Under consideration for publication in Math. Struct. in Comp. Science

An introduction to decidability of higher-order matching

Colin Stirling

School of Informatics, Informatics Forum, University of Edinburgh

email: cps@inf.ed.ac.uk

Received December 2010

Keywords: simply typed lambda calculus, higher-order matching, games, tree automata.

Contents

1. Introduction

Higher-order unification is the problem given an equation $t = u$ containing free variables is there a solution substitution θ such that $t\theta$ and $u\theta$ have the same normal form? The terms t and u belong to the simply typed lambda calculus and the same normal form is with respect to $\beta\eta$-equivalence. Higher-order matching is the particular instance when the term u is closed; can t be pattern matched to u? Although higher order unification is undecidable (even if free variables are only second-order (Goldfarb 1982)), higher-order matching was conjectured to be decidable by Huet (Huet 1976). If matching is decidable then it is known to have non-elementary complexity (Statman 1979; Wierzbicki 1999). Decidability had been proved for the general problem up to order 4 (by showing decidability of observational equivalence of lambda terms) and for various special cases (Padovani 1996; Padovani 2000; Schubert 1997; Schmidt-Schauβ 2003; Dougherty and Wierzbicki 2002). Comon and Jurski define tree automata that characterise all solutions to a 4th-order problem, thereby, showing that they form a regular set (Comon and Jurski 1997). Loader showed that matching is undecidable for the variant definition of the same normal form that only uses β-equivalence by encoding lambda definability as matching (Loader 2003): see (Joly 2005) for a proof that uses the halting problem.

In (Stirling 2009A) we confirmed Huet's conjecture that matching is decidable. It appeals to Padovani's and Schubert's reduction of matching to the interpolation problem consisting of equations of the form $xw_1 \dots w_k = u$ where there is only one free variable occurrence x (Schubert 1997; Padovani 2000). The proof technique is then somewhat different from the methods used in (Padovani 1996; Padovani 2000). It starts with a game theoretic understanding of a solution to an interpolation problem which is essentially a game semantic understanding of typed lambda calculus. The aim is to avoid explicit β-reduction and substitution by understanding them in terms of sequences of positions in a play on the fixed term tree $tw_1 \dots w_k$; t is a potential solution term which is applied to the arguments $w_1 \dots w_k$; and we want to know whether the normal form of the application is u; the terms t, $w_1, \dots, w_k, u$ are all in η-long normal form. The decidability proof is very complicated. It isolates uniformity properties of game playing and shows that based on them, subterms can be manipulated so that an interpolation problem has a solution iff it has a small solution. This is in the same spirit as the small model property in modal or temporal logics; if a formula is satisfiable then it is satisfiable in model bounded in the size of the formula.

One aim of this paper is to introduce the topic to a wider audience; so we include a brief introduction to simply typed lambda calculus, matching and interpolation with concrete examples. Another aim of the paper is to offer a simpler proof of decidability which still uses games. With the game theoretic characterisation of solutions to an interpolation problem $tw_1 \dots w_k$ in (Stirling 2009A) play only traverses nodes of the potential solution term t; its arguments $w_1, \dots, w_k$ are coded within states of a play position. Here, instead, we employ a symmetric game where positions may jump from nodes of t to nodes of w_j and back again. There still is the asymmetry in the analysis as the w_js are fixed in advance. The proof still involves identifying uniformity properties of sequences of positions and manipulating subterms. We also describe an alternative framework using tree automata for understanding interpolation trees $tw_1 \dots w_k$. We start with (Comon and Jurski 1997) characterisation of solutions to 4th-order problems and then consider how to generalise tree automata to higher-orders. This requires an excursion into automata that can cope with an infinite alphabet. Although we introduce a new kind of tree automaton and characterise solutions to interpolation problems this does not lead to decidability because the non-emptiness problem of such automata is undecidable. Much of the tree automata discussion is from (Stirling 2007; Stirling 2009).

In Section 2 simply typed lambda calculus, higher-order matching and interpolation are briefly introduced. In section 3 we describe tree automata and how they can be used following (Comon and Jurski 1997) to characterise solutions of interpolation equations of order 4. We then provide the game theoretic characterisation of interpolation, give examples and start to analyse properties of games in Section 4. Tree automata are revisited in Section 5 where we describe a new kind of such automata that leads to new characterisations of solutions to interpolation problems but not to decidability. Finally we proceed to the proof of decidability using the small solution property in Section 6.5. Then we conclude with general remarks and ideas for future work.

