Higher-Order Matching, Games and Automata

Colin Stirling
School of Informatics
University of Edinburgh
cps@inf.ed.ac.uk

Abstract

Higher-order matching is the problem given $t = u$ where t, u are terms of simply typed λ-calculus and u is closed, is there a substitution θ such that $t\theta$ and u have the same normal form with respect to $\beta\eta$-equality: can t be pattern matched to u? This paper considers the question: can we characterize the set of all solution terms to a matching problem? We provide an automata-theoretic account that is relative to resource: given a matching problem and a finite set of variables and constants, the (possibly infinite) set of terms that are built from those components and that solve the problem is regular. The characterization uses standard bottom-up tree automata.

1. Introduction

Higher-order matching is the problem given $t = u$ where t, u are terms of simply typed λ-calculus and u is closed, is there a substitution θ such that $t\theta$ and u have the same normal form with respect to $\beta\eta$-equality: can t be pattern matched to u? The problem was conjectured to be decidable by Huet [5]. Loader showed that it is undecidable when β-equality is the same normal form by encoding the undecidable problem of λ-definability as matching [8]: also see [6] for a proof that using the halting problem.

In previous work, we confirm Huet's conjecture [17]: a full (and very complicated) proof is in the long version of [17] which is not yet published. It first appeals to Padovani's and Schubert's reduction of matching to the conceptually simpler (dual) interpolation problem [12, 10]. It is then inspired by model-checking games (such as in [15]) where a model, a transition graph, is traversed relative to a property and players make choices at appropriate positions. We define a game where the model is a closed λ-term t and play moves around it relative to a (dual) interpolation problem P. The game captures the dynamics of β-reduction on t without changing it (using substitution). Unlike standard model-checking games, play may arbitrarily jump around a term because of binding. The principal virtue of the game is that small pieces of a solution term can be understood in terms of their subplays and how they, thereby, contribute to solving the problem P. Simple transformations on terms are defined and combinatorial properties shown. Decidability of matching follows from the *small model property*: if there is a solution to a problem then there is a small solution to it. The proof of this property uses "unfolding" a λ-term with respect to game playing, analogous to unravelling a transition system in modal logic, followed by refolding.

In this paper our interest is with a different, although related, question: can we independently characterize the set of *all* solution terms to an interpolation problem? Part of the hope is that this may lead to a simpler proof of decidability of matching. Again, we start with the term checking game. However, we slightly reformulate it and show that it underpins an automata-theoretic characterization relative to *resource*: given a problem P, a finite set of variables and constants the (possibly infinite) set of terms that are built from those components and that solve P is regular. The characterization uses standard bottom-up tree automata. The states of the automaton are built from abstractions of sequences of moves in the game. The automaton construction works for all orders. Comon and Jurski define tree automata that characterize all solutions to a 4th-order problem [2]. The states of their automata appeal to Padovani's observational equivalence classes of terms [10]. To define the states of their automata at higher-orders, one would need to solve the problem of how to quotient the potentially infinite set of terms into their respective finite observational equivalence classes: however, as Padovani shows this problem is, in fact, equivalent to the matching problem itself. Ong shows decidability of monadic second-order logic of the tree generated by an arbitrary higher-order scheme [9]. The proof uses a game-semantic characterization of a scheme as an infinite λ-term. A property, expressed as an alternating parity tree automaton, of the tree has to be transferred to the infinite term. A key ingredient of the transition from game to automaton is Ong's abstraction "variable profile" that captures a sequence of back-and-forth play jumping in a term which is also central here.

In Section 2 we describe matching and how it reduces to the interpolation problem. In Section 3 we introduce tree automata and how they may be used to define well-formed λ-terms. The interpolation game is defined in Section 4 and the automata-theoretic account with the main result is presented in Section 5. Finally, there are concluding comments in Section 6.

2 Matching and interpolation

Assume simple types that are generated from a single base type $\mathbf{0}$ using the binary $\rightarrow$ operator. A type is $\mathbf{0}$ or $A \rightarrow B$ where A and B are types. If $A \neq \mathbf{0}$ then it has the form $A_1 \rightarrow \ldots \rightarrow A_n \rightarrow \mathbf{0}$, assuming $\rightarrow$ associates to the right, written $(A_1, \ldots, A_n, \mathbf{0})$ following Ong [9]. A standard definition of *order* is: the order of $\mathbf{0}$ is 1 and the order of $(A_1, \ldots, A_n, \mathbf{0})$ is $k+1$ where k is the maximum of the orders of the A_is.

Terms of the simply typed λ-calculus are built from a countable set of typed variables $x, y, \ldots$ and constants $a, f, \ldots$ (each variable and constant has a unique type). The smallest set T of simply typed terms is: if x (f) has type A then $x : A \in T$ $(f : A \in T)$; if $t : B \in T$ and $x : A \in T$ then $\lambda x.t : A \rightarrow B \in T$; if $t : A \rightarrow B \in T$ and $u : A \in T$ then $tu : B \in T$. The *order* of a typed term is the order of its type. A typed term is *closed* if it does not contain free variables. Throughout, we assume the definitions of α-equivalence, β and η-reduction.

Definition 1. *A matching problem is an equation $v = u$ where $v, u : \mathbf{0}$ and u is closed. The order of the problem is the maximum of the orders of the free variables $x_1, \ldots, x_n$ in v. A solution is a sequence of (closed) terms $t_1, \ldots, t_n$ such that each t_i has the same type as x_i and $v\{t_1/x_1, \ldots, t_n/x_n\} =_{\beta\eta} u$.*

Given a matching problem $v = u$, the decision question is whether it has a solution: can v be pattern matched to u?

In the following we assume that all terms in *normal form* are in *η-long form*: if $t : \mathbf{0}$ then t is $u : \mathbf{0}$ where u is a constant or a variable, or $u\, t_1 \ldots t_k$ where $u : (B_1, \ldots, B_k, \mathbf{0})$ is a constant or a variable and each $t_i : B_i$ is in η-long form; if $t : (A_1, \ldots, A_n, \mathbf{0})$ then t is $\lambda y_1 \ldots y_n.t'$ where each $y_i : A_i$ and $t' : \mathbf{0}$ is in η-long form. Throughout, we write $\lambda z_1 \ldots z_n$ for $\lambda z_1 \ldots \lambda z_n$. A term is *well-named* if each occurrence of a variable y within a λ-abstraction is unique. Because of normal forms, β-equality and $\beta\eta$-equality coincide.

Definition 2. *Assume $u : \mathbf{0}$ and each $v_i : A_i$, $1 \leq i \leq n$, is a closed term in normal form and $x : (A_1, \ldots, A_n, \mathbf{0})$. An interpolation problem P has the form $xv_1 \ldots v_n = u$. The type of problem P is that of x and the order of P is the order of x. A solution of P of type A is a closed term $t : A$ in normal form such that $tv_1 \ldots v_n =_\beta u$. We write $t \models P$ if t is a solution of P.*

Conceptually, interpolation is simpler than matching because of its single variable. If $v = u$ is a matching problem with free variables $x_1 : A_1, \ldots, x_n : A_n$ where v and u are in normal form, then its *associated* interpolation problem is $x(\lambda x_1 \ldots x_n.v) = u$ where $x : ((A_1, \ldots, A_n, \mathbf{0}), \mathbf{0})$. This appears to raise order by 2 as with the reduction of matching to pairs of interpolation equations in Schubert [12]. However, we only need to consider potential solution terms (in normal form with the right type) $\lambda z.zt_1 \ldots t_n$ where each $t_i : A_i$ is closed and so cannot contain z: we say that such terms are *canonical*. Padovani provides a reduction of matching to *dual interpolation* that exactly preserves order [10]. A dual interpolation problem consists of a finite family of interpolation equations $xv_1^i \ldots v_n^i = u_i$ and a finite family of disequations $xv_1^j \ldots v_n^j \neq u_j$, all with the same free variable x: a solution is a term t in normal form such that for each equation $tv_1^i \ldots v_n^i =_\beta u_i$ and for each disequation $tv_1^j \ldots v_n^j \neq_\beta u_j$. Dual interpolation underpins a notion of observational quivalence on terms that is used by Padovani to solve 4th-order matching [10].

Fact 1. *A matching problem has a solution iff its associated interpolation problem has a canonical solution.*

Consequently, the higher-order matching problem is equivalent to the decision question: given an associated interpolation problem P, is there a canonical term $t \models P$? In the following we, therefore, concentrate on solutions to interpolation problems.

