Higher-order matching and tree automata

Hubert Comon and Yan Jurski
Laboratoire Spécification et Vérification, URA 2236 du CNRS
École Normale Supérieure de Cachan
61 avenue du président Wilson
94235 Cachan cedex, France
{comon,jurski}@lsv.ens-cachan.fr

1 Introduction

A *solution* of an equation $s = t$ where s, t are two terms of the simply typed lambda calculus is an assignment σ to the free variables of s, t such that $\sigma(s)$ and $\sigma(t)$ are equal modulo β-reduction and $\alpha\eta$ equivalence. Finding a solution (if one exists) is known as *higher-order unification* and was shown undecidable some time ago [5]. The higher-order matching problem consists in deciding the existence of a solution when t does not contain any free variables. This problem is still open.

In a classical way, we can associate to each type (and to each term of that type) an *order*: basic types o have order 1 and if $\tau = \tau_1, \ldots, \tau_n \to o$, then the order of τ is one plus the maximum of the orders of $\tau_1, \ldots, \tau_n$. If the order of all the free variables of s is smaller or equal to n, we get a *matching problem of order* n. For instance, first-order matching (as first-order unification) is decidable and there are actually either 1 or 0 solution to each matching problem. At order two, matching is again decidable (see e.g. [7]) and the number of solutions is, roughly, finite (up to α-conversion and if we discard the variables whose value is irrelevant and hence may be assigned to any term). Third-order matching is again decidable (see [2]), however the set of solutions might be infinite (modulo α-conversion) as shown by the example:

$$x(\lambda y.y) = a$$

where a is a constant of basic type. Then the solutions are the terms $\lambda x_1.x_1^n(a)$ where n is any natural number. This example also shows that there is not necessary any finite set of most general solutions (contrary to e.g. first-order unification).

4th order matching has also been shown decidable by V. Padovani [8] and there are partial results for fifth order matching [9].

Our purpose here is to relate tree automata and higher-order matching. More precisely, we show how to effectively compute an automaton which accepts the solutions of any 4th order matching problem. This gives of course a new proof of the decidability of 4th order matching since emptiness of the language recognized by a finite tree automaton can be decided (in linear time). This also provides a *representation* of the set of all solutions for 3rd or 4th order matching (the known decision algorithms do not yield such representations). This means in particular that 4th order matching equations can be combined: using the closure properties of recognizable languages, we may decide the emptiness of any Boolean combination of (or more generally, every first-order formula whose atoms are) such matching problems. This tree-automaton approach also allows us to derive some slightly more general results. For instance we can prove that 3rd-order matching is NP-complete, which shows that 3rd order matching is not harder than second order matching (which is also NP-complete). Up to now, the decision algorithms for third order matching were (at least) doubly exponential. For instance, G. Dowek's proof in [2] relies on a "pumping property" (which already suggest that there are tree automata around): he shows that a minimal-height solution should have a depth smaller than some polynomial in the data of the problem. Then the decision algorithm simply consists in trying all terms of adequate type and of depth bounded by this polynomial. In tree automata terms, this amounts to check emptiness by running the automaton on all terms of depth smaller than the number of states. This is not the best way of checking emptiness; using a marking technique, the reachability of a final state can be decided in linear time. Finally, we will try to structure the paper and the proof in such a way that the place where the order hypothesis is used is clearly circumscribed. For instance, the decidability proof of 4th order matching can be derived from the decidability proof of 3rd order matching, changing only few lines in the proof. We hope that this will give more insight on the reasons why 5th order matching is really more difficult than 4th order matching.

The paper tries to be self contained. It is structured as follows: section 1 contains the basic definitions we are using in lambda calculus. Section 2 explains the kind of automata we are using. Section 3 is devoted to the study of some particular matching problems: the 3rd order interpolation equations. In section 4, we show how to modify the proof of the previous section in order to show that the set of solutions of a fourth order interpolation equation is recognizable. In section 5, we give another proof that 3rd order matching can be reduced to interpolation equations and we derive from this reduction the NP-completeness of 3rd order matching. In section 6 we sketch Padovani's reduction, which yields the recognizability of solutions of general 4th order matching problems. Finally, we conclude in section 7 with some considerations on 5th-order matching.

2 Lambda Calculus, matching problems and interpolation equations

In this paper, we consider the simply typed lambda calculus. For other calculi such as e.g. polymorphic λ-calculi or Gödel's system T, pattern matching is known to be undecidable in general [3] and decidable at third order [11, 12].

2.1 Types

Definition 1. The set $\mathcal{T}$ of *types* is the smallest set containing the constant symbol o (basic type) and such that $\sigma \to \tau \in \mathcal{T}$ whenever $\sigma, \tau \in \mathcal{T}$.

We assume here that there is only one basic type o. However everything can be extended to a finite number of basic types in a straightforward way. (It can even be extended to calculi with subtypes induced by an ordering on the basic types).

In order to fit with the usual notations of tree automata, a type $\sigma_1 \to (\sigma_2 \to \ldots (\sigma_n \to \sigma) \ldots)$ is written $\sigma_1, \ldots, \sigma_n \to \sigma$. Then, every type has the form $\sigma_1, \ldots, \sigma_n \to o$.

Definition 2. The *order* O of a type $\tau \in \mathcal{T}$ is defined by:

- $O(o) = 1$
- $O(\sigma_1, \ldots, \sigma_n \to o) = 1 + \max(O(\sigma_1), \ldots, O(\sigma_n))$.

Finally, $Sub(\tau)$ is the set of types occurring in τ (i.e. the set of subterms of τ when τ is viewed as a term on the alphabet $\{o, \to\}$).

2.2 Terms

Definition 3. Let $\mathcal{C}$ be an alphabet of *constant symbols*, each symbol coming with its type in $\mathcal{T}$ and $\mathcal{X}$ be an alphabet of *variable symbols* each variable also coming with its type. Then the set of (typed) terms $T(\mathcal{C}, \mathcal{X})$ is the least set such that:

- $\mathcal{X}, \mathcal{C} \subseteq T(\mathcal{C}, \mathcal{X})$
- If $t \in T(\mathcal{C}, \mathcal{X})$ has type τ (which is abbreviated $t : \tau$) and $x : \sigma \in \mathcal{X}$, then $\lambda x.t : \sigma \to \tau \in T(\mathcal{C}, \mathcal{X})$
- If $u_1 : \sigma_1, \ldots, u_n : \sigma_n, t : \sigma_1, \ldots, \sigma_n \to o \in T(\mathcal{C}, \mathcal{X})$, then $t(u_1, \ldots, u_n) \in T(\mathcal{C}, \mathcal{X})$.

Note that this definition assumes that the terms are in so-called *η-long form* (except may be for the constants and the variables).

Substitutions are mappings from the set of variables to the set of terms, which preserve the types. We write $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ the mapping which assigns to each x_i the term t_i and which is the identity outside $\{x_1, \ldots, x_n\}$. Substitutions will be used in postfix notation and we always assume renamings (α-conversions) which avoid the capture of variables. If $\mathcal{C}_1 \subseteq \mathcal{C}$, a substitution is *out of* $\mathcal{C}_1$ if, for every variable $x \in \mathcal{X}$ and every $c \in \mathcal{C}_1$, $x\sigma$ does not contain any occurrence of c.

The (one step) β-reduction is defined as usual as the least binary relation $\underset{\beta}{\to}$ which is compatible with the term structure and such that for every term $(\lambda x u)(v) \in T(\mathcal{C}, \mathcal{X})$, $(\lambda x u)(v) \underset{\beta}{\to} u\{x \mapsto v\}$.

