \leftarrow c]$ and has the same label in the Böhm tree of u and in the Böhm tree of the normal form of $u[y \leftarrow c]$.

(3) If $\alpha =< s_1, ..., s_n >$ is an occurrence in the Böhm tree of u such that for each occurrence $\beta =< s_1, ..., s_k >$ in the path of α, $\beta \neq \alpha$, labeled with y, the term c is relevant in its r^{th} argument where r is the position of the son of β in the path of α i.e. $r = s_{k+1}$, then there exists an occurrence α' of the Böhm tree of the normal form of $u[y \leftarrow c]$ such that all the labels occurring in the path of α, except y, occur in the path of α' and the number of times they occur in the path of α' is greater than or equal to the number of times they occur in the path of α. Moreover if the occurrence α is labeled with a symbol different from y, then the occurrence α' is labeled with this same symbol.

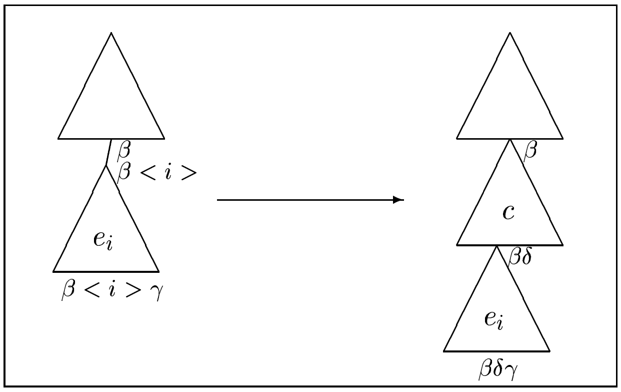
(4) Moreover if $|c| \neq 0$ then the length of α' is greater than or equal to the length of α.

Proof By induction on the number of occurrences of y in u. We substitute these occurrences one by one and we normalize the term. Let β be the occurrence in the Böhm tree of u corresponding to the occurrence of y in u we substitute. Let us write

$$c = \lambda z_1 : U_1. \ ... \ \lambda z_p : U_p.d$$

The term (u/β) has the form $\lambda v_1 : V_1. \ ... \ \lambda v_q : V_q.(y \ e_1 \ ... \ e_p)$. When we substitute y by the term c in $(y \ e_1 \ ... \ e_p)$ we get $(c \ e_1 \ ... \ e_p)$ and when we normalize this term we get the term $d[z_1 \leftarrow e_1, ..., z_p \leftarrow e_p]$ which is normal because the type of the e_i are first order.

Let us consider the occurrences in the Böhm tree of u, while substituting the occurrence of y corresponding to β, we have removed all the occurrences $\beta < i > \gamma$ where i is an integer $(i \leq p)$ and γ is an occurrence in the Böhm tree of e_i. We have added all the occurrences $\beta\delta$ where δ is an occurrence of the Böhm tree of c labeled with a symbol different from $z_1, ..., z_p$ and all the occurrences $\beta\delta\gamma$ where δ is a leaf occurrence in the Böhm tree of c labeled with a z_i and γ is an occurrence of the Böhm tree of e_i.



(1) Let β be an outermost occurrence of y in the Böhm tree of u. For each occurrence δ in the Böhm tree of c, $\beta\delta$ is an occurrence in the Böhm tree of the normal form of $u[y \leftarrow c]$. So $|c| \leq |u[y \leftarrow c]|$.

(2) When an occurrence β of y is substituted by c all the occurrences removed have the form $\beta < i > \gamma$. So if no occurrence in the path of α is labeled with y, the occurrence α remains in the normal form of $u[y \leftarrow c]$.

(3) If the occurrence β is not in the path of α then the occurrence α is still an occurrence in the normal form of $u[y \leftarrow c]$, we take $\alpha' = \alpha$.

If $\beta = \alpha$ then the occurrence β is an occurrence of the Böhm tree of the normal form of $u[y \leftarrow c]$. We take $\alpha' = \beta = \alpha$.

If β is in the path of α and $\beta \neq \alpha$, $\beta =< s_1, ..., s_k >$ then let r be the position of the son of β in the path of α i.e. $r = s_{k+1}$. Let γ be such that $\alpha = \beta < r > \gamma$. By hypothesis z_r has an occurrence in d, let δ be such an occurrence. The occurrence $\beta\delta\gamma$ is an occurrence in the Böhm tree of the normal form of $u[y \leftarrow c]$. We take $\alpha' = \beta\delta\gamma$.