Example 1. *$x_1(\lambda z.x_1(\lambda z'.za)) = a$ from [2] is a 4th-order matching problem where $z, z' : (\mathbf{0}, \mathbf{0})$ and $x_1 : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$. Its associated interpolation problem is $x(\lambda x_1.x_1(\lambda z.x_1(\lambda z'.za))) = a$ with $x : (((((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0}), \mathbf{0}), \mathbf{0})$. A canonical solution has the form $\lambda x.x(\lambda y.y(\lambda y_1^1 \ldots y(\lambda y_1^k.w) \ldots))$ where w is the constant a or one of the variables y_1^j, $1 \leq j \leq k$.*

Example 2. *$x(\lambda y_1 y_2.y_1)(\lambda y_3.fy_3y_3) = faa$ from [2] is an interpolation problem of order 3 with $x : ((\mathbf{0}, \mathbf{0}, \mathbf{0}), (\mathbf{0}, \mathbf{0}), \mathbf{0})$ and each $y_i : \mathbf{0}$.*

Example 3. *$x(\lambda z.z) = w$ where $w = f(\lambda x_1' x_2' x_3'.x_1'(x_3'))a$ also has order 3 where x has type $((\mathbf{0}, \mathbf{0}), \mathbf{0})$ and $f : (((\mathbf{0}, \mathbf{0}), \mathbf{0}, \mathbf{0}, \mathbf{0}), \mathbf{0}, \mathbf{0})$ assuming $x_2' : \mathbf{0}$.*

Example 4. *$x(\lambda y_1 y_2.y_1(\lambda y_3.y_2(y_1(\lambda y_4.y_3)))) = u$ where $u = h(g(h(ha)))$, due to Luke Ong, is 5th-order with $x : ((((\mathbf{0}, \mathbf{0}), \mathbf{0}), (\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$.*

Example 3 illustrates that a right term u of an interpolation problem may contain bound variables. Let $X' =$

$\{x'_1,\ldots,x'_k\}$ be the set of bound variables in u and let $C = \{c_1,\ldots,c_k\}$ be a fresh set of constants where each c_i has the same type as x'_i.

Definition 3. *The ground closure of a closed term w, whose bound variables belong to X', with respect to C, written $Cl(w,X',C)$, is defined inductively as follows: if $w = a : \mathbf{0}$ then $Cl(w,X',C) = \{a\}$; if $w = fw_1\ldots w_n$ then $Cl(w,X',C) = \{w\} \cup \bigcup \mathrm{Cl}(w_i,X',C)$; if $w = \lambda x'_{j_1}\ldots x'_{j_n}.u$ then $Cl(w,X',C) = \mathrm{Cl}(u\{c_{j_1}/x'_{j_1},\ldots,c_{j_n}/x'_{j_n}\},X',C)$ where $x'_{j_i} \in X'$ and $c_{j_i} \in C$ is the corresponding constant.*

The ground closure of w of Example 3 with respect to $\{c_1,c_2,c_3\}$ is $\{w, c_1(c_3), c_3, a\}$. The ground closure of u of Example 4 with respect to the empty set is its subterms $\{u, g(h(h(a))), h(h(a)), h(a), a\}$.

Definition 4. *Given the problem P with equation $xv_1\ldots v_n = u$, let X'_P be the (possibly empty) set of bound variables in u and C_P be a corresponding set of new constants (that do not occur in P), the forbidden constants. The right subterms are $\mathsf{R}_P = Cl(u,X'_P,C_P)$.*

We are interested in the set of closed terms t in normal form such that $t \models P$ and t does not contain forbidden constants (belonging to C_P).

In the literature there are slight variant definitions of matching. Statman describes the problem as a range problem [14]: given $v : (A_1,\ldots,A_n,B)$ and $u : B$ where both u and v are closed, are there terms $t_1 : A_1,\ldots,t_n : A_n$ such that $vt_1\ldots t_n =_{\beta\eta} u$? If $B = (A_1,\ldots,A_m,\mathbf{0})$ is of higher type then u in normal form is $\lambda x'_1\ldots x'_m.w$. Therefore, we can consider the matching problem $(vx_1\ldots x_n)c_1\ldots c_m = w\{c_1/x'_1,\ldots,c_m/x'_m\}$ where the c_i's are forbidden constants (and cannot occur in a solution term). In [10] a matching problem is a family of equations $v_1 = u_1,\ldots,v_m = u_m$ to be solved uniformly: therefore, they reduce to a single equation $fv_1\ldots v_m = fu_1\ldots u_m$ where f is a constant of the appropriate type.

3 Interpolation trees and tree automata

Given a potential solution term t in normal form to the interpolation problem P, $xv_1\ldots v_n = u$, there is the tree in Figure 1. If $x : (A_1,\ldots,A_n,\mathbf{0})$ then the explicit application operator $@ : ((A_1,\ldots,A_n,\mathbf{0}),A_1,\ldots,A_n,\mathbf{0})$ has its expected meaning: $@tv_1\ldots v_n = tv_1\ldots v_n$. The terms t and v_j, $1 \le j \le n$, do not contain forbidden constants. These terms t' are represented as labelled trees, tree(t'). If t' is $y : \mathbf{0}$ or $a : \mathbf{0}$ then tree(t') is the single node labelled with t'. In the case of $uw_1\ldots w_k$ when u is a variable, a constant or the initial @, we assume that a dummy λ with the empty sequence of variables is placed before any subterm

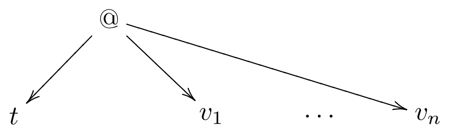

Figure 1. An interpolation tree

$w_i : \mathbf{0}$ in its tree representation. With this understanding, if t' is $uw_1\ldots w_k$ then tree(t') consists of the root node labelled u and k-successor nodes labelled with tree(w_i). We use the notation $u \downarrow_i t'$ to represent that tree t' is the ith successor of the node u. If t' is $\lambda\overline{y}.v$, where $\overline{y}$ is possibly the empty sequence of variables, then tree(t') consists of the root node labelled $\lambda\overline{y}$ and a single successor node tree(v), $\lambda\overline{y} \downarrow_1$ tree(v). In the following we use t' to be the λ-term t', its λ-tree or the label (a constant, variable, initial @ or $\lambda\overline{y}$) at its root node. For instance, the tree of the equation of Example 2 with solution term $\lambda x_1x_2.x_1(x_2(x_1(x_1ab)b))b$ and without explicit indices on edges is depicted in Figure 2: for instance, (2) $\downarrow_1$ (3) and (2) $\downarrow_2$ (15).

Given problem P, $xv_1\ldots v_n = u$, ideally we would like there to be a *tree automaton* that accepts exactly all instances of t in Figure 1 that solve P: that is, we want an automaton that accepts $@tv_1\ldots v_n$ if, and only if, $t \models P$.

Definition 5. *Assume Σ is a finite graded alphabet where each element $s \in \Sigma$ has an arity $ar(s) \ge 0$. A Σ-*tree *is a finite tree where each node is labelled with an element of Σ. If node n in t is labelled with s and $ar(s) = k$ then n has precisely k successors in t. Let T_Σ be the set of Σ-trees.*

Definition 6. *A Σ-tree automaton $\mathsf{A} = (Q,\Sigma,F,\Delta)$ where Q is a finite set of states, Σ is the finite alphabet, $F \subseteq Q$ is the set of final states and Δ is a finite set of transition rules $sq_1\ldots q_k \Rightarrow q$ where $k \ge 0$, $s \in \Sigma$, $ar(s) = k$ and $q_1,\ldots,q_k,q \in Q$.*

Definition 7. *A run of $\mathsf{A} = (Q,\Sigma,F,\Delta)$ on $t \in \mathsf{T}_\Sigma$ is a labelling of t with elements of Q that is defined bottom-up, starting from the leaves of t. If a node n of t is labelled $s \in \Sigma$, $ar(s) = k \ge 0$, $sq_1\ldots q_k \Rightarrow q \in \Delta$ and $q_1\ldots q_k$ label the k successors of s (in correct order) then n is labelled q. A accepts the Σ-tree t iff there is a run of A on t such that the root node of t is labelled with a final state, $q \in F$. Let $\mathsf{T}_\Sigma(\mathsf{A})$ be the set of Σ-trees accepted by A.*

A Σ-tree automaton A involves a finite set of states Q and transitions Δ (which can be nondeterministic). A run of A on the Σ-tree t is a labelling of it with elements of Q that starts from the leaves. If n in t is a leaf then it is labelled with an $s \in \Sigma$ where ar$(s) = 0$: Δ may contain a transition of the form $s \Rightarrow q$, so n can be labelled with q. States may then percolate through t using Δ: If n is labelled s and $sq_1\ldots q_k \Rightarrow q \in \Delta$ and $q_1\ldots q_k$ label the

$$\begin{tikzcd}
 & @ \arrow[dl] \arrow[d] \arrow[dr] & & \\
(1)\lambda x_1 x_2 \arrow[d] & (17)\lambda y_1 y_2 \arrow[d] & (19)\lambda y_3 \arrow[d] & \\
(2)x_1 \arrow[d] \arrow[dr] & (18)y_1 & (20)f \arrow[d] \arrow[dr] & \\
(3)\lambda \arrow[d] & (15)\lambda \arrow[d] & (21)\lambda \arrow[d] & (23)\lambda \arrow[d] \\
(4)x_2 \arrow[d] & (16)b & (22)y_3 & (24)y_3 \\
(5)\lambda \arrow[d] & & & \\
(6)x_1 \arrow[d] \arrow[dr] & & & \\
(7)\lambda \arrow[d] & (13)\lambda \arrow[d] & & \\
(8)x_1 \arrow[d] \arrow[dr] & (14)b & & \\
(9)\lambda \arrow[d] & (11)\lambda \arrow[d] & & \\
(10)a & (12)b & &
\end{tikzcd}$$

Figure 2. Example 2 with solution term

k successors of n (in correct order) then n can be labelled q. This is repeated until no further nodes can be labelled. A run is then accepting if the root of t is labelled with a final state, $q \in F$.