$\underset{\beta}{\to}$ is strongly normalizing and confluent on simply typed terms (see e.g. [6]). Hence each term t has a unique normal form (up to α-conversion), which is denoted $t\downarrow$.

The transitive closure (resp. reflexive and transitive) closure of a binary relation $\to$ is denoted by $\xrightarrow{+}$ (resp. $\xrightarrow{*}$).

For each term t, the set of *relevant types* $Sub(t)$ is the finite set of types which occur as a subterm of the type of t. For instance, assume that $t : (o, o \to o), o \to o$, then $Sub(t) = \{o; o, o \to o; (o, o \to o), o \to o\}$. Finally, the *order* of a term is the order of its type.

2.3 Pattern matching

Definition 4. Given two terms $u, v \in T(\mathcal{C}, \mathcal{X})$ and a set $\mathcal{C}_1 \subseteq \mathcal{C}$, a *solution* (resp. a *solution out of* $\mathcal{C}_1$*) of the equation* $u = v$ *is a substitution (resp. a substitution out of* $\mathcal{C}_1$) σ such that $u\sigma\downarrow =_\alpha v\sigma\downarrow$.

The (higher-order) pattern matching problem (resp. the matching problem out of $\mathcal{C}_1$) is, given two terms $u, v \in T(\mathcal{C}, \mathcal{X})$ such that v does not contain any free variable, does there exist a solution to $u = v$?

The *order* of a matching problem $u = v$ is the maximal order of a free variable in u. General second-order matching is NP-complete (we do not know any reference to this result. Hence we sketch in the following the easy encoding of 3-SAT into 2nd order matching). General third order matching is decidable [1] and the known upper bound is a double exponential. Fourth order matching is decidable [8] and the known upper bound is a tower of several exponentials (6?).

Without loss of generality, we may assume that u, v are in normal form (which we will do in the following). We may also restrict our attention to solutions σ which map variables to terms in normal form.

If u is an abstraction : $u = \lambda x_1 \dots \lambda x_n.u'$, then, for any substitution σ, $u\sigma\downarrow$ $= \lambda x_1 \dots \lambda x_n.u'\sigma\downarrow$, hence v has to be an abstraction as well: $v = \lambda y_1 \dots \lambda y_n.v'$ and σ is a solution of $u = v$ iff σ is a solution out of $\{x_1, \dots, x_n\}$ of $u' = v'\{y_1 \mapsto x_1, \dots y_n \mapsto x_n\}$ Hence, at the price of going from matching to matching out of a set of constants, hereafter called *frozen constants*, we may always assume that the right hand side of a matching problem has type o.

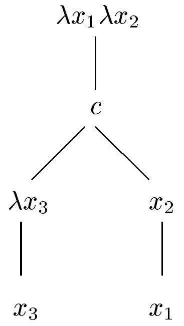
3 Tree automata

3.1 λ-terms as trees

As usual, positions in a tree are (finite) sequences of positive integers. A set of positions is assumed to be prefix closed and to satisfy $p \cdot i \in P \Rightarrow p \cdot j \in P$ for every integers $1 \leq j \leq i$. A *tree* t consists in a finite set of positions $Pos(t)$ together with a labeling function t from $Pos(t)$ into $\mathcal{F}$, a set of *function symbols*. Each function symbol f comes with its arity $a(f)$. The labeling should be such that the number of sons of a node labeled with f is exactly $a(f)$.

We use a representation of λ-terms as trees (which is similar to Böhm trees): the set $\mathcal{F}$ consists in $\mathcal{C} \cup \mathcal{X} \cup \{\lambda x_1 \dots, \lambda x_n \mid n \geq 1, x_1, \dots, x_n \in \mathcal{X}\}$. The arity of $f \in \mathcal{C} \cup \mathcal{X}$ is n if the type of f is $\sigma_1, \dots, \sigma_n \to o$. Abstractions $\lambda x_1 \dots \lambda x_n$ are assumed to have arity one.

For instance, $\lambda x_1 \lambda x_2.c(\lambda x_3.x_3, x_2(x_1))$ (assumed in normal form) has the following representation as a tree:



In what follows, we assume that $\mathcal{F}$ is finite. This is not a restriction as, for countably infinite alphabets, there is always another alphabet $\mathcal{F}'$, which is finite, and an injective tree homomorphism h from $T(\mathcal{F})$ into $T(\mathcal{F})'$ such that $h(T(\mathcal{F}))$ is recognizable by a finite tree automaton and the size of $h(t)$ is linear with respect to the size of t. [1] However, for sake of clarity, we will keep the standard notations instead of using the encodings of $\mathcal{F}$.

3.2 $\square$-automata

We will use a slight modification of tree automata. The main difference with the definitions of [13, 4] is the presence of special symbols $\square_\tau$ which should be interpreted as any term of type τ. This slight modification is necessary because, for instance, the set of all closed terms is not recognizable by a classical finite tree automaton: roughly, while running the automaton on a term t, one should remember which variables are free in the term and the amount of memory which is required is not bounded independently of t. Using Böhm trees (in which free variables are indicated in the node label) does not help since we would have to check that the labels indeed correspond to free variables, which again requires an unbounded memory.

The following definition is thus a slight modification of the classical definition of bottom-up tree automata. The only change is the addition of boxes $\square_\tau$ which can be replaced with any term of type τ.

Definition 5. A $\square$-*automaton* $\mathcal{A}$ is a tuple $(\mathcal{F}, Q, Q_f, \Delta)$ where:

- $\mathcal{F}$ is a finite alphabet of labels, including a finite set of 0-ary special function symbols $\square_{\tau_1}, \ldots, \square_{\tau_m}$
- Q is a finite set of *states*
- Q_f is a subset of Q; its members are the *final states*

[1] For instance, if $x_i, i \geq 0$ is the set of first order variables, they will be represented using a unary tree $x(\bar{i})$ where $\bar{i}$ is a tree representation of the integer i.

– Δ is a *transition relation*: it is a finite set of rules of the form

$$f(q_1, \ldots, q_n) \to q$$

where $f \in \mathcal{F}$ has arity n and $q_1, \ldots, q_n, q \in Q$

Definition 6. The *forgetful relation* $\sqsubseteq$ is defined as the least relation on terms such that:

$\Box_\tau \sqsubseteq u$ for every term u of type τ.
$u_1 \sqsubseteq v_1, \ldots u_n \sqsubseteq v_n \Rightarrow f(u_1, \ldots, u_n) \sqsubseteq f(v_1, \ldots, v_n)$ for every symbol f of adequate type.

A *box-replacement* of a term u is a term v such that $u \sqsubseteq v$ and v does not contain any symbol $\Box_\tau$.

Definition 7. A $\Box$-automaton $\mathcal{A}$ *accepts* (resp.*accepts in state* q) a term $t \in T(\mathcal{C}, \mathcal{X})$ if there is a term $u \in T(\mathcal{C} \cup \{\Box_{\tau_1}, \ldots, \Box_{\tau_n}\}, \mathcal{X})$ and a state $q_f \in Q_f$ such that $u \xrightarrow[\Delta]{*} q_f$ (resp. $u \xrightarrow[\Delta]{*} q$) and $u \sqsubseteq t$.

The *language* $\mathcal{L}(\mathcal{A})$ recognized by a $\Box$-automaton $\mathcal{A}$ is the set of terms which are accepted by $\mathcal{A}$.