In all the cases, all the labels occurring in the path of α, except y, occur in the path of α' and the number of times they occur in the path of α' is greater than or equal to the number of times they occur in the path of α.

If the occurrence α is labeled with a symbol different from y, then the occurrence α' is labeled with the same symbol as α.

(4) If $\delta =< \ >$ then $c = \lambda z_1 : U_1. \ ... \ \lambda z_p : U_p.z_r$ and $|c| = 0$. So if $|c| \neq 0$ then $\delta \neq < \ >$ and the length of α' is greater than or equal to the length of α.

Corollary Let us consider a normal term u, a variable y of type T of order at most two and a ground term c of type T. If c is relevant in all its arguments and $|c| \neq 0$ then $|u| \leq |u[y \leftarrow c]|$.
Proof We take for α the longest occurrence in the Böhm tree of u. When we substitute one by one the occurrences of y, by part (4) of the key lemma, we get occurrences that are at least long. So there is an occurrence in the Böhm tree of the normal form of $u[y \leftarrow c]$ which is at least long as α. So $|u| \leq |u[y \leftarrow c]|$.

3.1.2 Computing the Substitution σ'

Let us consider an equation $(x\ c_1\ ...\ c_n) = b$. Let σ be a solution to this equation and let $t = \sigma x$. Let us write $t = \lambda y_1 : T_1.\ ...\ \lambda y_n : T_n.u$. The normal form of the term $\sigma(x\ c_1\ ...\ c_n)$ is the normal form of $u[y_1 \leftarrow c_1, ..., y_n \leftarrow c_n]$. If all the c_i are relevant in their arguments and $|c_i| \neq 0$ then using the corollary of the key lemma we have $|t| \leq |(t\ c_1\ ...\ c_n)|$, so $|t| \leq |b|$ and this gives a bound on the depth of t. But the depth of t may decrease when applied to the terms c_i and normalized in two cases:
- if one of the terms c_i is not relevant in one of its arguments,
- if one of the terms c_i is such that $|c_i| = 0$.

So solutions may have an arbitrary depth. When this happens, we compute another solution to the problem whose depth is bounded by an integer s depending only on the initial problem.

This new substitution is constructed in two steps. In the first step we deal with non relevant terms and in the second with terms of depth 0.

Example 2 Let x be an instantiable variable of type $T \to (T \to T) \to T$. Consider the problem

$$(x\ a\ \lambda z : T.b) = b$$

The variable z has no occurrence in b so this problem has solutions of arbitrary depth

$$x \leftarrow \lambda o : T.\lambda s : T \to T.(s\ t)$$

where t is an arbitrary term of type T. In this example we will compute the substitution

$$x \leftarrow \lambda o : T.\lambda s : T \to T.(s\ c)$$

where c is a constant.

Example 1 (continued) The term $\lambda z : T.z$ has depth 0, so we have solutions of an arbitrary depth. In this example we will compute the substitution

$$x \leftarrow \lambda o : T.\lambda s : T \to T.(s\ o)$$

Definition 22 (Occurrence Accessible with Respect to an Equation of the Form $(x\ c_1\ ...\ c_n) = b$)
Let us consider an equation

$$(x\ c_1\ ...\ c_n) = b$$

and the term

$$t = \sigma x = \lambda y_1 : T_1.\ ...\ \lambda y_n : T_n.u$$

Let us consider the Böhm tree of t. The set of the occurrences of the Böhm tree of t *accessible* with respect to the equation $(x\ c_1\ ...\ c_n) = b$ is inductively defined as:
• the root of the Böhm tree of t is accessible,
• if α is an accessible occurrence labeled with y_i and c_i is relevant in its j^{th} argument then the occurrence $\alpha < j >$ (the j^{th} son of α) is accessible,
• if α is an accessible occurrence labeled with a symbol different from all the y_i then all the sons of α are accessible.

Definition 23 (Occurrence Accessible with Respect to an Interpolation Problem)
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 24 (Term Accessible with Respect to an Interpolation Problem)
A term is *accessible* with respect to an interpolation problem if all the occurrences of its Böhm tree which are not leaves are accessible with respect to this problem.