We state some pertinent properties of tree automata [1].

Proposition 1. *The emptiness problem, given a Σ-tree automaton A is $\mathsf{T}_\Sigma(\mathsf{A}) = \emptyset$?, is decidable in linear time. For each Σ-tree automaton A there is a $\overline{\mathsf{A}}$ such that $\mathsf{T}_\Sigma(\overline{\mathsf{A}}) = \mathsf{T}_\Sigma - \mathsf{T}_\Sigma(\mathsf{A})$. For Σ-tree automata A_1, A_2 there is an A with $\mathsf{T}_\Sigma(\mathsf{A}) = \mathsf{T}_\Sigma(\mathsf{A}_1) \cap \mathsf{T}_\Sigma(\mathsf{A}_2)$.*

Consider a potential solution term t to Example 2 such as the term rooted at (1) in Figure 2. Up to α-equivalence, t is $\lambda x_1 x_2.w$ with $x_1 : (\mathbf{0}, \mathbf{0}, \mathbf{0})$ and $x_2 : (\mathbf{0}, \mathbf{0})$. What are its possible components (represented as a tree with dummy lambdas)? First there can be occurrences of x_1 with arity 2 and x_2 with arity 1. There can also be constant occurrences $a : \mathbf{0}$, of arity 0, f of arity 2 and other constants. It suffices just to allow one other constant $b : \mathbf{0}$ (because constants other than f and a cannot contribute to a solution of this interpolation problem[1]). Therefore, any possible $\lambda x_1 x_2.w$ (with this restriction on constant occurrences) as a tree belongs to T_Σ when $\Sigma = \{\lambda x_1 x_2, x_1, x_2, \lambda, f, a, b\}$.

There are Σ-trees, such as $x_1(\lambda x_1 x_2.x_2)a$, that are not the trees of well-formed terms in T. However, it is straightforward to define a tree automaton that accepts exactly the well-formed typed λ-terms (trees) over Σ of the form $\lambda x_1 x_2.w$. The idea is to internalize the finite alphabet Σ as states. The automaton A is $Q = \{[s] \mid s \in \Sigma\}$, $F = \{[\lambda x_1 x_2]\}$ and Δ consists of $a \Rightarrow [a]$, $b \Rightarrow [b]$, $\lambda\,[s] \Rightarrow [\lambda]$ for any $s \in \Sigma - \{\lambda, \lambda x_1 x_2\}$, $\lambda x_1 x_2\,[s] \Rightarrow [\lambda x_1 x_2]$ for any $s \in Q - \{\lambda, \lambda x_1 x_2\}$, $x_1[\lambda][\lambda] \Rightarrow [x_1]$, $x_2[\lambda] \Rightarrow [x_2]$, $f[\lambda][\lambda] \Rightarrow [f]$. The automaton could be extended to recognise all well-formed instances of the interpolation equation, such as Figure 2, by adding the elements @, $\lambda y_1 y_2$, y_1, λy_3, y_3 to Σ.

The 5th-order equation in Example 4 has a solution term in Figure 3. A potential solution has the form $\lambda z.w$ with $z : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), (\mathbf{0}, \mathbf{0}), \mathbf{0})$. Consider its possible components. For constants, we have a, b, g and h (the latter two of arity 1). A z component has two successors λx_1^j, λx_2^j where $x_1^j : (\mathbf{0}, \mathbf{0})$ and $x_2^j : \mathbf{0}$ which may bind variable occurrences x_1^j with arity 1 and x_2^j with arity 0. The problem is that the set of well-formed terms requires an infinite alphabet Σ (even up to α-equivalence) because of the family $\lambda z.z(\lambda x_1^1.z(\lambda x_1^2.\ldots z(\lambda x_1^k.x_1^1(x_1^2 \ldots x_1^k(u_0)))u_k)\ldots)u_1$ of terms as k increases: this is also described in [2].

If we cannot define a tree automaton that accepts the well-formed terms of type A of order 5 and above (modulo the restriction on constants) then it is extremely unlikely

[1]This can be exactly formalised using the game-theoretic characterisation of interpolation of Section 4.

that we could define an automaton that accepts just the solutions to an interpolation problem of type A. Although there is active research extending automata on words and trees to infinite alphabets which preserve "good" properties, such as decidability of non-emptiness, see [11] for a recent survey, the results do not apply to the case caused by higher-order binding.

Therefore, we restrict the overall goal of seeking an automaton that can accept the solutions to a problem P. First, it is clear that given a type A and the finite alphabet Σ, there is a tree automaton that accepts the well-formed Σ-terms. Similarly for matching, we take as input not only P but also the finite alphabet Σ. The main result, Theorem 2, is that there is a tree automaton that accepts exactly the set of well-formed Σ-terms that solve P. Therefore, relative to a finite alphabet Σ, the set of solutions to a problem P is regular.

Example 5. *In the case of Example 4 we can define a tree automaton that recognises the set of well-formed terms (trees) of the form $\lambda z.w$ when Σ is the set $\{\lambda z, \lambda z_1, \lambda z_2, \lambda x_1, \lambda x_2, \lambda, z, z_1, z_2, x_1, x_2, a, b, g, h\}$ with arities as in Figure 3 and Σ_1 is $\{z, z_1, z_2, x_1, x_2\}$. The idea is again to internalize the syntax and also to include a finite memory as to which binders are still outstanding. $Q = \{[s, S] \mid s \in \Sigma, S \subseteq \Sigma_1\}$, $F = \{[\lambda z, \emptyset]\}$ and Δ contains rules such as: $x_2 \Rightarrow [x_2, \{x_2\}]$ (representing that having seen x_2 there is an outstanding λx_2 binder), $z\,[\lambda z_1, S_1]\,[\lambda z_2, S_2] \Rightarrow [z, S_1 \cup S_2 \cup \{z\}]$, $\lambda x_2[s, S] \Rightarrow [\lambda x_2, S - \{x_2\}]$ for $s \in \{z, z_1, z_2, x_1, x_2, a, b, g, h\}$. Again, we can extend the automaton to accept all well-formed instances of the interpolation problem in Figure 3 by extending Σ.*

Comon and Jurski define tree automata that characterize all solutions to a 4th-order problem [2]. In Section 8 of their paper they make it clear that there are two problems with extending their automata beyond the 4th-order case. The first is that the states of the automaton are constructed out of observational equivalence classes of terms due to Padovani [10]. Up to a 4th-order problem, one only needs to consider finitely many terms. With 5th and higher orders, this is no longer true and one needs to quotient the potentially infinite terms into their respective observational equivalence classes in order to define only finitely many states: however as Padovani shows this procedure is, in fact, equivalent to the matching problem itself [10]. The second problem they mention and illustrate with an example is that fifth-order terms may (essentially) contain infinitely many different variables as discussed here. We are able to overcome the first problem but not the second. Our solution is indirect and proceeds via a game-theoretic characterisation of matching, Theorem 1. It underlies a more elementary understanding of β-reduction in terms of families of sequences of moves which can then be abstracted into structures that

$$\begin{tikzcd}
 & & @ \arrow[dl] \arrow[dr] & \\
 & (1)\lambda z \arrow[d] & & (21)\lambda y_1 y_2 \arrow[d] \\
 & (2)z \arrow[dl] \arrow[dr] & & (22)y_1 \arrow[d] \\
3)\lambda z_1 \arrow[d] & & (17)\lambda z_2 \arrow[d] & (23)\lambda y_3 \arrow[d] \\
(4)z \arrow[d] \arrow[dr] & & (18)h \arrow[d] & (24)y_2 \arrow[d] \\
(5)\lambda x_1 \arrow[d] & (13)\lambda x_2 \arrow[d] & (19)\lambda \arrow[d] & (25)\lambda \arrow[d] \\
(6)z_1 \arrow[d] & (14)g \arrow[d] & (20)z_2 & (26)y_1 \arrow[d] \\
(7)\lambda \arrow[d] & (15)\lambda \arrow[d] & & (27)\lambda y_4 \arrow[d] \\
(8)x_1 \arrow[d] & (16)x_2 & & (28)y_3 \\
(9)\lambda \arrow[d] & & & \\
(10)z_1 \arrow[d] & & & \\
(11)\lambda \arrow[d] & & & \\
(12)a & & &
\end{tikzcd}$$

Figure 3. A 5th order example

will become states of an automaton.