Example 1. Consider the following $\Box$ automaton:

– $\mathcal{F} = \{a, \lambda x_1, \Box_o, x_1\}$ where a is a constant and x_1 is a variable of type $o, o \to o$
– Q consists of three states $q_\Box$, q_f and q_a; q_f is the final state.
– Δ is the set of transition rules:

$$\begin{array}{rcl} a \to q_a & & \Box_o \to q_\Box \\ x_1(q_a, q_\Box) \to q_a & & \lambda x_1 q_a \quad \to \quad q_f \end{array}$$

The automaton $\mathcal{A}$ recognizes the language of solutions of $x(\lambda y_1 \lambda y_2.y_1) = a$ (up to α-conversion). For instance the term $\lambda x_1.x_1(x_1(a, b), c)$ where b, c are arbitrary terms, is accepted by $\mathcal{A}$ as shown by the computation of the automaton:

$$\lambda x_1.x_1(x_1(a, \Box_o), \Box_o) \xrightarrow[\Delta]{*} \lambda x_1.x_1(x_1(q_a, q_\Box), q_\Box) \xrightarrow[\Delta]{} \lambda x_1.x_1(q_a, q_\Box) \xrightarrow[\Delta]{} \lambda x_1.q_a \xrightarrow[\Delta]{} q_f$$

Then $\lambda x_1.x_1(x_1(a, \Box_o), \Box_o) \sqsubseteq \lambda x_1.x_1(x_1(a, b), c)$.

The following result shows that we can use $\Box$-recognizability as recognizability:

Theorem 8. *The set of $\Box$-recognizable subsets of Λ is closed under Boolean operations, and the complexity of each operation is the same as for the operations on usual tree automata.*

Sketch of the proof:
The proof of this result is similar to the proof of closure properties of finite tree automata. We only have to be slightly more careful in the reduction of non-determinism (which is necessary for complementation): if we have two rules $f(q, q_{\Box_\tau}) \to q_1$ and $f(q, q_2) \to q_3$, we cannot consider the automaton as deterministic. To see what happens in this case, it is better to use an algebraic characterization of the languages: the language accepted by the $\Box$-automaton $\mathcal{A}$ is the union of L_{q_f} for $q_f \in Q_f$ and $\{L_q, q \in Q\}$ is the least solution of the set of equations:

$$L_q = \bigcup_{i=1}^{n} f_i(L_{q_{1,i}}, \ldots, L_{q_{m_i,i}})$$

where $f_i(q_{1,i}, \ldots, q_{m_i,i}) \to q$ are the transition rules whose target is q and $f_1, \ldots, f_n$ are not boxes, and $L_q = T_\tau$ (the whole set of terms of type τ) if some $f_i = \Box_\tau$. Then the complement can be computed by complementing both sides of the equations:

$$\overline{L_q} = \bigcap_{i=1}^{n}((\bigcup_{f \neq f_i} f(T_{\tau_1}, \ldots, T_{\tau_m})) \cup (\bigcup_{l=1}^{m_i} f_i(T_{\tau_1}, \ldots, \overline{L_{q_l}}, \ldots, T_{\tau_{m_i}}))$$

Then we are left to prove that the intersection (resp. union) of two recognizable languages can be computed in polynomial time. It is quite straightforward for the union. Concerning intersection, we have to be a bit more careful: as in the classical case, the states of the intersection automaton contain pairs of states of each component automaton. There are also additional rules for arguments of the form $(q, q_{\Box_\tau})$ or $(q_{\Box_\tau}, q)$. $\diamond$

3.3 Automata on tuples

As for classical tree automata, terms can be overlapped, yielding the notion of *recognizability of tuple of terms.* Then the class of recognizable sets of tuples is, in addition, closed under projection.

4 Third order interpolation equations

We consider first some particular kinds of third order matching problems, which, as we will see, are at the heart of the problem.

Definition 9. An *interpolation equation* is an equation

$$x(s_1, \ldots, s_n) = t$$

where $s_1, \ldots, s_n, t$ are closed terms in normal form

Assume $\tau(x) = \tau_1, \ldots, \tau_n \to o$.

In the automaton $\mathcal{A}_{s_1,\ldots,s_n,t} = (\mathcal{F}, Q, Q_f, \Delta)$, Q only depends on t. The idea is the following: any solution can be written $\lambda x_1, \ldots \lambda x_n.u$. If a term of type o, say v, occurs in u, then either it will be reduced to a subterm of t along the reduction of $u\{x_1 \mapsto s_1, \ldots x_n \mapsto s_n\}$ to t, or else v could be replaced by any term without affecting the result of the reduction. (This corresponds to Padovani's observational equivalence classes). The set of states reflects this property: if $v\{x_1 \mapsto s_1, \ldots x_n \mapsto s_n\}$ reduces to a subterm t' of t, then it will be accepted in state $q_{t'}$.

The alphabet $\mathcal{F}$ consists in

- the constant symbols occurring in t and, more generally, every label of a node in the tree representation of t (except the frozen constants if any)
- the symbols $x_1, \ldots, x_n$ of type $\tau_1, \ldots, \tau_n$ respectively, which are assumed to be distinct from the above symbols
- the special symbols $\Box_\tau$ for every $\tau \in Sub(x)$

The set of states Q consists of all subterms of t, which we write q_u (instead of u) and a state $q_{\Box_o}$. In addition, we have the final state q_f.

The transition rules Δ consist in

- The rules
$$f(q_{t_1}, \ldots, q_{t_n}) \to q_{f(t_1,\ldots,t_n)}$$
each time $q_{f(t_1,\ldots,t_n)} \in Q$
- For $i = 1, \ldots, n$, the rules
$$x_i(q_{t_1}, \ldots, q_{t_n}) \to q_u$$
where u is a subterm of t such that $s_i(t_1, \ldots, t_n)\!\downarrow = u$ and $t_j = \Box_o$ whenever $s_i(t_1, \ldots, t_{j-1}, \Box_o, t_{j+1}, \ldots, t_n)\!\downarrow = u$.
- the rule $\lambda x_1, \ldots, \lambda x_n.q_t \to q_f$

We claim that a term u is a solution of the interpolation equation if and only if it is α-equivalent to a term which is accepted by $\mathcal{A}_{s_1,\ldots,s_n,t}$.

Example 2. Let us consider the interpolation equation

$$x(\lambda y_1 \lambda y_2.y_1, \lambda y_3.f(y_3, y_3)) = f(a, a)$$

where y_1, y_2 are assumed to be of base type o. Then $\mathcal{F} = \{a, f, x_1, \Box_o\}$. $Q = \{q_a, q_{f(a,a)}, q_{\Box_o}\}$ and the rules of the automaton are:

$$\begin{array}{rr}
a \to q_a & f(q_a, q_a) \to q_{f(a,a)} \\
\Box_o \to q_{\Box_o} & x_1(q_a, q_{\Box_o}) \to q_a \\
x_1(q_{f(a,a)}, q_{\Box_o}) \to q_{f(a,a)} & x_2(q_a) \to q_{f(a,a)} \\
\lambda x_1 \lambda x_2.q_{f(a,a)} \to q_f &
\end{array}$$

For instance the term $\lambda x_1 \lambda x_2.x_1(x_2(x_1(x_1(a,\Box_o),\Box_o)),\Box_o)$ is accepted by the automaton :

$$\begin{aligned}
\lambda x_1 \lambda x_2.x_1(x_2(x_1(x_1(a,\Box_o),\Box_o)),\Box_o) &\xrightarrow[\mathcal{A}]{*} \lambda x_1 \lambda x_2.x_1(x_2(x_1(x_1(q_a,q_{\Box_o}),q_{\Box_o})),q_{\Box_o}) \\
&\xrightarrow[\mathcal{A}]{} \lambda x_1 \lambda x_2.x_1(x_2(x_1(q_a,q_{\Box_o})),q_{\Box_o}) \\
&\xrightarrow[\mathcal{A}]{} \lambda x_1 \lambda x_2.x_1(x_2(q_a),q_{\Box_o}) \\
&\xrightarrow[\mathcal{A}]{} \lambda x_1 \lambda x_2.x_1(q_{f(a,a)},q_{\Box_o}) \\
&\xrightarrow[\mathcal{A}]{} \lambda x_1 \lambda x_2.q_{f(a,a)} \\
&\xrightarrow[\mathcal{A}]{} q_f
\end{aligned}$$

And indeed, for every terms t_1,t_2,t_3 of type o, $\lambda x_1 \lambda x_2.x_1(x_2(x_1(x_1(a,t_1),t_2)),t_3)$ is a solution of the interpolation problem.