Definition 25 (Accessible Solution Built from a Solution)
Let Φ be an interpolation problem and let σ be a solution to this problem. For each instantiable variable x occurring in the equations of Φ we consider the term $t = \sigma x$. In the Böhm tree of t, we prune all the occurrences non accessible with respect to the equations of Φ in which x has an occurrence and put Böhm trees of ground terms of depth 0 of the expected type as leaves. The tree obtained that way is the Böhm tree of some term t'. We let $\hat{\sigma} x = t'$.

Example 2 (continued) From the solution

$$x \leftarrow \lambda o : T.\lambda s : T \rightarrow T.(s\ t)$$

where t is an arbitrary term, we compute the substitution

$$x \leftarrow \lambda o : T.\lambda s : T \rightarrow T.(s\ c)$$

where c is a constant.

Proposition 6 Let Φ be an interpolation problem and let σ be a solution to Φ, then the accessible solution $\hat{\sigma}$ built from σ is a solution to Φ.
Proof Let us consider an equation $(x\ c_1\ ...\ c_n) = b$ of Φ and the terms

$$\sigma x = t = \lambda y_1 : T_1.\ ...\ \lambda y_n : T_n.u$$

and

$$\hat{\sigma} x = t' = \lambda y_1 : T_1.\ ...\ \lambda y_n : T_n.u'$$

We prove by decreasing induction on the depth of the occurrence α of the Böhm tree of u that if α is accessible with respect to the equation $(x\ c_1\ ...\ c_n) = b$ then α is also an occurrence of the Böhm tree of u' and

$$(u'/\alpha)[y_1 \leftarrow c_1, ..., y_n \leftarrow c_n] = (u/\alpha)[y_1 \leftarrow c_1, ..., y_n \leftarrow c_n]$$

and then since the root of u is accessible with respect to this equation we have

$$u'[y_1 \leftarrow c_1, ..., y_n \leftarrow c_n] = u[y_1 \leftarrow c_1, ..., y_n \leftarrow c_n]$$

i.e.

$$((\hat{\sigma}x)\ c_1\ ...\ c_n) = b$$

So $\hat{\sigma}$ is a solution to Φ.

Proposition 7 Let Φ be an interpolation problem and let σ be a solution to Φ. Let h be the maximum depth of the right hand side of the equations of Φ. Let $\hat{\sigma}$ the accessible solution built from σ. Let

$$t = \hat{\sigma}x = \lambda y_1 : T_1.\ ...\ \lambda y_n : T_n.u$$

(u atomic). There are at most $h+1$ occurrences of symbols not in $\{y_1, ..., y_n\}$ on a path of the Böhm tree of t.

Proof Let α be an occurrence in the Böhm tree of t such that there are more than $h+1$ occurrences of symbols not in $\{y_1, ..., y_n\}$ in the path of α.

Let β be the $(h+1)-th$ occurrence of such a symbol. Since there are more that $h+1$ occurrences of symbols not in $\{y_1, ..., y_n\}$ in the path of α, the occurrence β is not a leaf, so it is accessible with respect to some equation $(x\ c_1\ ...\ c_n) = b$ of Φ. Also, since this occurrence is not a leaf, it is labeled with a symbol f whose type is not first order.

For each occurrence $\gamma =< s_1, ..., s_k >$ in the path of β labeled with y_i, let r be the position of the son of this occurrence in this path (i.e. $r = s_{k+1}$). Since the occurrence β is accessible with respect to the equation $(x\ c_1\ ...\ c_n) = b$, the term c_i is relevant in its r^{th} argument. So using n times the part (3) of the key lemma there exists an occurrence β' in the Böhm tree of the normal form of the term $b = (\hat{\sigma}x\ c_1\ ...\ c_n)$ such that the path of β' contains at least $h+1$ occurrences. Thus, the length of this occurrence is at least h. This occurrence is labeled with the symbol f whose type is not first order, so it has a son β'' whose length is at least $h+1$.

So the depth of b is greater than or equal to $h+1$ which is contradictory.

Definition 26 (Compact Term)

A term $t = \lambda y_1 : T_1.\ ...\ \lambda y_n : T_n.u$ (u atomic) is *compact* with respect to an interpolation problem Φ if no variable y_i has more than $h+1$ occurrences in a path of its Böhm tree, where h is the maximum depth of the right hand side of the equations of Φ.

Proposition 8 Let Φ be an interpolation problem and let $\hat{\sigma}$ be an accessible solution to Φ. Let h be the maximum depth of the right hand side of the equations of Φ. Let us consider an instantiable variable x and

$$t = \hat{\sigma}x = \lambda y_1 : T_1.\ ...\ \lambda y_n : T_n.u$$

(u atomic). Let us consider a variable y_i and an occurrence α of the Böhm tree of t such that there are more than $h+1$ occurrences on the path of α labeled with the variable y_i.