4 Interpolation games

We are going to define a game on interpolation trees such as Figure 1 which is now the *arena*. Games have been a key ingredient in our previous work on matching [16, 17]. The aim is to account for the dynamics, β-reduction, without changing terms by using substitution. It turns out that it could also be described using game-semantics following Ong [9]. Indeed, we slightly reformulate previous versions of the game so that it is closer to game-semantics. However, as in [17], we avoid questions, answers and their justification pointers by appealing to look-up tables.

Assume P is the problem $xv_1 \ldots v_n = u$, t is a potential solution term and assume the sets $\mathsf{R} = \mathsf{R}_P$ and $C = C_P$. We define the game $\mathsf{G}(t, P)$ played by one participant, player $\forall$, the *refuter* who attempts to show that t is not a solution of P.

Definition 8. *N is the set of nodes of the interpolation tree $@\, tv_1 \ldots v_n$, S is the set $\{[\, x\,] : x \in \mathsf{R} \cup \{\forall, \exists\}\}$ of game-states. $[\,\forall\,]$ and $[\,\exists\,]$ are the final game-states. V is the set of variables that occur in t and in $v_1, \ldots, v_n$. For each $i \geq 1$, the set of look-up tables Θ_i is iteratively defined: $\Theta_1 = \{\theta_1\}$ where $\theta_1 = \emptyset$ and Θ_{i+1} is the set of partial maps $V \to (N \times \bigcup_{j \leq i} \Theta_j) \cup C$.*

Definition 9. *A play of $\mathsf{G}(t, P)$ is a finite sequence of positions $t_1 q_1 \theta_1, \ldots, t_n q_n \theta_n$ where each $t_i \in N$, each $q_i \in S$ and q_n is final and each $\theta_i \in \Theta_i$ is a look-up table. For the initial position $t_1 = @$ where $@$ is the root of the interpolation tree, $q_1 = [\, u\,]$ where u is the right term of P and θ_1 is the empty look-up table. Player $\forall$ loses the play if the final state is $[\,\exists\,]$, otherwise she wins the play.*

The central feature of a play of $\mathsf{G}(t, P)$ is that control jumps from nodes of t to nodes of the v_j's and back again repeatedly[2]. The game appeals to a finite set of states S comprising goal states $[\, r\,]$, $r \in \mathsf{R}$, and *final* states, $[\,\forall\,]$, winning for the refuter, and $[\,\exists\,]$, losing for the refuter. As play proceeds by moving about t and the v_j's of Figure 1 one needs an account of the meaning of the current free variables. A free variable in a subtree of t is either associated with a subtree of a v_j or with a forbidden constant. Similarly, a free variable in a subtree of v_j is associated with a subtree of t or with a forbidden constant. So, the game appeals to the look-up tables $\theta_k \in \Theta_k$, $k \geq 1$: θ_k is a partial map from variables to forbidden constants C or to pairs

[2]The presentation of the game in [17] starts from the assumption that only t is the common structure for a dual interpolation problem P that may involve multiple equations and disequations. So, there play is always in t. Jumping in and out of the respective v_j's is coded in the states, as play traverses t.

A. $t_m = @ : A$, $A = ((A_1, \ldots, A_n, \mathbf{0}), A_1, \ldots, A_n, \mathbf{0})$, for $i : 1 \leq i \leq n+1$, $t_m \downarrow_i t'_i$ and $t'_1 = \lambda x_1 \ldots x_n$. Then $t_{m+1} = t'_1$, $q_{m+1} = q_m$ and $\theta_{m+1} = \theta_m\{t'_2\theta_m/x_1, \ldots, t'_{n+1}\theta_m/x_n\}$.

B. $t_m = \lambda \overline{y}$ and $t_m \downarrow_1 t'$. Then $t_{m+1} = t'$, $q_{m+1} = q_m$ and $\theta_{m+1} = \theta_m$.

C. $t_m = a : \mathbf{0}$. Then $t_{m+1} = t_m$ and $\theta_{m+1} = \theta_m$. If $q_m = [\, a\,]$ then $q_{m+1} = [\,\exists\,]$ else $q_{m+1} = [\,\forall\,]$.

D. $t_m = f : (A_1, \ldots, A_n, \mathbf{0})$, $n > 0$ and for each $i : 1 \leq i \leq n$, $t_m \downarrow_i t'_i$. If $q_m \neq [\, fw_1 \ldots w_n\,]$ then $t_{m+1} = t_m$, $q_{m+1} = [\,\forall\,]$ and $\theta_{m+1} = \theta_m$. Otherwise $\forall$ chooses a direction $j : 1 \leq j \leq n$ and $t_{m+1} = t'_j$. If $A_j = \mathbf{0}$ then $q_{m+1} = [\, w_j\,]$ and $\theta_{m+1} = \theta_m$. Otherwise, $w_j = \lambda x'_{i_1} \ldots x'_{i_k}.w$ and $t_{m+1} = \lambda z_1 \ldots z_k$. So, $q_{m+1} = [\, w\{c_{i_1}/x'_{i_1}, \ldots, c_{i_k}/x'_{i_k}\}\,]$ and $\theta_{m+1} = \theta_m\{c_{i_1}/z_1, \ldots, c_{i_k}/z_{i_k}\}$.

E. $t_m = y : \mathbf{0}$ and $\theta_m(y) = t'\theta_k$. Then $t_{m+1} = t'$, $q_{m+1} = q_m$ and $\theta_{m+1} = \theta_k$.

F. $t_m = y : \mathbf{0}$ and $\theta_m(y) = c$. Then $t_{m+1} = t_m$ and $\theta_{m+1} = \theta_m$. If $q_m = [\, c\,]$ then $q_{m+1} = [\,\exists\,]$ else $q_{m+1} = [\,\forall\,]$.

G. $t_m = y : (A_1, \ldots, A_n, \mathbf{0})$, $n > 0$, $\theta_m(y) = t'\theta_k$, $t' = \lambda z_1 \ldots z_n$ and for each $i : 1 \leq i \leq n$, $t_m \downarrow_i t'_i$. Then $t_{m+1} = t'$, $q_{m+1} = q_m$ and $\theta_{m+1} = \theta_k\{t'_1\theta_m/z_1, \ldots, t'_n\theta_m/z_n\}$.

H. $t_m = y : (A_1, \ldots, A_n, \mathbf{0})$, $n > 0$, $\theta_m(y) = c$ and for each $i : 1 \leq i \leq n$, $t_m \downarrow_i t'_i$. If $q_m \neq [\, cw_1 \ldots w_n\,]$ then $t_{m+1} = t_m$, $q_{m+1} = [\,\forall\,]$ and $\theta_{m+1} = \theta_m$. Otherwise $\forall$ chooses a direction $j : 1 \leq j \leq n$ and $t_{m+1} = t'_j$. If $A_j = \mathbf{0}$ then $q_{m+1} = [\, w_j\,]$ and $\theta_{m+1} = \theta_m$. Otherwise, $w_j = \lambda x'_{i_1} \ldots x'_{i_k}.w$ and $t_{m+1} = \lambda z_1 \ldots z_k$. So, $q_{m+1} = [\, w\{c_{i_1}/x'_{i_1}, \ldots, c_{i_k}/x'_{i_k}\}\,]$ and $\theta_{m+1} = \theta_m\{c_{i_1}/z_1, \ldots, c_{i_k}/z_{i_k}\}$.

Figure 4. Game moves

$t'\theta_j$ where t' is a subtree and θ_j a previous look-up table, so $j < k$. A variable y in t (or v) may be associated with a subtree t' of v_j (or t) which itself may contain free variables: hence, the need for $\theta_k(y)$ to be a pair $t'\theta_j$ as θ_j records the values of the free variables in t' at the earlier position.