2. Background

2.1. *Simply typed lambda calculus*

Simple types are generated from base types using the binary function operator $\rightarrow$. For ease of exposition, we assume that there is just one base type $\mathbf{0}$: what is to follow can be extended to the case when there are arbitrary many base types.

Definition 1. Simple types are defined by the following grammar.

$$A ::= \mathbf{0} \mid A \rightarrow A$$

We use capital letters from the initial part of the alphabet to range over types. $A \rightarrow B$ is the type of functions that take elements of type A and return elements of type B. Assuming $\rightarrow$ associates to the right, so $A \rightarrow B \rightarrow B'$ is $A \rightarrow (B \rightarrow B')$, if A is not the base type $\mathbf{0}$ then it has the form $A_1 \rightarrow \dots \rightarrow A_n \rightarrow \mathbf{0}$ which is abbreviated to $(A_1, \dots, A_n, \mathbf{0})$.

Definition 2. The order of $\mathbf{0}$ is 1 and the order of $(A_1, \dots, A_n, \mathbf{0})$ is $k+1$ where k is the maximum of the orders of the A_is. The arity of $\mathbf{0}$ is 0 and the arity of $(A_1, \dots, A_n, \mathbf{0})$ is m where m is the maximum of n and the arities of the A_is.

Example 1. $(\mathbf{0}, \mathbf{0}, \mathbf{0})$ is the type of a function that takes two elements of base type, in turn, and returns an element of base type; it has order 2 and arity 2. $((\mathbf{0}, \mathbf{0}), \mathbf{0})$ has arity 1 and order 3 because it is the type of a function that takes a function (from base type to base type) and returns an element of base type. The monster type $M = ((((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0}), \mathbf{0}, \mathbf{0})$, see (Joly 2001), has arity 2 and order 5: $\text{order}(M) = 1 + \text{order}((((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})) = 2 + \text{order}(((\mathbf{0}, \mathbf{0}), \mathbf{0})) = 3 + \text{order}((\mathbf{0}, \mathbf{0})) = 4 + \text{order}(\mathbf{0}) = 5$. It is the type of a function that takes two arguments, the second of which has base type, and returns an element of base type; the first argument has type $(((\mathbf{0},\mathbf{0}),\mathbf{0}),\mathbf{0})$ that takes a function of type $((\mathbf{0},\mathbf{0}),\mathbf{0})$ and returns an element of base type. $\square$

Terms that have simple types are generated from a countable set of *typed* variables $x^A, y^B, \dots$ and *typed* constants $a^{\mathbf{0}}, f^A, \dots$ (and, therefore, in Church style (Barendregt 1992)) using the operators of lambda abstraction and function application. We write $t : A$ to mean term t has type A.

Definition 3. The set of simply typed terms is the smallest set T such that

1 if x^A then $x : A \in T$,
2 if f^A then $f : A \in T$,
3 if $t : B \in T$ and $x : A \in T$ then $\lambda x.t : A \rightarrow B \in T$,
4 if $t : A \rightarrow B \in T$ and $u : A \in T$ then $(tu) : B \in T$.

We let $s, t, \dots$ range over terms in T. Function abstraction is given by lambda abstraction: $\lambda x.t$ has type $A \rightarrow B$, if $x : A$ and $t : B$. Function application is written using juxtaposition; $t : A \rightarrow B$ applied to $u : A$ is (tu) of type B. In a sequence of unparenthesised applications, we adopt the familiar convention that application associates to the left, so $tu_1 \dots u_k$ is $((\dots(tu_1)\dots)u_k)$.

Usual definitions of when a variable occurrence is free or bound and when a term is closed are assumed: for instance in $\lambda x.\lambda y.xy(\lambda u.xzu)$ both occurrences of x in $xy(\lambda u.xzu)$ are bound (by λx) and the occurrence of z is free; this term is, therefore, not closed. As this example illustrates, we shall avoid explicitly presenting types when they are not germane to the discussion. We also assume the usual notion of α-equivalence between terms where bound variable occurrences and their binders may be renamed with fresh variables of the same type; thus, $\lambda x_1.\lambda y_1.x_1y_1(\lambda u.x_1zu)$ is α-equivalent to the term above.