We consider all the equations $(x\ c_1\ ...\ c_n) = b$ of Φ such that the $(h+2)-th$ occurrence of y_i is accessible with respect to this equation. Then there exists an integer j such that for every such equation we have

$$c_i = \lambda z_1 : U_1.\ ...\ \lambda z_p : U_p.z_j$$

Proof Let β be the first occurrence of y_i in the path of α. Let j be the integer such that $\alpha = \beta < j > \beta'$.

Let $(x\ c_1\ ...\ c_n) = b$ be an equation of Φ such that the $(h+2)-th$ occurrence of y_i on the considered path is accessible with respect to this equation.

If the head of c_i is a symbol different from a z_k then $|c_i| \neq 0$. Using part (3) of the key lemma when we substitute $c_1, ..., c_{i-1},\ c_{i+1},\ ...,\ c_n$ we have an occurrence α' that has more than $h+1$ occurrences of y_i on its path. Then using part (4) of the key lemma, when we substitute c_i we have an occurrence α'' whose length is greater than or equal to $h+1$ so

$$h + 1 \leq |u[y_1 \leftarrow c_1, ..., y_n \leftarrow c_n]|$$

i.e. $h+1 \leq |b|$ which is contradictory. So we have

$$c_i = \lambda z_1 : U_1. \ ... \ \lambda z_p : U_p.z_k$$

Since $h+2 > 1$ the occurrence $\beta < j >$ is accessible with respect to the equation $(x\ c_1\ ...\ c_n) = b$. Thus as the occurrence β is labeled with y_i and the occurrence $\beta < j >$ is accessible with respect to this equation, the term c_i is relevant in its j^{th} argument. Therefore $k = j$ and

$$c_i = \lambda z_1 : U_1. \ ... \ \lambda z_p : U_p.z_j$$

Definition 27 (Compact Accessible Solution Built from an Accessible Solution)
Let Φ be an interpolation problem and let $\hat{\sigma}$ be an accessible solution to this problem. Let h be the maximum depth of a right hand side of the equations of Φ. We let

$$\hat{\sigma}x = t = \lambda y_1 : T_1. \ ... \ \lambda y_n : T_n.u$$

For each α, occurrence in t labeled with y_i such that the corresponding occurrence α' in the Böhm tree of t has more than $h+1$ occurrences labeled with y_i in its path, we have $c_i = \lambda z_1 : U_1. \ ... \ \lambda z_p : U_p.z_j$ in all the equations $(x\ c_1\ ...\ c_n) = b$ of Φ such that α' is accessible with respect to this equation. We substitute the occurrence α by the term $\lambda z_1 : U_1. \ ... \ \lambda z_p : U_p.z_j$. We get that way a term t'. We let $\sigma' x = t'$.

Example 1 (continued) We build the substitution

$$x \leftarrow \lambda o : T.\lambda s : T \rightarrow T.(s\ o)$$

Example 3 Consider an instantiable variable x of type $(T \rightarrow T \rightarrow T) \rightarrow T$. And the problem

$$(x\ \lambda y : T.\lambda z : T.y) = a$$

$$(x\ \lambda y : T.\lambda z : T.z) = b$$

We have the solution

$$x \leftarrow \lambda f : T \rightarrow T \rightarrow T.(f\ a\ (f\ c\ (f\ d\ b)))$$

This solution is accessible but not compact. The first occurrence of f is accessible with respect to both equations, but the second and third occurrences are accessible only with respect to the second one. We have $h = 0$, so we substitute the second and third occurrences of f by the term $\lambda y : T.\lambda z : T.z$ and we get the substitution

$$x \leftarrow \lambda f : T \rightarrow T \rightarrow T.(f\ a\ b)$$

Note that we must not substitute the first occurrence of f by $\lambda y : T.\lambda z : T.z$, because we would get the substitution $x \leftarrow \lambda f : T \rightarrow T \rightarrow T.b$ which is not a solution to the first equation.

Proposition 9 Let Φ be an interpolation problem and let σ be a solution to Φ. Let $\hat{\sigma}$ the accessible solution built from σ and σ' the compact accessible solution built from $\hat{\sigma}$. Then σ' is a solution to Φ.