Definition 10. *If the current position in $\mathsf{G}(t, P)$ is $t_m q_m \theta_m$ and q_m is not final then the next position $t_{m+1} q_{m+1} \theta_{m+1}$ is determined by a unique move in Figure 4.*

At the initial @, play proceeds to the first successor subtree $\lambda x_1 \dots x_n$ and the look-up table is updated accordingly (each of the other successor subtrees t'_{i+1} with the empty look-up table θ_1 is associated with x_i: if later play reaches a variable occurrence $x_i : A_i$ then it jumps to $t'_{i+1} : A_i$). We assume standard updating notation for θ_{m+1}: $\gamma\{\delta_1/y_1, \dots, \delta_m/y_m\}$ is the partial function similar to γ except that $\gamma(y_i) = \delta_i$. If play is at $\lambda\overline{y}$, where $\overline{y}$ can be empty, then it descends the tree. At a constant $a : \mathbf{0}$, the refuter loses if the goal state is $[\,a\,]$ and wins otherwise. At a constant f with arity more than 0, $\forall$ immediately wins if the goal state is not of the form $[\,f w_1 \dots w_n\,]$. Otherwise $\forall$ chooses a successor j and play moves to the jth successor of (the node labelled) f, the t' such that $f \downarrow_j t'$. If $w_j : \mathbf{0}$ then the goal state is $[\,w_j\,]$. However, if w_j is higher-order then forbidden constants are introduced and the look-up table is updated accordingly. If play is at a variable $y : \mathbf{0}$ and its entry in the current look-up table is $t'\theta_k$ then play jumps to t' and θ_k becomes the look-up table. If the entry for y is a forbidden constant then a final state will be reached. If y is higher-order and its entry is $t'\theta_k$ where $t' = \lambda z_1 \dots z_n$ then play jumps to t' and the look-up table is θ_k together with the association of $t'_i\theta_m$ for z_i when $y \downarrow_i t'_i$, where $i : 1 \le i \le n$: this illustrates how back-and-forth play jumping captures β-reduction as the occurrence of y is associated with t', and, therefore, occurrences of free z_i beneath t' are associated with y's ith successor. The final case is when y is higher-order and its entry in the look-up table is a forbidden constant: again, there is the possibility that further forbidden constants will be introduced. Player $\forall$ can only exercise choice at a higher-order constant whose arity is more than one (rules D and H) thereby carving out a branch of the right term u (modulo introduction of forbidden constants).

Definition 11. *Player $\forall$ loses the game $\mathsf{G}(t, P)$ if she loses every play and otherwise she wins the game.*

The game *characterizes* interpolation which essentially follows from [16, 17].

Theorem 1. *Player $\forall$ loses $\mathsf{G}(t, P)$ if, and only if, $t \models P$.*

Example 6. *A solution to Example 2 is depicted in Figure 2 The play initially proceeds as follows.*

$@[\,faa\,]\theta_1$	$(1)[\,faa\,]\theta_2 = \theta_1\{(17)\theta_1/x_1, (19)\theta_1/x_2\}$
$(2)[\,faa\,]\theta_3 = \theta_2$	$(17)[\,faa\,]\theta_4 = \theta_1\{(3)\theta_3/y_1, (15)\theta_3/y_2\}$
$(18)[\,faa\,]\theta_5 = \theta_4$	$(3)[\,faa\,]\theta_6 = \theta_3$
$(4)[\,faa\,]\theta_7 = \theta_6$	$(19)[\,faa\,]\theta_8 = \theta_1\{(5)\theta_7/y_3\}$
$(20)[\,faa\,]\theta_9 = \theta_8$	

Play starts at node @ with goal state $[\,faa\,]$ and descends to node (1) with the subtree at (17) associated with x_1 and the subtree at (19) with x_2. After play descends to (2), it jumps to (17) and then the subtree at node (3) is associated with y_1 and the subtree at (15) with y_2. So play descends to (18) and jumps to (3). The move to (4) results in a jump to (19) and then to (20). At which point there is a $\forall$ choice, as to (21) or (23), both with the same reduced goal state $[\,a\,]$. If the first choice is chosen then play continues as follows.

$(21)[\,a\,]\theta_{10} = \theta_9$	$(22)[\,a\,]\theta_{11} = \theta_{10}$
$(5)[\,a\,]\theta_{12} = \theta_7$	$(6)[\,a\,]\theta_{13} = \theta_{12}$
$(17)[\,a\,]\theta_{14} = \theta_1\{(7)\theta_{13}/y_1, (13)\theta_{13}/y_2\}$	$(18)[\,a\,]\theta_{15} = \theta_{14}$
$(7)[a]\theta_{16} = \theta_{13}$	$(8)[\,a\,]\theta_{17} = \theta_{16}$
$(17)[\,a\,]\theta_{18} = \theta_1\{(9)\theta_{17}/y_1, (11)\theta_{17}/y_2\}$	$(18)[\,a\,]\theta_{19} = \theta_{18}$
$(9)[\,a\,]\theta_{20} = \theta_{17}$	$(10)[\,a\,]\theta_{21} = \theta_{20}$
$(10)[\,\exists\,]\theta_{21} = \theta_{20}$	

The other choice is similar as both choices lead to play returning to node (5) with the goal $[\,a\,]$. Play then descends to (6) and, therefore, jumps again to (17) but now with (7) associated with y_1 and (13) with y_2. From (18) it jumps to (7). Similarly, at (8) it jumps to (17) and returns to (9) without changing state $[\,a\,]$. Consequently, play ends at (10) and $\forall$ loses.

Example 7. *Assume P is Example 4 with a solution term t all depicted in Figure 3. The single play for $\mathsf{G}(t, P)$, presented in Figure 5, is quite intricate with significant jumping in the interpolation tree. For instance, at move 3 when at node (2) play jumps to (21), descends to (22) and jumps to (3) and then later at move 11 when at node (6) jumps to (23). Even later at move 61 at node (10) which like (6) is also bound by (3), play again returns to (23). Player $\forall$ eventually loses when play reaches (12).*

If $t \models P$, so $\forall$ loses the game $\mathsf{G}(t, P)$, then the total number of different plays is at most the number of branches in the right term u of P. There are many interesting combinatorial properties of plays, some of which are briefly described in [17, 16]. For the proof of the main result, Theorem 2, the main feature is understanding the back-and-forth play jumping.

5 Interpolation tree automata

In this section we define tree automata for an interpolation problem P relative to the finite alphabet Σ. Assume a fixed problem P, $xv_1 \dots v_n = u$, and Σ with $\mathsf{R} = \mathsf{R}_P$, $C = C_P$.