We first show an order-dependent technical lemma.

Lemma 10. *If $\mathcal{E}$ is a set of terms which is closed under subterm, then for any second-order term s and any first order terms $u_1,\dots,u_n$, $s(u_1,\dots,u_n)\downarrow \in \mathcal{E}$ implies that, for each j, either $u_j \in \mathcal{E}$ or else $s(u_1,\dots,u_{j-1},\Box_o,u_{j+1},\dots,u_n)\downarrow$ $= s(u_1,\dots,u_n)\downarrow$.*

Proof. Let $w = s(u_1,\dots,u_m)\downarrow \in \mathcal{E}$. Assume that $u_j \notin \mathcal{E}$ for some j. We show by induction on the size of w that

$$s(u_1,\dots,u_{j-1},\Box_o,u_{j+1},\dots,u_m) \xrightarrow[\beta]{*} w$$

Let $s = \lambda z_1 \dots \lambda z_m.r$,

$$s(u_1,\dots,u_n) \xrightarrow[\beta]{+} r\{z_1 \mapsto u_1,\dots z_m \mapsto u_m\}$$

$O(r) = 1$. If $r = z_k$ for some k, then $u_k = w$ and j must be distinct from k. Then we have also $r\{z_1 \mapsto u_1,\dots,z_{j-1} \mapsto u_{j-1}, z_j \mapsto \Box_o, z_{j+1} \mapsto u_{j+1},\dots,z_m \mapsto u_m\} = w$. If r is not a variable, $r = c(r_1,\dots,r_k)$ and $w = c(w_1,\dots,w_k)$. Moreover, $(\lambda z_1,\dots,\lambda z_m.r_l)(u_1,\dots,u_m)\downarrow = w_i$ and, by induction hypothesis, $(\lambda z_1,\dots,\lambda z_m.r_l)(u_1,\dots,u_{j-1},\Box_o,u_{j+1},\dots,u_m)\downarrow = w_l$, which yields the result. $\diamond$

Theorem 11. *Let $s_1,\dots,s_n,t$ be closed terms. The automaton $\mathcal{A}_{s_1,\dots,s_n,t}$ accepts the set of solutions (resp. the solutions out of $\mathcal{C}_1$) up to α-conversion of the third order equation $x(s_1,\dots,s_n) = t$.*

Proof. **Assume that u is a solution** (in normal form). Then (only using the typing constraints) u is α-equivalent to some term $\lambda x_1.\dots.\lambda x_n.v$ where the only free variables of v are $x_1,\dots,x_n$ and $\tau(x_i) = \tau(s_i)$ for every i. Let σ be the substitution $\{x_1 \mapsto s_1,\dots x_n \mapsto s_n\}$ Let us show by induction on the size of v that if w is a subterm of t and $Var(v) \subseteq Var(w) \cup \{x_1,\dots,x_n\}$ and $v\sigma\downarrow = w$ then v is accepted in state q_w.

Base cases When v is a constant (or a variable) (of type o) and occurs in t, then, by construction, there is a rule $v \rightarrow q_v$.

When v is a variable (of type o) not occurring in t, then $v = x_i$ for some i and, by construction of the automaton, there is a rule $x_i \rightarrow q_{s_i}$.

Induction step Now, assume that v has a size larger or equal to 2. There are three possible constructions:

$v = \lambda z.v_1$. And z may be assumed distinct from x_i. Then $v\sigma\downarrow = \lambda z.v_1\sigma\downarrow$. Since it should be a subterm of t, z may be assumed (at the price of an α conversion) to be bound in t and $v\sigma\downarrow = \lambda z.w_1$ where w_1 is a subterm of t. By induction hypothesis, v_1 is accepted in state q_{w_1} and there is a rule $\lambda z.q_{w_1} \underset{\mathcal{A}}{\rightarrow} q_{\lambda z.w_1}$. Hence v is accepted in state $q_{\lambda z.w_1}$.

$v = z(v_1, ...v_m)$ **where** z **is variable** . Then either z is a variable which occurs free in some subterm of t or $z = x_i$ for some i. in the first case, $w = v\sigma\downarrow = z(v_1\sigma\downarrow, \ldots v_m\sigma\downarrow)$ is a subterm of t and hence $w_i = v_i\sigma\downarrow$ is a subterm of t for all i. Then, by induction hypothesis, each term v_i is accepted in state q_{w_i}. Finally, using the rule $z(q_{w_1}, \ldots q_{w_m}) \underset{\mathcal{A}}{\rightarrow} w$, v is accepted in state q_w.

In the second case, $w = v\sigma\downarrow = s_i(v_1\sigma\downarrow, \ldots, v_m\sigma\downarrow)\downarrow$. Let $u_j = v_j\sigma\downarrow$. $O(x) = 3$, hence $O(s_i) = 2$ and we can apply lemma 10 with $\mathcal{E}=$ the set of subterms of t: for each index j, either u_j is a subterm of t, in which case, by induction hypothesis, v_j is accepted in state q_{u_j}, or else v_j can be replaced with $\Box_o$ without changing the result of the reduction: let $q_j = q_{u_j}$ if u_j is a subterm of t and $q_{\Box_o}$ otherwise. Using the rule $x_i(q_1, \ldots, q_m) \underset{\mathcal{A}}{\rightarrow} q_w$, v is a term accepted by the automaton in state q_w.

$v = c(v_1, ..., v_m)$ where c is a constant. Then $v\sigma\downarrow = c(v_1\sigma\downarrow, \ldots v_n\sigma\downarrow)$ and $w_i = v_i\sigma\downarrow$ is a subterm of t. By induction hypothesis, v_i is accepted in state q_{w_i} for all i and, thanks to the rule $c(q_{w_1}, \ldots, q_{w_m}) \underset{\mathcal{A}}{\rightarrow} q_w$, v is accepted in state q_w.

It follows in particular that, if $\lambda x_1, \ldots, \lambda x_n.v$ is a solution, then v is accepted by the automaton in state q_t. Hence $\lambda x_1, \ldots, \lambda x_n.v$ is accepted by the automaton.

Assume that $u \overset{*}{\underset{\mathcal{A}}{\rightarrow}} q_f$ and let us show that for any $\Box$-replacement v of u, $\lambda x_1 \ldots \lambda x_n.v$ is a solution. Actually, we show more generally, by induction on the size of u, that if $u \overset{*}{\underset{\mathcal{A}}{\rightarrow}} q_w$, then $v\sigma\downarrow = w$.