Proof We consider an equation $(x\ c_1\ ...\ c_n) = b$ and we let

$$\hat{\sigma}x = t = \lambda y_1 : T_1.\ ...\ \lambda y_n : T_n.u$$

and

$$\sigma' x = t = \lambda y_1 : T_1.\ ...\ \lambda y_n : T_n.u'$$

The term u' is obtained by substituting in the term u some occurrences (say $\beta_1, ..., \beta_k$) by some terms (say $e_1, ..., e_k$). If α is an occurrence of u then we define u'_α as the term obtained by substituting in the term u/α the occurrence γ_i by the term e_i if $\beta_i = \alpha\gamma_i$.

We prove by decreasing induction on the depth of the occurrence α of the Böhm tree of u that if α is accessible with respect to the equation $(x\ c_1\ ...\ c_n) = b$ then

$$(u'_\alpha)[y_1 \leftarrow c_1, ..., y_n \leftarrow c_n] = (u/\alpha)[y_1 \leftarrow c_1, ..., y_n \leftarrow c_n]$$

Thus for the root we get

$$u'[y_1 \leftarrow c_1, ..., y_n \leftarrow c_n] = u[y_1 \leftarrow c_1, ..., y_n \leftarrow c_n]$$

i.e.

$$((\sigma' x)\ c_1\ ...\ c_n) = b$$

So σ' is a solution to all the equations of Φ.

Proposition 10 Let Φ be an interpolation problem and let σ be a solution to Φ. Let $\hat{\sigma}$ be the accessible solution built from σ and σ' the compact accessible solution built from $\hat{\sigma}$. Let h be the maximum depth of the right hand side of the equations of Φ. For every instantiable variable x of arity n, $\sigma' x$ has a depth less than or equal to $(n+1)(h+1) - 1$.

Proof In a path of the Böhm tree of $\sigma' x$ each y_i has at most $h+1$ occurrences and there are at most $h+1$ occurrences of other symbols, so there are at most $(n+1)(h+1)$ occurrences. Therefore the depth of $\sigma' x$ is bounded by $(n+1)(h+1) - 1$.

Lemma 3 Let Φ be a third order interpolation problem. If Φ has a solution σ then it also has a solution σ' such that for every instantiable variable x, σx has a depth less than or equal to $(n+1)(h+1) - 1$, where h is maximum of the depths of the right hand side of the equations and n the arity of x.

Proof The compact accessible solution σ' built from the accessible solution built from the solution σ is a solution and for every instantiable variable x, $\sigma' x$ has a depth less than or equal to $(n+1)(h+1) - 1$.

This bound is met, for instance by the example 3.

3.2 General Case

Let $a = b$ be an equation and let σ be a solution to this equation. We construct an interpolation problem $\Phi(a = b, \sigma)$ such that for every equation $(x\ c_1\ ...\ c_n) = b'$ of $\Phi(a = b, \sigma)$ we have $|b'| \leq |b|$, σ is a solution to $\Phi(a = b, \sigma)$ and every solution to $\Phi(a = b, \sigma)$ is a solution to $a = b$.

Definition 28 Let $a = b$ be an equation and let σ be a (ground) solution to this equation. By induction on the number of occurrences of a we construct an interpolation problem $\Phi(a = b, \sigma)$.

• If $a = \lambda x : T.d$ then since σ is a solution to the problem $a = b$ we have $b = \lambda x : T.e$ and σ is a solution to the problem $d = e$. We let

$$\Phi(a = b, \sigma) = \Phi(d = e, \sigma)$$

• If $a = (f\ d_1\ ...\ d_n)$ with f a constant or a local variable then since σ is a solution to $a = b$ we have $b = (f\ e_1\ ...\ e_n)$ and σ is a solution to the problems $d_i = e_i$. We let

$$\Phi(a = b, \sigma) = \bigcup_i \Phi(d_i = e_i, \sigma)$$

• If $a = (x\ d_1\ ...\ d_n)$ with x instantiable then for all i such that z has an occurrence in the normal form of the term $(\sigma x\ \sigma d_1\ ...\ \sigma d_{i-1}\ z\ \sigma d_{i+1}\ ...\ \sigma d_n)$ we let $c_i = \sigma d_i$ and $H_i = \Phi(d_i = \sigma d_i, \sigma)$ (obviously σ is a solution to $d_i = \sigma d_i$). Otherwise we let $c_i = z_i$ where z_i is a new local variable and $H_i = \emptyset$. We let