$@[\,hghha\,]\theta_1$	$(1)[\,hghha\,]\theta_2 = \theta_1\{(21)\theta_1/z\}$
$(2)[\,hghha\,]\theta_3 = \theta_2$	$(21)[\,hghha\,]\theta_4 = \theta_1\{(3)\theta_3/y_1, (17)\theta_3/y_2\}$
$(22)[\,hghha\,]\theta_5 = \theta_4$	$(3)[\,hghha\,]\theta_6 = \theta_3\{(23)\theta_5/z_1\}$
$(4)[\,hghha\,]\theta_7 = \theta_6$	$(21)[\,hghha\,]\theta_8 = \theta_1\{(5)\theta_7/y_1, (13)\theta_7/y_2\}$
$(22)[\,hghha\,]\theta_9 = \theta_8$	$(5)[\,hghha\,]\theta_{10} = \theta_7\{(23)\theta_9/x_1\}$
$(6)[\,hghha\,]\theta_{11} = \theta_{10}$	$(23)[\,hghha\,]\theta_{12} = \theta_5\{(7)\theta_{11}/y_3\}$
$(24)[\,hghha\,]\theta_{13} = \theta_{12}$	$(17)[\,hghha\,]\theta_{14} = \theta_3\{(25)\theta_{13}/z_2\}$
$(18)[\,hghha\,]\theta_{15} = \theta_{14}$	$(19)[\,ghha\,]\theta_{16} = \theta_{15}$
$(20)[\,ghha\,]\theta_{17} = \theta_{16}$	$(25)[\,ghha\,]\theta_{18} = \theta_{13}$
$(26)[\,ghha\,]\theta_{19} = \theta_{18}$	$(3)[\,ghha\,]\theta_{20} = \theta_3\{(27)\theta_{19}/z_1\}$
$(4)[\,ghha\,]\theta_{21} = \theta_{20}$	$(21)[\,ghha\,]\theta_{22} = \theta_1\{(5)\theta_{21}/y_1, (13)\theta_{21}/y_2\}$
$(22)[\,ghha\,]\theta_{23} = \theta_{22}$	$(5)[\,ghha\,]\theta_{24} = \theta_{21}\{(23)\theta_{23}/x_1\}$
$(6)[\,ghha\,]\theta_{25} = \theta_{24}$	$(27)[\,ghha\,]\theta_{26} = \theta_{19}\{(7)\theta_{25}/y_4\}$
$(28)[\,ghha\,]\theta_{27} = \theta_{26}$	$(7)[\,ghha\,]\theta_{28} = \theta_{11}$
$(8)[\,ghha\,]\theta_{29} = \theta_{28}$	$(23)[\,ghha\,]\theta_{30} = \theta_9\{(9)\theta_{29}/y_3\}$
$(24)[\,ghha\,]\theta_{31} = \theta_{30}$	$(13)[\,ghha\,]\theta_{32} = \theta_7\{(25)\theta_{31}/x_2\}$
$(14)[\,ghha\,]\theta_{33} = \theta_{32}$	$(15)[\,hha\,]\theta_{34} = \theta_{33}$
$(16)[\,hha\,]\theta_{35} = \theta_{34}$	$(25)[\,hha\,]\theta_{36} = \theta_{31}$
$(26)[\,hha\,]\theta_{37} = \theta_{36}$	$(5)[\,hha\,]\theta_{38} = \theta_7\{(27)\theta_{37}/x_1\}$
$(6)[\,hha\,]\theta_{39} = \theta_{38}$	$(23)[\,hha\,]\theta_{40} = \theta_5\{(7)\theta_{39}/y_3\}$
$(24)[\,hha\,]\theta_{41} = \theta_{40}$	$(17)[\,hha\,]\theta_{42} = \theta_3\{(25)\theta_{41}/z_2\}$
$(18)[\,hha\,]\theta_{43} = \theta_{42}$	$(19)[\,ha\,]\theta_{44} = \theta_{43}$
$(20)[\,ha\,]\theta_{45} = \theta_{44}$	$(25)[\,ha\,]\theta_{46} = \theta_{41}$
$(26)[\,ha\,]\theta_{47} = \theta_{46}$	$(3)[\,ha\,]\theta_{48} = \theta_3\{(27)\theta_{47}/z_1\}$
$(4)[\,ha\,]\theta_{49} = \theta_{48}$	$(21)[\,ha\,]\theta_{50} = \theta_1\{(5)\theta_{49}/y_1, (13)\theta_{49}/y_2\}$
$(22)[\,ha\,]\theta_{51} = \theta_{50}$	$(5)[\,ha\,]\theta_{52} = \theta_{49}\{(23)\theta_{51}/x_1\}$
$(6)[\,ha\,]\theta_{53} = \theta_{52}$	$(27)[\,ha\,]\theta_{54} = \theta_{47}\{(7)\theta_{53}/y_4\}$
$(28)[\,ha\,]\theta_{55} = \theta_{54}$	$(7)[\,ha\,]\theta_{56} = \theta_{39}$
$(8)[\,ha\,]\theta_{57} = \theta_{56}$	$(27)[\,ha\,]\theta_{58} = \theta_{37}\{(9)\theta_{57}/y_4\}$
$(28)[\,ha\,]\theta_{59} = \theta_{58}$	$(9)[\,ha\,]\theta_{60} = \theta_{29}$
$(10)[\,ha\,]\theta_{61} = \theta_{60}$	$(23)[\,ha\,]\theta_{62} = \theta_5\{(11)\theta_{61}/y_3\}$
$(24)[\,ha\,]\theta_{63} = \theta_{62}$	$(17)[\,ha\,]\theta_{64} = \theta_3\{(25)\theta_{63}/z_2\}$
$(18)[\,ha\,]\theta_{65} = \theta_{64}$	$(19)[\,a\,]\theta_{66} = \theta_{65}$
$(20)[\,a\,]\theta_{67} = \theta_{66}$	$(25)[\,a\,]\theta_{68} = \theta_{63}$
$(26)[\,a\,]\theta_{69} = \theta_{68}$	$(3)[\,a\,]\theta_{70} = \theta_3\{(27)\theta_{69}/z_1\}$
$(4)[\,a\,]\theta_{71} = \theta_{70}$	$(21)[\,a\,]\theta_{72} = \theta_1\{(5)\theta_{71}/y_1, (13)\theta_{71}/y_2\}$
$(22)[\,a\,]\theta_{73} = \theta_{72}$	$(5)[\,a\,]\theta_{74} = \theta_{71}\{(23)\theta_{73}/x_1\}$
$(6)[\,a\,]\theta_{75} = \theta_{74}$	$(27)[\,a\,]\theta_{76} = \theta_{69}\{(7)\theta_{75}/y_4\}$
$(28)[\,a\,]\theta_{77} = \theta_{76}$	$(11)[\,a\,]\theta_{78} = \theta_{61}$
$(12)[\,a\,]\theta_{79} = \theta_{78}$	$(12)[\,\exists\,]\theta_{80} = \theta_{79}$

Figure 5. The play for Example 7

Definition 12. *Assume an interpolation tree. Variable occurrence z (at node n) is an immediate descendent of u (at node m) if for some i, $u \downarrow_i \lambda\overline{z}$ and $\lambda\overline{z}$ binds z (at node n). Variable occurrence z is a descendent of u if z is an immediate descendent of u or variable occurrence y is an immediate descendent of u and z is a descendent of y. Variable occurrence z is a descendent of a constant if there is a constant occurrence f and z is a descendent of f.*

The term $\lambda y.y(y(f\lambda xz_1z_2.x(y(z_2))a))$ is a solution to Example 3. The single variable occurrences of x and z_2 are descendents of the constant f. In the interpolation game, entries for such variable occurrences in a look-up table are forbidden constants. Therefore, without loss of generality, we assume that the set of variables $V \subseteq \Sigma$ is partitioned into V_1 and V_2: V_1 contains variables whose occurrences must be descendents of constants and V_2 contains those that cannot be descendents of constants. We ensure that Σ contains enough variables to guarantee this.

In the game, play jumps around the interpolation tree as illustrated in Examples 6 and 7. The question is how to capture this jumping with a tree automaton. The solution is based on Ong [9] (which is a different setting, with a fixed infinite λ-term and an alternating parity automaton). We need to break apart the *motion of jumping into and out of components* into constituents. This we do using *variable assumptions*: Ong calls them "variable profiles" in his setting.

$$\begin{array}{llll}
(z, hghha, \{ & (y_1, hghha, \{ & (z_1, hghha, \{ & (y_3, ghha, \emptyset)\}) \\
 & & (z_1, hha, \{ & (y_3, ha, \emptyset)\}) \\
 & & (z_1, ha, \{ & (y_3, a, \emptyset)\})\}) \\
 & (y_1, ghha, \{ & (z_1, ghha, & \emptyset)\}) \\
 & (y_1, ha, \{ & (z_1, ha, & \emptyset)\}) \\
 & (y_1, a, \{ & (z_1, a & \emptyset)\}) \\
 & (y_2, hghha, \{ & (z_2, ghha, & \emptyset)\}) \\
 & (y_2, hha, \{ & (z_2, ha, & \emptyset)\}) \\
 & (y_2, ha, \{ & (z_2, a, & \emptyset)\})\})
\end{array}$$

Figure 6. A z assumption for Example 3

Definition 13. *Given variables $V = V_1 \cup V_2$, right terms R and constants C, for each $z \in V$, $\Gamma(z)$ is the set of z assumptions: if $z : A$ and $z \in V_1$ then $\Gamma(z) = \{(z, r, c) \mid c : A \in C\}$; if $z : \mathbf{0}$ and $z \in V_2$ then $\Gamma(z) = \{(z, r, \emptyset) \mid r \in \mathsf{R}\}$; if $z : (A_1, \ldots, A_k, \mathbf{0})$ and $z \in V_2$ then $\Gamma(z) = \{(z, r, \Gamma) \mid r \in \mathsf{R}, \Gamma \subseteq \bigcup_{1 \le i \le k} \bigcup_{x : A_i \in V_2} \Gamma(x)\}$. A mode is a pair (r, Γ) where $r \in \mathsf{R}$ and $\Gamma \subseteq \bigcup_{z \in V} \Gamma(z)$.*

Although a variable assumption is defined independently of the interpolation game, it is intended to be an abstraction of subsequences of moves in the game $\mathsf{G}(t, P)$ for terms t (whose syntax is restricted by Σ). Consider the play in Example 6 and the interpolation tree of Figure 2. A y_1 assumption because $y_1 : \mathbf{0}$ has the form $(y_1, r', \emptyset)$ which captures subsequences of length 1 in a play. For instance, $(y_1, faa, \emptyset)$ captures the move $(18)\,[faa]\,\theta_5$. An x_1 assumption has the form (x_1, r, Γ) where Γ is a set of y_1 and y_2 assumptions. The assumption $(x_1, faa, \{(y_1, faa, \emptyset)\})$ represents the sequence of moves $(2)[faa]\theta_3, (17)[faa]\theta_4, (18)[faa]\theta_5, (3)[faa]\theta_6$ capturing the interaction between x_1 and y_1. There is abstraction here, because, for example, $(x_1, a, \{(y_1, a, \emptyset)\})$ represents the *two* sequences $(6)[a]\theta_{13}, (17)[a]\theta_{14}, (18)[a]\theta_{14}, (7)[a]\theta_{15}$ and $(8)[a]\theta_{17}, (17)[a]\theta_{18}, (18)[a]\theta_{19}, (9)[a]\theta_{20}$. A more elaborate assumption is in Figure 6 which represents the interaction between nodes (2), (21), (22), (3), (6), (23), (24), (7), (10), (26), (27), (28) and (11) of Figure 3 in the play of Figure 5.