Definition 4. The *order* of a term $t : A$ is the order of A.

Example 2. Let $f : (\mathbf{0},\mathbf{0},\mathbf{0})$ and $a, b : \mathbf{0}$ be constants; the term $fab : \mathbf{0}$ because unabbreviated $f : \mathbf{0} \rightarrow (\mathbf{0} \rightarrow \mathbf{0})$, so $(fa) : \mathbf{0} \rightarrow \mathbf{0}$ and $((fa)b) : \mathbf{0}$. The term $t = \lambda x.\lambda y.x(\lambda z.zy)$ is closed and has the monster type M of Example 1 when $x : ((((\mathbf{0},\mathbf{0}),\mathbf{0})),\mathbf{0})$, $y : \mathbf{0}$ and $z : (\mathbf{0},\mathbf{0})$; so, $zy : \mathbf{0}$, $\lambda z.zy : ((\mathbf{0},\mathbf{0}),\mathbf{0})$, $x(\lambda z.zy) : \mathbf{0}$, and, therefore, $\lambda y.x(\lambda z.zy) : (\mathbf{0},\mathbf{0})$ and $t : M$ has order 5. $\square$

Another abbreviation is $\lambda z_1 \dots z_m$ for $\lambda z_1 \dots \lambda z_m$; so the term $t : M$ in Example 2 is $\lambda xy.x(\lambda z.zy)$.

Terms have dynamics through rewriting or reduction rules.

β-reduction	$(\lambda x.t)u \rightarrow_\beta t\{u/x\}$	
η-reduction	$\lambda x.tx \rightarrow_\eta t$	if x is not free in t

Here $\{u_1/x_1\}$ and, more generally, $\{u_1/x_1, \dots, u_n/x_n\}$ is (simultaneous) substitution where the x_is are distinct, of simultaneously replacing all free occurrences of x_i with u_i, and avoiding variable capture by renaming bound variables; u_i and x_i must have the same type for each i. The application of a rewrite rule to a term consists of replacing a subterm by its reduction: for instance, $\lambda z.((\lambda x.x)(faz)) \rightarrow_\beta \lambda z.faz$ because $(\lambda x.x)(faz) \rightarrow_\beta faz$;

and $\lambda z.faz \to_\eta fa$. Some well known properties of reduction are now described, see, for instance, (Barendregt 1984; Barendregt 1992; Dowek 2001; Hindley and Seldin 1986); we use standard notation, $\to_{\beta\eta}$ is $\to_\beta \cup \to_\eta$ and $\to^*_\delta$ is the reflexive and transitive closure of $\to_\delta$.

Fact 1. Let $t : A$ and $\delta \in \{\beta, \eta, \beta\eta\}$.

1 (Subject reduction) If $t \to_\delta t'$ then $t' : A$.
2 (Strong normalisation) There is not an infinite sequence of the form $t \to_\delta t_1 \to_\delta \ldots \to_\delta t_n \to_\delta \ldots$.
3 (Confluence) If $t \to^*_\delta t_1$ and $t \to^*_\delta t_2$ then for some s, $t_1 \to^*_\delta s$ and $t_2 \to^*_\delta s$.

Definition 5. Assume $\delta \in \{\beta, \eta, \beta\eta\}$. Terms t, t' are δ-equivalent, $t =_\delta t'$, if there are s, s' such that $t \to^*_\delta s$ and $t' \to^*_\delta s'$ and s is α-equivalent to s'.

We are primarily interested in $\beta\eta$-equivalence, $=_{\beta\eta}$, between terms. Confluence and strong normalisation ensure that terms reduce to (unique) *normal forms*; that is, to terms where there is no application of reduction. However, as is common, we wish to understand $\beta\eta$-equivalence without invoking η-reductions; the method for doing this uses β-normal forms of a particular kind.

Definition 6.

1 Term t is in β-normal form if there is not a t' such that $t \to_\beta t'$.
2 Term t in β-normal form is η-long if there is not a t' such that $t' \to_\eta t$.