$$\Phi(a = b, \sigma) = \{(x\ c_1\ ...\ c_n) = b\} \cup \bigcup_i H_i$$

Proposition 11 Let $t = (x\ d_1\ ...\ d_n)$ be a term and let σ be a substitution. Let $c_i = \sigma d_i$ if z has an occurrence in $(\sigma x\ \sigma d_1\ ...\ \sigma d_{i-1}\ z\ \sigma d_{i+1}\ ...\ \sigma d_n)$ and $c_i = z_i$ where z_i is a new local variable of the same type as d_i otherwise. The variables z_i do not occur in the normal form of $(\sigma x\ c_1\ ...\ c_n)$.
Proof Let us assume that some of these variables have an occurrence in the normal form of $(\sigma x\ c_1\ ...\ c_n)$ and consider an outermost occurrence of such a variable z_i in the Böhm tree of the normal form of $(\sigma x\ c_1\ ...\ c_n)$. By part (2) of the key lemma, the variable z_i has also an occurrence in the normal form of term $(\sigma x\ c_1\ ...\ c_n)[z_j \leftarrow \sigma d_j \mid j \neq i]$ i.e. in the normal form of the term $(\sigma x\ \sigma d_1\ ...\ \sigma d_{i-1}\ z_i\ \sigma d_{i+1}\ ...\ \sigma d_n)$, which is contradictory.

Proposition 12 Let $a = b$ be an equation and let σ be a solution to this equation,

• the substitution σ is a solution to $\Phi(a = b, \sigma)$,

• conversely, if σ' is a solution to $\Phi(a = b, \sigma)$ then σ' is also a solution to the equation $a = b$.

Proof

• By induction on the number of occurrences of a. When a is an abstraction $a = \lambda x : T.d$ (resp. an atomic term whose head is a constant or local variable $a = (f\ d_1\ ...\ d_n)$) then b is also an abstraction $b = \lambda x : T.e$ (reps. an atomic term with the same head $b = (f\ e_1\ ...\ e_n)$) and by induction hypothesis σ is a solution to all the equations of the set $\Phi(d = e, \sigma)$ (resp. $\Phi(d_i = e_i, \sigma)$), so it is a solution to all the equations of $\Phi(a = b, \sigma)$.

When $a = (x\ d_1\ ...\ d_n)$ then by induction hypothesis σ is a solution to all the equations of the sets H_i and using the previous proposition the variables z_i have no occurrences in the term $(\sigma x\ c_1\ ...\ c_n)$ so we have

$$(\sigma x\ c_1\ ...\ c_n) = (\sigma x\ c_1\ ...\ c_n)[z_i \leftarrow \sigma d_i]$$

$$(\sigma x\ c_1\ ...\ c_n) = (\sigma x\ \sigma d_1\ ...\ \sigma d_n) = b$$

So σ is a solution to the equation $(x\ c_1\ ...\ c_n) = b$.

• By induction on the number of occurrences of a. Let σ' be a substitution solution to $\Phi(a = b, \sigma)$. If a is an abstraction $a = \lambda x : T.d$ (resp. an atomic term whose head is a constant or a local variable $a = (f\ d_1\ ...\ d_n)$) then b is also an abstraction $b = \lambda x : T.e$ (reps. an atomic term with the same head $b = (f\ e_1\ ...\ e_n)$) and by induction hypothesis we have $\sigma' d = e$ (resp. $\sigma' d_i = e_i$) and so $\sigma' a = b$.

If $a = (x\ d_1\ ...\ d_n)$ then we have

$$(\sigma' x\ c_1\ ...\ c_n) = b$$

and for all i such that z has an occurrence in $(\sigma x\ \sigma d_1\ ...\ \sigma d_{i-1}\ z\ \sigma d_{i+1}\ ...\ \sigma d_n)$ by induction hypothesis we have $\sigma' d_i = \sigma d_i$, so $c_i = \sigma' d_i$. Therefore

$$(\sigma' x\ c_1\ ...\ c_n)[z_i \leftarrow \sigma' d_i] = b[z_i \leftarrow \sigma' d_i]$$

$$(\sigma' x\ c_1\ ...\ c_n)[z_i \leftarrow \sigma' d_i] = b$$

$$(\sigma' x\ \sigma' d_1\ ...\ \sigma' d_n) = b$$

$$\sigma' a = b$$

Proposition 13 Let $a = b$ be an equation and let σ be a solution to this equation, if $a' = b'$ is an equation of $\Phi(a = b, \sigma)$ then $|b'| \leq |b|$.