A mode is a pair (r, Γ) where $r \in \mathsf{R}$ and Γ is a set of variable assumptions. Because R is finite and Σ is fixed, there can only be boundedly many different modes (r, Γ). We now come to the formal definition of the tree automaton whose states are sets of modes.

Definition 14. *Assume P, $xv_1 \ldots v_n = u$, Σ and the variables V partitioned into $V_1 \cup V_2$. The Σ-tree automaton for P is $\mathsf{A}_P = (Q, \Sigma, F, \Delta)$ where Q is the set of sets of modes $\{(r_1, \Gamma_1), \ldots, (r_k, \Gamma_k)\}$, $k \ge 0$, $F = \{\{(u, \emptyset)\}\}$ and the transition relation Δ is defined on nodes of the Σ-tree by cases on Σ.*

- $a : \mathbf{0}$. *Then $a \Rightarrow \Lambda$ where $\Lambda \subseteq \{(a, \emptyset)\}$.*
- $z : \mathbf{0}$ *and* $z \in V_1$. *Then* $z \Rightarrow \Lambda \subseteq$

$\{(c_1, \{(z, c_1, c_1)\}), \ldots, (c_k, \{(z, c_k, c_k)\})\}$ *for* $k \geq 0$ *and* $c_1 : \mathbf{0}, \ldots, c_k : \mathbf{0} \in C$.

- $z : \mathbf{0}$ *and* $z \in V_2$. *Then* $z \Rightarrow \Lambda \subseteq \{(r_1, \{(z, r_1, \emptyset)\}), \ldots, (r_m, \{(z, r_m, \emptyset)\})\}$ *for* $m \geq 0$ *and* $r_1, \ldots, r_m \in \mathsf{R}$.
- $f : (A_1, \ldots, A_k, \mathbf{0})$. *Then* $f \Lambda_1 \ldots \Lambda_k \Rightarrow \Lambda$ *when (1) and (2). (1). If* $(r, \Gamma) \in \Lambda$ *then* $r = f w_1 \ldots w_k$ *and for each* $i : 1 \leq i \leq k$ *there is* $(r_i, \Gamma_i) \in \Lambda_i$ *with* $w_i = \lambda x'_{i_1} \ldots x'_{i_{m_i}}.w'_i$, $m_i \geq 0$, $f \downarrow_i \lambda y^i_1 \ldots y^i_{m_i}$, $r_i = w'_i\{c_{i_1}/x'_{i_1}, \ldots, c_{i_{m_i}}/x'_{i_{m_i}}\}$ *and* $\Gamma = \bigcup_{1 \leq i \leq k}(\Gamma_i - \bigcup_{1 \leq j \leq m_i} \Gamma(y^i_j))$. *(2). If* $(r_i, \Gamma_i) \in \Lambda_i$ *for* $i : 1 \leq i \leq k$ *then there is* $(f w_1 \ldots w_k, \Gamma) \in \Lambda$ *with* $w_i = \lambda x'_{i_1} \ldots x'_{i_{m_i}}.w'_i$, $m_i \geq 0$, $f \downarrow_i \lambda y^i_1 \ldots y^i_{m_i}$, $r_i = w'_i\{c_{i_1}/x'_{i_1}, \ldots, c_{i_{m_i}}/x'_{i_{m_i}}\}$ *and* $(\Gamma_i - \bigcup_{1 \leq j \leq m_i} \Gamma(y^i_j)) \subseteq \Gamma$.
- $z : (A_1, \ldots, A_k, \mathbf{0})$ *and* $z \in V_1$. *Then* $z \Lambda_1 \ldots \Lambda_k \Rightarrow \Lambda$ *when (1) and (2). (1). If* $(r, \Gamma) \in \Lambda$ *then* $r = c w_1 \ldots w_k$, $(z, r, c) \in \Gamma$ *and for each* $i : 1 \leq i \leq k$ *there is* $(r_i, \Gamma_i) \in \Lambda_i$ *with* $w_i = \lambda x'_{i_1} \ldots x'_{i_{m_i}}.w'_i$, $m_i \geq 0$, $z \downarrow_i \lambda y^i_1 \ldots y^i_{m_i}$, $r_i = w'_i\{c_{i_1}/x'_{i_1}, \ldots, c_{i_{m_i}}/x'_{i_{m_i}}\}$ *and* $\Gamma - \{(z, r, c)\} = \bigcup_{1 \leq i \leq k}(\Gamma_i - \bigcup_{1 \leq j \leq m_i} \Gamma(y^i_j))$. *(2). If* $(r_i, \Gamma_i) \in \Lambda_i$ *for* $i : 1 \leq i \leq k$ *then there is* $(r, \Gamma) \in \Lambda$ *with* $r = c w_1 \ldots w_k$ *and* $(z, r, c) \in \Gamma$, $w_i = \lambda x'_{i_1} \ldots x'_{i_{m_i}}.w'_i$, $m_i \geq 0$, $z \downarrow_i \lambda y^i_1 \ldots y^i_{m_i}$, $r_i = w'_i\{c_{i_1}/x'_{i_1}, \ldots, c_{i_{m_i}}/x'_{i_{m_i}}\}$ *and* $(\Gamma_i - \bigcup_{1 \leq j \leq m_i} \Gamma(y^i_j)) \subseteq \Gamma$.
- $z : (A_1, \ldots, A_k, \mathbf{0})$ *and* $z \in V_2$. *Then* $z \Lambda_1 \ldots \Lambda_k \Rightarrow \Lambda$ *when (1) and (2). (1). If* $(r, \Gamma) \in \Lambda$ *then there is a* $(z, r, \Gamma') \in \Gamma$ *and* $\Gamma' = \{(y_{11}, r_{11}, \Gamma'_{11}), \ldots, (y_{1m_1}, r_{1m_1}, \Gamma'_{1m_1}), \ldots, (y_{k1}, r_{k1}, \Gamma'_{k1}), \ldots, (y_{km_k}, r_{km_k}, \Gamma'_{km_k})\}$ *for* $m_i \geq 0$, $1 \leq i \leq k$ *and each* $y_{ij} : A_i$ *and* $(r_{ij}, \Gamma'_{ij} \cup \Sigma_{ij}) \in \Lambda_i$ *where* $\bigcup_{1 \leq i \leq k} \bigcup_{1 \leq j \leq m_i} \Sigma_{ij} \cup \{(z, r, \Gamma')\} = \Gamma$. *(2). If* $(r_i, \Gamma_i) \in \Lambda_i$ *then there is* $(r, \Gamma) \in \Lambda$ *and* $(z, r, \Gamma') \in \Gamma$ *and* $(y, r_i, \Gamma'') \in \Gamma'$, $y : A_i$ *and* $\Gamma'' \subseteq \Gamma_i$ *and* $\Gamma_i - \Gamma'' \subseteq \Gamma$.
- $\lambda \overline{y}$. *Then* $\lambda \overline{y}\, \Lambda \Rightarrow \Lambda$ *for any* Λ.
- $@ : ((A_1, \ldots, A_n, \mathbf{0}), A_1, \ldots, A_n, \mathbf{0})$. *Then* $@\Lambda_1 \ldots \Lambda_{n+1} \Rightarrow \{(u, \emptyset)\}$ *when (1). (1).* $\Lambda_1 = \{(u, \Gamma)\}$ *and if* $(z, r', \Gamma') \in \Gamma$ *then* $z = x_i$ *for some* i *and* $(r', \Gamma') \in \Lambda_i$, *and if* $(r_i, \Gamma_i) \in \Lambda_i$ *then* $(x_i, r_i, \Gamma_i) \in \Gamma$.