Base cases If u is a constant then $u = v = w$ (the only possible transition from u is $u \underset{\mathcal{A}}{\rightarrow} q_u$).

If u is a variable, then u must be either a variable occurring in t in which case $u = w$ or some variable x_i. In the latter case, $u \underset{\mathcal{A}}{\rightarrow} q_w$ only if $w = s_i = v\sigma$.

Induction step We investigate all possible constructions of u:

$u = c(u_1, \ldots, u_m)$ where c is either a constant or a variable bound in t. Then the only possible rule yielding q_w is

$$c(q_1, \ldots, q_n) \rightarrow q_w$$

where $q_i = q_{w_i}$ and $w = c(w_1, \ldots, w_m)$ and every u_i is accepted in state q_i. Then, by induction hypothesis, $v_i\sigma\downarrow = w_i$ and v_i is a $\Box$-replacement of u_i for every i. Then let $v = c(v_1, , v_n)$. $v\sigma\downarrow = c(w_1, \ldots, w_n) = w$ and v is a $\Box$-replacement of u

$u = \lambda z_1, \ldots \lambda z_m.u_1$ According to the definition of the automaton, there should be a subterm $\lambda z_1, \ldots, \lambda z_m.w_1$ of t such that $u_1 \xrightarrow[\mathcal{A}]{*} q_{w_1}$. Then, by induction hypothesis, there is a v_1 which is a $\Box$-replacement of u_1 and such that $v_1\sigma\downarrow = w_1$ and hence $v\sigma\downarrow = w$, if $v = \lambda z_1.v_1$.

$u = x_i(u_1, \ldots, u_m)$ According to the definition of the automaton, for every i, $u_i \xrightarrow[\mathcal{A}]{*} q_{w_i}$ such that $s_i(w_1, \ldots, w_m)\downarrow = w$. For every index i such that q_{w_i} is in the set of states, by induction hypothesis, there is a term v_i such that v_i is a $\Box$-replacement of u_i and $v_i\sigma\downarrow\ w_i$. For the other indices, we may replace the occurrences of $\Box_o$ with the appropriate v_is in $s_i(w_1, \ldots, w_n)$, without modifying the result of the reduction: $s_i(v_1, \ldots, v_n)\downarrow = w$.

which complete the converse implication. Note that we did not use the order assumption in this part of the proof.
$\diamond$

Lemma 12. *The size of the automaton $\mathcal{A}_{s_1,\ldots,s_n,t}$ is $O(n \times m \times |t|)$ where m is the maximal number of arguments of an s_i.*

Sketch of the proof: The number of states is $O(|t|)$. Now, the total size of all rules of the form $f(q_{t_1}, \ldots, q_{t_m}) \rightarrow q_{f(t_1,\ldots,t_m)}$ is at most $2 \times |t|$.[2] Finally, according to lemma 10, for each i, the total size of all rules $x_i(q_{u_1}, \ldots, q_{u_m}) \rightarrow q_u$ is at most $|t| \times (m + 2)$. $\diamond$

Note that the automaton is not *any* tree automaton: for instance it can be seen easily that the language has star height 1. What is more interesting for us is the following result:

Lemma 13. *The existence of a solution to a system of n third order interpolation equations can be decided in non-deterministic polynomial time.*

Sketch of the proof: We may assume without loss of generality that there is only one free variable in the system (otherwise, there is a solution iff there is a solution for each of the individual systems).

In the automaton $\mathcal{A}_{s_1,\ldots,s_n,t}$, the set of rules can be divided in three parts:

[2] The *size* of a rule $f(q_1, \ldots, q_m) \rightarrow q$ is $m + 2$.

- the looping rules of the form $x_i(q_{\Box_o}, \ldots q_{\Box_o}, q_u, q_{\Box_o}, \ldots, q_{\Box_o}) \rightarrow q_u$ which correspond to a term s_i which is the jth projection
- rules which increase the states: assuming that $\Box_o$ is smaller than any other terms and that the subterms of t are ordered using the subterm ordering,

$$f(q_{u_1}, \ldots, q_{u_m}) \rightarrow q_u \Rightarrow \forall j, u_j < u$$

  where f is a constant or a variable.
- the rule yielding the final state

We may decompose $\mathcal{A}_{s_1,\ldots,s_n,t}$ as a finite union of automata whose rules consist in looping rules, the rule yielding the final state and a set of rules in which each state occurs at most once. For all such automata, the states are included in the set of occurrences of t plus the $\Box$-state (but may not be included in the original set of states: different positions of the same subterm of t are now distinghished, see the example below). For instance consider the automaton of example 2. It is split into two automata whose rules are respectively:

$$\begin{array}{rr} a \rightarrow q_a & \Box_o \rightarrow q_{\Box_o} \\ x_1(q_a, q_{\Box_o}) \rightarrow q_a & x_1(q_{f(a,a)}, q_{\Box_o}) \rightarrow q_{f(a,a)} \\ x_2(q_a) \rightarrow q_{f(a,a)} & \lambda x_1 \lambda x_2 . q_{f(a,a)} \rightarrow q_f \end{array}$$

and

$$\begin{array}{rr} a \rightarrow q_{a,1} & \Box_o \rightarrow q_{\Box_o} \\ x_1(q_{a,1}, q_{\Box_o}) \rightarrow q_{a,1} & x_1(q_{f(a,a)}, q_{\Box_o}) \rightarrow q_{f(a,a)} \\ f(q_{a,1}, q_{a,2}) \rightarrow q_{f(a,a)} & a \rightarrow q_{a,2} \\ x_1(q_{a,2}, q_{\Box_o}) \rightarrow q_{a,2} & \lambda x_1 \lambda x_2 . q_{f(a,a)} \rightarrow q_f \end{array}$$

Now, intersecting two automata of the above form, we get again a finite disjunction of automata of the same form and whose size of productive rules is bounded by $O(n \times m \times (|t_1| + |t_2|))$ (where n is the arity of x), i.e. the sum of the sizes of the two intersected automata. If we want to check the emptiness of the intersection, we simply guess one automaton each time there is a disjunction and check the emptiness of the resulting automaton, whose size is $O(n \times m \times |t|)$. $\Diamond$

This contrasts with the result on arbitrary tree automata for which the emptiness of intersection is EXPTIME-complete [10].

5 Fourth order interpolation equations

Theorem 11 uses the order of the matching problem only through lemma 10. Our purpose is then to generalize lemma 10 to fourth order, in such a way that the proof of theorem 11 also works for fourth order.

Lemma 14. *Let $\mathcal{E}$ be a set of terms which is closed under subterm and such that, for every $k \leq N$, for every first-order terms $r_1, \ldots, r_k \in \mathcal{E} \cup \{\Box_o\}$, for every first order term $v \in \mathcal{E}$, then the solutions of $z(r_1, \ldots, r_k) = v$ are again in*

$\mathcal{E}$. Assume that $s(u_1,\ldots,u_m)\downarrow \in \mathcal{E}$ for some third order term s and that N is larger than the largest number of arguments of any $\tau \in Sub(s)$. Then, for every j, either $u_j \in \mathcal{E}$ or else $s(u_1,\ldots,u_{j-1},\Box_\tau,u_{j+1},\ldots,u_m)\downarrow = s(u_1,\ldots,u_m)\downarrow$.