Proof By induction on the number of occurrences of a. When a is an abstraction $a = \lambda x : T.d$ (reps. an atomic term whose head is a constant or a local variable $a = (f\ d_1\ ...\ d_n)$) then b is also an abstraction $b = \lambda x : T.e$ (reps. an atomic term with the same head $b = (f\ e_1\ ...\ e_n)$) and by induction hypothesis $|b'| \leq |e|$ (resp. $|b'| \leq |e_i|$) so $|b'| \leq |b|$.

When $a = (x\ d_1\ ...\ d_n)$ and the considered equation is $(x\ c_1\ ...\ c_n) = b$ then we have $b' = b$ so $|b'| \leq |b|$. When the considered equation is in one of the sets H_i, the set H_i is non empty so z has an occurrence in the normal form of the term $(\sigma x\ \sigma d_1\ ...\ \sigma d_{i-1}\ z\ \sigma d_{i+1}\ ...\ \sigma d_n)$ and $(\sigma x\ \sigma d_1\ ...\ \sigma d_{i-1}\ z\ \sigma d_{i+1}\ ...\ \sigma d_n)[z \leftarrow \sigma d_i] = b$ so using part (1) of the key lemma we have $|\sigma d_i| \leq |b|$ and by induction hypothesis $|b'| \leq |\sigma d_i|$ so $|b'| \leq |b|$.

Definition 29 Let Ψ be a third order matching problem and let σ be a solution to Ψ. We let $\Phi(\Psi, \sigma)$ be the following third order interpolation problem:

$$\Phi(\Psi, \sigma) = \bigcup_{a=b \in \Psi} \Phi(a = b, \sigma)$$

Proposition 14 Let Ψ be a third order matching problem and let σ be a solution to Ψ. Let h be the maximum of the depth of the right hand side of the equations of Ψ. Then σ is a solution to the problem $\Phi(\Psi, \sigma)$, each substitution σ' solution to the problem $\Phi(\Psi, \sigma)$ is a solution to Ψ and if $a' = b' \in \Phi(\Psi, \sigma)$ then $|b'| \leq h$.

Proof By propositions 12 and 13.

Lemma 4 Let Ψ be third order matching problem. Let h be the maximum of the depth of the the right hand side of the equations of Ψ. If this problem has a solution σ then it also has a solution σ' such that for every instantiable variable x, σx has a depth less than or equal to $(n+1)(h+1)-1$ where n the arity of x.
Proof The substitution σ is a solution to the problem $\Phi(\Psi, \sigma)$, thus, by lemma 3, this problem has a solution σ' such that for every instantiable variable x, $\sigma' x$ has a depth less than or equal to $(n+1)(h+1)-1$. This solution σ' is a solution to the problem Ψ.

Remark This method, in which an interpolation problem $\Phi(\Psi, \sigma)$ is constructed from a pair $< \Psi, \sigma >$ where Ψ is an arbitrary problem and σ a solution to Ψ, can be compared to the one used in the completeness proof of [9] in which a problem in solved form is constructed from such a pair.

4 A Decision Procedure

Theorem Third Order Matching is Decidable
Proof A decision procedure is obtained by considering the problem Φ and enumerating all the ground substitutions such that the term substituted for x has a depth less than or equal to $(n+1)(h+1)-1$, where h maximum depth of b for $a = b \in \Phi$ and n is the arity of x. If one of these substitutions is a solution then success else failure. This decision procedure is obviously sound. By lemma 4, it is complete.

Remark A more efficient decision algorithm is obtained by enumerating the nodes of the tree obtained by pruning Huet's search tree [5] [6] at each node corresponding to a substitution whose depth is larger than $(n+1)(h+1)-1$. This tree is obviously finite and thus this algorithm terminates. It is obviously sound. By lemma 4, it is complete.

Remark This result can be used to design an algorithm which enumerates a complete set of solutions to a third order matching problem and either terminates if the problem has a finite complete set of solutions or keeps enumerating solutions forever if it the problem admits no such set. Such an algorithm is got by enumerating the nodes of the tree obtained by pruning Huet's search tree [5] [6] at each node labeled with a problem that has no solution (by the theorem above, it is decidable if such a problem has a solution or not). Obviously, this algorithms still produces a complete set of solutions.