Example 8. *Consider the automaton for Example 2. A potential solution term such as the one depicted in Figure 2 has the form* $\lambda x_1 x_2.w$. *So* $\Sigma = \{a, b, f, x_1, x_2, y_1, y_3, \lambda, \lambda y_3, \lambda y_1 y_2, \lambda x_1 x_2\}$. *The variables* V *are* $\{x_1, x_2, y_1, y_3\}$, $V_1 = \emptyset$ *and* $V_2 = V$. *The set* $\mathsf{R} = \{faa, a\}$. *A mode is* (r, Γ) *where* $r \in \mathsf{R}$ *and* Γ *is a set of variable assumptions. Next, we consider the transitions, first for constants.*

$$a \Rightarrow \Lambda \subseteq \{(a, \emptyset)\} \quad b \Rightarrow \emptyset \quad f\, \emptyset\, \emptyset \Rightarrow \emptyset$$

$$f\{(a, \Gamma_1)\}\, \{(a, \Gamma_2)\} \Rightarrow \{(faa, \Gamma_1 \cup \Gamma_2)\}$$

The last rule allows f *to be labelled with the state* $\{(faa, \Gamma_1 \cup \Gamma_2)\}$ *when its descendents are labelled with the states* $\{(a, \Gamma_1)\}$ *and* $\{(a, \Gamma_2)\}$. *Next we examine the rules for variables* y_1 *and* y_3 *which are both of type* $\mathbf{0}$.

$$y_1 \Rightarrow \Lambda \subseteq \{(faa, \{(y_1, faa, \emptyset)\}), (a, \{(y_1, a, \emptyset)\})\}$$

$$y_3 \Rightarrow \Lambda \subseteq \{(a, \{(y_3, a, \emptyset)\})\}$$

It is unnecessary to include $(faa, \{(y_3, faa, \emptyset)\})$ *as a mode because it could never appear within a state of an accepting run of the automaton. The rules for* x_1 *and* x_2 *are more interesting. Let* $A(x_1) = (x_1, faa, \{(y_1, faa, \emptyset)\})$, $A'(x_1) = (x_1, a, \{(y_1, a, \emptyset)\})$ *and* $A(x_2) = (x_2, faa, \{(y_3, a, \emptyset)\})$. *Assume* Γ' *is* $\{A(x_1), A'(x_1), A(x_2)\}$.

$$x_1\, \emptyset\, \emptyset \Rightarrow \emptyset \quad x_2\, \emptyset \Rightarrow \emptyset$$

$$x_2\{(a, \Gamma)\} \Rightarrow \{(faa, \Gamma \cup \{A(x_2)\})\} \ \ \Gamma \subseteq \{A'(x_1)\}$$

$$x_1\, \{(a, \Gamma)\}\, \emptyset \Rightarrow \{(a, \Gamma \cup \{A'(x_1)\})\}\ \Gamma \subseteq \{A'(x_1)\}$$

$$x_1\, \{(faa, \Gamma)\}\, \emptyset \Rightarrow \{(faa, \Gamma \cup \{A(x_1)\})\}\ \Gamma \subseteq \Gamma'$$

A variable occurrence x_1 *can be labelled with the state* $\{(faa, \Gamma'')\}$ *provided that* $A(x_1) \in \Gamma''$ *and its first successor is labelled* $\{(faa, \Gamma)\}$ *such that* $\Gamma \cup \{A(x_1)\} = \Gamma''$. *There are the rules for the* λ*'s and* $@$.

$$\lambda\, \Lambda \Rightarrow \Lambda \quad \lambda x_1 x_2\, \Lambda \Rightarrow \Lambda$$

$$\lambda y_1 y_2\, \Lambda \Rightarrow \Lambda \quad \lambda y_3\, \Lambda \Rightarrow \Lambda$$

$$@\ \{(faa, \Gamma)\}\ \Lambda_1\ \Lambda_2 \Rightarrow \{(faa, \emptyset)\} \text{ if (1)}$$

where (1) is: $A(x_1) \in \Gamma$ *iff* $(faa, \{(y_1, faa, \emptyset)\}) \in \Lambda_1$, $A'(x_1) \in \Gamma$ *iff* $(a, \{(y_1, a, \emptyset)\}) \in \Lambda_1$ *and* $A(x_2) \in \Gamma$ *iff* $(faa, \{(y_3, a, \emptyset)\}) \in \Lambda_2$. *Using these rules, it is easy to show that the tree of Figure 2 is accepted. The rules for* x_1 *allow "pumping" in the sense of [3]: the automaton accepts any term* $\lambda x_1 x_2.x_1(x_1 \ldots x_1(x_2(x_1 \ldots x_1 ab)b \ldots) \ldots)b$.

The following is the main result of the paper.

Theorem 2. *Assume* P, $xv_1 \ldots v_n = u$, Σ *and the* Σ*-tree automaton* A_P. *For any well-formed* Σ*-term* t, A_P *accepts the tree* $@tv_1 \ldots v_n$ *iff* $t \models P$.

6 Conclusion

It is unclear whether we can generalize Theorem 2 without developing automata for languages with infinite alphabets. In fact, there is the more specific problem of how to define automata in the presence of higher-order binding. For 4th-order problems, there is an automaton that recognises its full set of solutions (up to α-equivalence) even though the syntax may be infinite. Comon and Jurski define a special kind of automata, $\Box$-automata, to achieve this [2]. The occurrence of a leaf $\Box$ in a term tree represents any (syntactically correct) subtree. A $\Box$ cannot contribute to the solution of the matching problem. Given a 4th-order problem P, there is a finite alphabet Σ such that every solution t is recognised by a $\Box$-automaton (where $\Box$s of the correct type replace subtrees of t and all other nodes of t are labelled by elements of Σ). In our framework, we can also achieve this. Play in $\mathsf{G}(t, P)$ does not enter a subtree that is replaced by $\Box$. Consequently, given the set Σ, we can assume the extra schematic automaton rule $u\emptyset \dots \emptyset \Rightarrow \emptyset$ for all $u \notin \Sigma$.

What about decidability of higher-order matching? A corollary of Theorem 2 is that matching relative to *resource* is decidable. Indeed this is also true for λ-definability [7], even though without out this restriction the problem is undecidable. For higher-order matching, a proof of the small model property is, therefore, still needed. However given the property, there is now a more intelligent decision procedure using non-emptiness of tree automata. Moreover, it is possible that automata could underpin a more direct proof of this property. The thought is that given an interpolation equation $xv_1 \dots v_n = u$, $x : A = (A_1, \dots, A_n, \mathbf{0})$, there can only be finitely many "inequivalent" $x_i : A_i$ variable assumptions because of their bounded depth irrespective of the number of different variables a term $t : A$ may contain.

References

[1] Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S. and Tommasi, M. *Tree Automata Techniques and Applications*. Draft Book, http://l3ux02.univ-lille3.fr/tata/ (2002).

[2] Comon, H. and Jurski, Y. Higher-order matching and tree automata. *Lecture Notes in Computer Science*, **1414**, 157-176, (1997).

[3] Dowek, G. Third-order matching is decidable. *Annals of Pure and Applied Logic*, **69**, 135-155, (1994).

[4] Dowek, G. Higher-order unification and matching. In A. Robinson and A. Voronkov ed. *Handbook of Automated Reasoning*, Vol 2, 1009-1062, North-Holland, (2001).

[5] Huet, G. *Rèsolution d'èquations dans les langages d'ordre* 1, 2, ... ω. Thèse de doctorat d'ètat, Universitè Paris VII, (1976).

[6] Joly, T. Encoding of the halting problem into the monster type and applications. *Lecture Notes in Computer Science*, **2701**, 153-166, (2003).

[7] Jung, A. and Tiuryn, J. A new characterisation of lambda definability. *Lecture Notes in Computer Science*, **664**, 245-257, (1993).

[8] Loader, R. Higher-order β-matching is undecidable, *Logic Journal of the IGPL*, **11(1)**, 51-68, (2003).

[9] Ong, C.-H. L. *On model-checking trees generated by higher-order recursion schemes*, Procs LICS 2006, 81-90. (Longer version available from Ong's web page, 42 pages preprint.)

[10] Padovani, V. Decidability of fourth-order matching. *Mathematical Structures in Computer Science*, **10(3)**, 361-372, (2001).

[11] Segoufin, L. Automata and logics for words and trees over an infinite alphabet. *Lecture Notes in Computer Science*, **4207**, 41-57, (2006).

[12] Schubert, A. Linear interpolation for the higher-order matching problem. *Lecture Notes in Computer Science*, **1214**, 441-452, (1997).

[13] Statman, R. The typed λ-calculus is not elementary recursive. *Theoretical Computer Science*, **9**, 73-81, (1979).

[14] Statman, R. Completeness, invariance and λ-definability. *The Journal of Symbolic Logic*, **47**, 17-26, (1982).

[15] Stirling, C. *Modal and Temporal Properties of Processes*, Texts in Computer Science, Springer, (2001).

[16] Stirling, C. Higher-order matching and games. *Lecture Notes in Computer Science*, **3634**, 119-134, (2005).

[17] Stirling, C. A game-theoretic approach to deciding higher-order matching. *Lecture Notes in Computer Science*, **4052**, 348-359, (2006).