Sketch of the proof. We "freeze" u_j in the computation of $s(u_1,\ldots,u_m)\downarrow$: the redexes involving u_j are delayed as long as possible. We get a reduction sequence

$$s(u_1,\ldots,u_m) \xrightarrow[\beta]{*} C[u_j(r_1^1,\ldots,r_1^k),\ldots u_j(r_n^1,\ldots,r_n^k)] \xrightarrow[\beta]{*} u$$

where C is an irreducible context, the terms r_i^l are first order terms in normal form and $u \in \mathcal{E}$. $k \le N$ and u must be of the form $u = C[v_1,\ldots,v_n]$. Hence either $n = 0$ or u_j is a solution of the system of second order interpolation equations

$$z(r_i^1,\ldots,r_i^k) = v_i$$

According to lemma 10, each r_i^l is either a subterm of v_i or else it can be replaced with $\Box_o$ without changing the set of solutions. Now, r_i^l is a subterm of v, hence in $\mathcal{E}$, as soon as it is a subterm of v_i. Hence $u_j \in \mathcal{E}$ if $n > 0$. $\diamond$

This suggests to use the same automaton as in the previous section, except that the set of states has to be closed under "solutions of second-order interpolation equations". The next lemma shows that this indeed yields a finite number of states.

Lemma 15. *The least set Q_N containing t, closed by subterm and such that for every $k \le N$ for every first-order terms $r_1,\ldots,r_k \in Q_N \cup \{\Box_o\}$, for every first order term $v \in Q_N$, then the solutions of $z(r_1,\ldots,r_k) = v$ are again in Q_N is finite.*

Sketch of the proof. Q_N can be computed as a least fixed point, starting with t and iterating the closure under subterm and under solutions of second order interpolation equations.

A first iteration of the two closures gives rise to terms of the form

$$\lambda z_1,\ldots,\lambda z_k.C[z_{i_1},\ldots,z_{i_m}]$$

where $z_{i_1},\ldots,z_{i_m} \in \{z_1,\ldots,z_k\}$ and C is an irreducible context such that $u = C[u_1,\ldots,u_m]$ for some subterm u of t. At the second iteration, $z_1,\ldots,z_k$ belong to Q_N (because of the closure by subterm). However, any solution of an interpolation equation over this new set of terms has the form

$$\lambda y_1\ldots\lambda y_k.C[y_{i_1},\ldots,y_{i_m},z_{j_1},\ldots,z_{j_n}]$$

and we have always $k+n \le |t|$. Actually, at any iteration, we need at most $|t|$ distinct variables to express the solutions (these variables may be re-used) which yields the finiteness of Q_N. $\diamond$

Example 3. Let $t = c(a)$ and $N = 1$. Initially $Q_N = \{c(a)\}$. Closing under subterm gives rise to a new element: a. Then we have to solve several second-order interpolation equations:

Interpolation equation	Solutions
$z(\Box_o) = a$	$\lambda z_1.a$
$z(\Box_o) = c(a)$	$\lambda z_1.c(a)$
$z(a) = a$	$\lambda z_1.z_1;\ \lambda z_1.a$
$z(a) = c(a)$	$\lambda z_1.c(z_1);\ \lambda z_1.c(a)$
$z(c(a)) = a$	$\lambda z_1.a$
$z(c(a)) = c(a)$	$\lambda z_1.z_1;\ \lambda z_1.c(a)$

Then we get the following new terms in Q_1: $\{\lambda z_1.a, \lambda z_1 c(a), \lambda z_1.z_1, \lambda z_1.c(z_1)\}$.

In the second step, we closed under subterm and get the two additional terms: $\{z_1, c(z_1)\}$, which again lead to solve new second-order matching problems. We get then the new terms: $\{\lambda z_2.z_1, \lambda z_2.c(z_1)$. (All other solutions are already in Q_N up to α-conversion). Finally Q_N contains 10 elements

Remark: the size of Q_N may be doubly exponential in the size of t.

Theorem 16. *The solutions (resp. the solutions out of $\mathcal{C}_1$) of a fourth-order interpolation equation are recognized by an effectively computable tree automaton. (Whose size is at most doubly exponential w.r.t. the right hand side).*

Proof. The automaton $\mathcal{A}_{s_1,\ldots,s_n,t}$ is constructed as in the previous section, except that the set of states is now the set Q_N of lemma 15,using for N the maximal number of arguments of any type in $Sub(x)$, x itself excepted. Then the alphabet is larger: it should contain any symbol occurring in any state. The set of transition rules is constructed in the same way.

Then we use exactly the same proof as for theorem 11, replacing the reference to lemma 10 with a reference to 14. $\diamond$

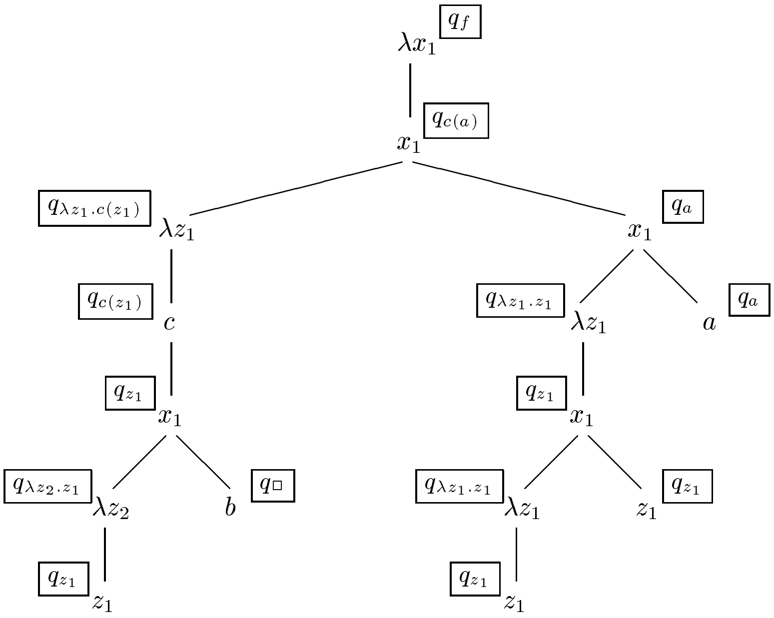
Example 4. Consider the fourth order interpolation equation:

$$x(\lambda y \lambda z.y(z)) = c(a)$$

The set of states has been computed in example 3. Let us now precise the set of transition rules:

$$\begin{array}{rcl@{\qquad}rcl}
a & \to & q_a & \Box & \to & q_\Box \\
c(q_a) & \to & q_{c(a)} & z_1 & \to & q_{z_1} \\
\lambda z_1.q_a & \to & q_{\lambda z_1.a} & \lambda z_1 q_{z_1} & \to & q_{\lambda z_1.z_1} \\
\lambda z_1.q_{c(a)} & \to & q_{\lambda z_1.c(a)} & c(q_{z_1}) & \to & q_{c(z_1)} \\
\lambda z_1 q_{c(z_1)} & \to & q_{\lambda z_1.c(z_1)} & \lambda z_2.q_{z_1} & \to & q_{\lambda z_2.z_1} \\
\lambda z_2.q_{c(z_1)} & \to & q_{\lambda z_2.c(z_1)} & \lambda x_1.q_{c(a)} & \to & q_f \\
x_1(q_{\lambda z_1.z_1}, q_a) & \to & q_a & x_1(q_{\lambda z_1.z_1}, q_{z_1}) & \to & q_{z_1} \\
x_1(q_{\lambda z_1.z_1}, q_{c(a)}) & \to & q_{c(a)} & x_1(q_{\lambda z_1.z_1}, q_{c(z_1)}) & \to & q_{c(z_1)} \\
x_1(q_{\lambda z_1.a}, q_\Box) & \to & q_a & x_1(q_{\lambda z_1.c(a)}, q_\Box) & \to & q_{c(a)} \\
x_1(q_{\lambda z_2.z_1}, q_\Box) & \to & q_{z_1} & x_1(q_{\lambda z_2.c(z_1)}, q_\Box) & \to & q_{c(z_1)} \\
x_1(q_{\lambda z_1.c(z_1)}, q_a) & \to & q_{c(a)} & x_1(q_{\lambda z_1.c(z_1)}, q_{z_1}) & \to & q_{c(z_1)}
\end{array}$$

For instance the following term is accepted by the automaton (the states that at reached at each node are indicated in a frame):



6 General third order matching

Theorem 17. *The set of solutions of a third order matching equations is recognizable and third-order matching is NP-complete*

Sketch of the proof:. We can prove the theorem in two ways: either we rely on Padovani's result relating the interpolation equations and the matching problem. Then the result can be obtained thanks to closure properties of automata and theorem 11. Or else, we can construct directly an automaton on tuples in the spirit of the proof of theorem 11. Let us sketch quickly this second construction, which will also yield a complexity result.