Let us show now that when a matching problem has a finite complete set of solutions then this algorithm terminates. Recall that a set of substitutions is called *minimal* if no substitution of this set is an instance of another and that Huet's algorithm applied to a matching problem produces a minimal complete set of solutions [6]. It is routine to verify that if a a problem has a finite complete set of solutions then any minimal complete set of solutions is also finite. So, if a problem has a finite complete set of solutions then Huet's tree for this problem has a finite number of success nodes and thus a finite number of nodes labeled with a problem that has a solution. The pruned tree is therefore finite and the algorithm obtained by enumerating its nodes terminates.

Remark This decidability result can be compared with the decidability of the equations of the form $P(x_1, ..., x_n) = b$ where P is a polynomial whose coefficients are natural numbers and b is a natural number.

If this equation has a solution $< a_1, ..., a_n >$ then it has a solution $< a'_1, ..., a'_n >$ such that $a'_1 \leq b$. Indeed either $Q(X) = P(X, a_2, ..., a_n)$ is not a constant polynomial and for all n, $Q(n) \geq n$, so $a_1 \leq b$, or the polynomial Q is identically equal to b and $< 0, a_2, ..., a_n >$ is also a solution. So a simple induction on n proves that if the equation has a solution then it also has a solution in $\{0, ..., b\}^n$ and an enumeration of this set gives a decision procedure.

Conclusion: Towards Higher Order Matching

The proof given here is based on the fact that if t is a third order term then when we reduce the term $(t\ c_1\ ...\ c_n)$, *in the general case*, we get a term deeper than t (or, at least, if it is not, the depth loss can bounded). This gives a bound (in terms of the depth of b) on the depth of the solutions of the equation $(x\ c_1\ ...\ c_n) = b$. In *the particular cases* in which the depth loss is greater than the bound, some part of the term t is superfluous and that we can construct a smaller term t' such that $(t'\ c_1\ ...\ c_n) = (t\ c_1\ ...\ c_n)$.

Generalizing this property of reduction to the full λ-calculus would give the decidability of higher order matching. To get the normal form of the term $(t\ c_1\ ...\ c_n)$ we have followed the strategy hinted by the weak normalization theorem and reduced first all the second order redexes, then all the first order redexes. So a generalization of this proof to higher order should require an induction on the maximal order of a redex. In the proof for the third order case, we quickly get the normal form of the term $(t\ c_1\ ...\ c_n)$ and we do not need to define the depth of a non-normal term. It seems that the generalization of this result to higher order requires such a definition.

Acknowledgments

The author would like to thank Gérard Huet, Richard Statman and Gopalan Nadathur for many very helpful discussions on this problem and remarks on previous drafts of this paper. This research was partly supported by ESPRIT Basic Research Action "Logical Frameworks".

References

[1] H. Barendregt, The Lambda Calculus, its Syntax and Semantics, *North Holland*, 1981, 1984.

[2] N.G. de Bruijn, Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem, *Indagationes Mathematicae*, 34, 5, 1972, pp. 381-392.

[3] S. Gorn, Explicit Definitions and Linguistic Dominoes, *University of Toronto*, 1967.

[4] J.R. Hindley, J.P. Seldin, Introduction to Combinators and λ-Calculus, *Cambridge University Press*, 1986.

[5] G. Huet, A Unification Algorithm for Typed λ-calculus, *Theoretical Computer Science*, 1, 1975, pp. 27-57.

[6] G. Huet, Résolution d'Équations dans les Langages d'Ordre 1,2, ..., ω, *Thèse de Doctorat d'État*, Université de Paris VII, 1976.

[7] G. Huet, B. Lang, Proving and Applying Program Transformations Expressed with Second Order Patterns, *Acta Informatica*, 11, 1978, pp. 31-55.

[8] D. A. Miller, Unification Under a Mixed Prefix, *Journal of Symbolic Computation*, 14, 1992, pp. 321-358.

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

[10] R. Statman, Completeness, Invariance and λ-definability, *Journal of Symbolic Logic*, 47, 1, 1982, pp. 17-26.

[11] D.A. Wolfram, The Clausal Theory of Types, *PhD Thesis*, University of Cambridge, 1989.

[12] M. Zaionc, The Set of Unifiers in Typed λ-Calculus as Regular Expression, *Proceedings of Rewriting Techniques and Applications*, Lecture Notes in Computer Science 202, Springer-Verlag, 1985, pp. 430-440.