Consider the equation $s = t$. Initially the set $\mathcal{E}$ is empty. Then, we repeat the following steps until s does not contain any free variable:

1. let $x(s_1, \dots, s_n)$ be an inner occurrence of a free variable in s. Let $\mathcal{Z}$ be the (finite) set of free variables in $s_1, \dots, s_n$ (these variables are bound higher up in s).
2. Guess whether the result is significant or not: if not, then replace $x(s_1, \dots, s_n)$ with $\Box$ in s and go on to step 1. Otherwise guess the result r in the finite set $\mathcal{S}(\mathcal{Z}, t)$ (which is defined below.
3. Add $x(s_1, \dots, s_n) = r$ (solutions out of $\mathcal{Z}$) to $\mathcal{E}$
4. replace $x(s_1, \dots, s_n)$ with r in s.

Finally, the set of solutions is recognized by the union for all sets $\mathcal{E}$ computed as above and such that the last replacement of s yields t, of the automaton which accepts the solutions of the system $\mathcal{E}$ (with the restriction on the instances). Now, we precise $\mathcal{S}(\mathcal{Z}, t)$: it is the set of subterms of t in which some of the subterms have been replaced with variables in $\mathcal{Z}$.

Using theorem 8 and theorem 11, we get the first result: the set of solutions is recognizable.

The membership of third order matching to NP is a consequence of lemma 13: if the matching equation has a solution, then it is a solution of one of the above-computed systems $\mathcal{E}$.

NP-hardness is a consequence of the lemma 18, which shows that second-order matching is already NP-hard.[3]

$\diamond$

Example 5. Let us show an example of the decision procedure for third-order matching. Consider for instance the problem

$$x(\lambda z_1.x(\lambda z_2.z_1)) = c(a)$$

We guess first the result of $x(\lambda z_2.z_1)$: there are five possible outcome: $\square$ (irrelevant result), a, $c(a)$, z_1, $c(z_1)$. For each of them we have a system to solve:

$$\begin{array}{ll}
1. & x(\lambda z_1.\square) = c(a) \\
2. & x(\lambda z_1.a) = c(a) \wedge x(\lambda z_2.z_1) = a \\
3. & x(\lambda z_1.c(a)) = c(a) \wedge x(\lambda z_2.z_1) = c(a) \\
4. & x(\lambda z_1.z_1) = c(a) \wedge x(\lambda z_2.z_1) = z_1 \\
5. & x(\lambda z_1.c(z_1)) = c(a) \wedge x(\lambda z_2.z_1) = c(z_1)
\end{array}$$

Where z_1 is prohibited in the solution of the second equations. The first and third systems have only one solution: the constant function $\lambda x_1.c(a)$. The second system has no solution. The fourth system has infinitely solutions (which are recognized by the intersection automaton): $\lambda x_1.x_1^n(c(x_1^m(a)))$ with $n > 0$. The last system has no solution.

Lemma 18. *Second-order matching is NP-hard.*

Sketch of the proof: We encode 3-SAT as follows: we consider a first order variable x_P for each propositional variable P. In addition, we have for each clause one second-order variable x_C whose type is $o, o, o \to o$ and a binary constant symbol f (for convenience we write $f(u, f(v, w))$ as $f(u, v, w)$).

For each clause $C = P \vee Q \vee R$, let t_C be the term

$$f(x_C(0,0,0), x_C(1,1,1), x_C(x_P, x_Q, x_R), x_C(0,0,0)).$$

[3] We believe that this result is well-known, but we were unable to find a reference.

For each clause $\neg P \vee Q \vee R$, $t_C = f(x_C(0,0,0), x_C(1,1,1), x_C(1, x_Q, x_R), x_C(x_P, 0, 0))$ and symmetrically for the two other cases. Then the matching problem we are considering is

$$f(t_{C_1}, \ldots, t_{C_m}) = f(f(0,1,1,0), \ldots, f(0,1,1,0))$$

σ is a solution iff, for every clause C, $t_C\sigma\downarrow = f(0,1,1,0)$. Each of the equations $t_C = f(0,1,1,0)$ is equivalent to a system of four equations. The solutions for the first two equations: $x_C(0,0,0) = 0$ and $x_C(1,1,1) = 1$ are the three projections. The two last equations impose a truth value for the variables, depending on the projection; for instance $x_C(1, x_Q, x_R) = 1$ and $x_C(x_P, 0, 0) = 0$ has three solutions: $x = \pi_1 \wedge x_P = 0$, $x_C = \pi_2 \wedge x_Q = 1$, $x_C = \pi_3 \wedge x_R = 1$ which correspond to the three assignments satisfying the clause C. $\diamond$

7 General fourth order matching

We do not try here to derive the precise complexity of the problem.

Theorem 19. *The set of solutions of a fourth order matching problem is effectively recognizable. Hence fourth order matching is decidable.*

Sketch of the Proof It is not possible to proceed as in the last section because the set of possible results of $x(s_1, \ldots, s_n)$ is potentially infinite. However, it might be possible to guess a result, taking into account the actual arguments $s_1, \ldots, s_n$ (which we did not do for third order matching). We follow here Padovani's technique [8]: It is possible to compute a finite set of representatives for the third order *observational equivalence* relative to the right hand side t[4] Now, instead of guessing the result of $x(s_1, \ldots, s_n)$ starting with innermost occurrences, we start with outermost occurrences and guess the observational class of $s_1, \ldots, s_n$. Assume we guessed $v_1, \ldots, v_n$, then we replace each s_i with v_i in the matching equation and add the constraints $s_i \simeq_t v_i$ for every i. These last equations can be checked by enumerating all (second-order) representatives of the observational equivalence and solve $s_i(w_1, \ldots, w_k)\downarrow = v_i(w_1, \ldots, w_k)\downarrow$ (plus a system of disequalities, see Padovani's proof). Note that $v_i(w_1, \ldots, w_k)\downarrow$ may still contain occurrences of x at this stage: we have to iterate the processus. By the closure properties of tree automata, we get the recognizability of the set of solutions. $\diamond$

Instead of giving more details on the proof (which is essentially due to Padovani), we give below an example of how the procedure works:

[4] According to [8], $u \simeq_t v$ if u, v have the same type $\tau_1, \ldots, \tau_n \to o$ and, for every terms $w_1, \ldots, w_n$ either $u' = u(w_1, \ldots, w_n)\downarrow = v' = v(w_1, \ldots, w_n)\downarrow$ or else neither u' nor v' is a subterm of t. Padovani shows that the observational equivalence is a congruence of finite index. Computing a set of representatives for this equivalence at order n is equivalent to nth order matching.

Example 6. Let us consider the matching equation

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

where y has order 2. To solve (M), we first guess a representative v of the observationnal equivalence classes of order 3, and solve the system

$$(1)\begin{cases} x(v) = a \\ v \simeq_a \lambda z.x(\lambda y.z(a)) \end{cases}$$

We have first to determine representatives for the observational equivalence classes with respect to a at order 2 and 3. At order 2, this means solving the 4 following systems (b stands for any other constant) :

$$\begin{cases} y(a) = a \\ y(b) = a \end{cases} \begin{cases} y(a) = a \\ y(b) \neq a \end{cases} \begin{cases} y(a) \neq a \\ y(b) = a \end{cases} \begin{cases} y(a) \neq a \\ y(b) \neq a \end{cases}$$

Representatives of the solutions are respectively :

$$s_1^2 = \lambda y.a, s_2^2 = \lambda y.y, \emptyset, s_3^2 = \lambda y.b$$

We compute then the observational equivalence representatives at order 3:

$$\begin{cases} y(s_1^2) = a \\ y(s_2^2) = a \\ y(s_3^2) = a \end{cases} \begin{cases} y(s_1^2) = a \\ y(s_2^2) = a \\ y(s_3^2) \neq a \end{cases} \begin{cases} y(s_1^2) \neq a \\ y(s_2^2) = a \\ y(s_3^2) = a \end{cases} \begin{cases} y(s_1^2) \neq a \\ y(s_2^2) = a \\ y(s_3^2) \neq a \end{cases}$$

$$\begin{cases} y(s_1^2) = a \\ y(s_2^2) \neq a \\ y(s_3^2) = a \end{cases} \begin{cases} y(s_1^2) = a \\ y(s_2^2) \neq a \\ y(s_3^2) \neq a \end{cases} \begin{cases} y(s_1^2) \neq a \\ y(s_2^2) \neq a \\ y(s_3^2) = a \end{cases} \begin{cases} y(s_1^2) \neq a \\ y(s_2^2) \neq a \\ y(s_3^2) \neq a \end{cases}$$

Corresponding representatives are :

$$s_1^3 = \lambda y.a, s_2^3 = \lambda y.y(a), \emptyset, \emptyset, \emptyset, s_3^3 = \lambda y.y(b), \emptyset, s_4^3 = \lambda y^2.b$$

Then, back to solve (M), we guess a representative v of order 3, say $v = \lambda z.z(a)$, then (1) becomes, after β-reduction :

$$\begin{cases} x(v) = a \\ x(\lambda y.a) = a \\ x(\lambda y.a) = a \\ x(\lambda y.b) \neq a \end{cases}$$

We can compute the solutions, using the automata techniques of the previous section and we get:

$$\lambda f.f(\lambda t_1.f(\lambda t_2 \ldots f(\lambda t_n.u)))$$

Where all t_i are identical, except possibly for one index i_0, and $u \in \{a, t_{i_0}\}$.

Note however that the Padovani's procedure which is sketched in the previous theorem is very expensive. Our automaton for a fourth order interpolation equation was already doubly exponential. We have at least to guess representatives for the third order observational equivalence and to solve a system of triply exponential size. It is an open question whether such a procedure can be improved, as we did for third-order matching.

8 Fifth order matching and beyond

What about a possible extension to higher order matching ? Of course, we tried, but without success, to extend our techniques. Let us describe the problems which arise at order 5.

If we want to follow the same technique, we have to replace lemma 14 (the only order-dependent part of the proof) with another result which works at order 5. However, we would have to replace solutions of second order interpolation equations with solutions of third order matching equations. In contrast with second order matching which admits a finite basis of solutions, third order matching is not finitary. Hence the set of states, if computed as a straightforward extension of our construction, would be infinite.

The next idea is to put in the states *representatives* of the third order observational equivalence, instead of all solutions. This would keep the set of states finite. Equivalently, we could consider the states of an automaton accepting the solutions of derived third order interpolation problems instead of trying to compute the set of solutions in extension. There are however additional problems with free variables. We can derive from theorem 16 that every solution of a fourth-order matching problem can be expressed (via α-conversion) using a bounded number of variables (which only depend on the matching problem). These variables are part of the alphabet. This is no longer possible at fifth order, as shown by the simple example:

$$x(\lambda y \lambda z.y(\lambda z'.z(z'))) = a$$

A solution of this problem is the term

$$\lambda x_1.x_1(\lambda y_1.x_1(\ldots x_1(\lambda y_n.y_{i_1}(y_{i_2}(\ldots(y_{i_k}(a))\ldots)), u_n)\ldots, u_2), u_1)$$

where, for some $m \leq k$, every u_{i_j} $(j \leq m)$ is the identity and $u_{i_{m+1}}$ is the constant function a. k is arbitrarily large, hence we cannot use a bounded number of variables. Still, there is a way to overcome this difficulty: we may say that y_i is equivalent to y_j if $u_i \simeq_t u_j$. Then there are finitely many classes of variables, which all behave the same way. This yields a finite tree automaton accepting the set of solutions (for this example). We were unable to generalize this idea of equivalence classes of variables. Though, we did not find any higher-order matching problem whose set of solutions is not recognizable.

We conjecture that the set of solutions is always recognizable (basically because the observational equivalence is of finite index). Even if this conjecture is true, it remains to effectively compute the automaton accepting the solutions.

Acknowledgements

We thank several researchers with whom we had fruitful discussions, among them Gilles Dowek, Sophie Malecki, Tobias Nipkow, Vincent Padovani.

References

1. G. Dowek. A complete proof synthesis method for the cube of type systems. 3(3):287–315, 1993.
2. G. Dowek. Third order matching is decidable. *Annals of Pure and Applied Logic*, 1993.
3. G. Dowek. The undecidability of pattern matching in calculi where primitive recursive functions are representable. *Theoretical Comput. Sci.*, 107:349–356, 1993.
4. M. Gécseg and M. Steinby. *Tree Automata.* Akademia Kiadó, Budapest, 1984.
5. W. D. Goldfarb. Note on the undecidability of the second-order unification problem. *Theoretical Comput. Sci.*, 13:225–230, 1981.
6. R. Hindley and J. Seldin. *Introduction to Combinators and λ-calculus.* Cambridge University Press, 1986.
7. G. Huet. *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots \omega$. Thèse d'Etat, Univ. Paris 7, 1976.
8. V. Padovani. *Filtrage d'ordre supérieur.* PhD thesis, Université de Paris VII, 1996.
9. A. Schubert. Linear interpolation for the higher order matching problem. In *Proc. CAAP'97*, Lille, 1997.
10. H. Seidl. Haskell overloading is DEXPTIME-complete. *Inf. Process. Lett.*, 52:57–60, 1994.
11. J. Springintveld. Third-order matching in the polymorphic lambda calculus. In G. Dowek, J. Heering, K. Meinke, and B. Möller, editors, *Proc. Higher-Order Algebra, Logic and Term Rewriting*, volume 1074 of *LNCS*, Paderborn, Germany, September 1995. Springer-Verlag.
12. J. Springintveld. Third-order matching in the presence of type constructors. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proc. 2nd Int. Conf. on Typed Lambda Calculi and Applications*, volume 902 of *LNCS*, Edinburgh, United Kingdom, April 1995. Springer-Verlag.
13. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 134–191. Elsevier, 1990.

This article was processed using the LaTeX macro package with LLNCS style