We abbreviate η-long β-normal form to *long normal form*, or lnf for short. Lnfs are preserved under β-reductions.

Fact 2. If $t : A \to B$ and $u : A$ are both in lnf, and $tu \to^*_\beta t'$ and t' is in β-normal form then t' is in lnf.

Fact 3.

1 If $t =_\beta t'$ then there is a unique s (up to α-equivalence) in β-normal form such that $t =_\beta s$ and $t' =_\beta s$.
2 If $t =_{\beta\eta} t'$ then there is a unique s (up to α-equivalence) in lnf such that $t =_{\beta\eta} s$ and $t' =_{\beta\eta} s$.

Next we characterise syntactically terms that are in normal form. The two kinds of normal form are similar. We proceed by cases on the type. A term of type $\mathbf{0}$ is in β-normal form if it is

— a constant or a variable $u : \mathbf{0}$, or
— $u\, t_1 \ldots t_k$ where $u : (A_1, \ldots, A_k, \mathbf{0})$ is a constant or a variable and each $t_i : A_i$ is in β-normal form.

A lnf of type $\mathbf{0}$ is

— a constant or a variable $u : \mathbf{0}$, or
— $u\, t_1 \ldots t_k$ where $u : (A_1, \ldots, A_k, \mathbf{0})$ is a constant or a variable and each $t_i : A_i$ is in *lnf*.

A term of type $(A_1, \ldots, A_n, \mathbf{0})$ is in β-normal form if it is

— a constant or variable $u : (A_1, \ldots, A_n, \mathbf{0})$, or

— $ut_1 \ldots t_m$, $m \geq 1$ where $u : (B_1, \ldots, B_m, A_1, \ldots, A_n, \mathbf{0})$ is a constant or variable and each $t_i : B_i$ is in β-normal form, or

— $\lambda x_1 \ldots x_k.t'$, $1 \leq k \leq n$, each $x_i : A_i$, t' is in β-normal form and if $k < n$ then $t' : (A_{k+1}, \ldots, A_n, \mathbf{0})$ and if $k = n$ then $t' : \mathbf{0}$.

For lnf it is much simpler; a term of type $(A_1, \ldots, A_n, \mathbf{0})$ is in lnf if it is

— $\lambda x_1 \ldots x_n.t'$, each $x_i : A_i$ and $t' : \mathbf{0}$ is in lnf.

Example 3. Assume $x : (\mathbf{0}, \mathbf{0})$. Then x and $\lambda x.x : ((\mathbf{0}, \mathbf{0}), \mathbf{0}, \mathbf{0})$ are in β-normal form but not lnf. If $z : \mathbf{0}$ then both $\lambda z.xz : (\mathbf{0}, \mathbf{0})$ and $\lambda xz.xz : ((\mathbf{0}, \mathbf{0}), \mathbf{0}, \mathbf{0})$ are in lnf. □

Definition 7. For any type A and set of constants C, $T_A(C)$ is the set of closed lnf terms of type A whose constants belong to C.

Example 4. Let $A = (\mathbf{0}, \mathbf{0}, \mathbf{0})$ and $C = \{f, g\}$ where $f : (\mathbf{0}, \mathbf{0}, \mathbf{0})$ and $g : (\mathbf{0}, \mathbf{0})$. The set $T_A(C)$ consists of the terms $\lambda x_1 x_2.s$ with $s ::= x_1 \mid x_2 \mid gs_1 \mid fs_1 s_2$. An example of a term in $T_M(\emptyset)$ where M is the monster type is $\lambda xy.x(\lambda z_1.x(\lambda z_2.z_1 y))$ when $x : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$, $z_i : (\mathbf{0}, \mathbf{0})$ and $y : \mathbf{0}$. □

2.2. *Higher-order matching and interpolation*

We now introduce the higher-order matching problem, posed in (Huet 1976), and also (Statman 1982).

Definition 8. A matching problem is an equation $v = u$ where $v, u : \mathbf{0}$ are in lnf 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 terms $t_1, \ldots, t_n$ such that $v\{t_1/x_1, \ldots, t_n/x_n\}$ $=_{\beta\eta}$ u. The arity of the problem is the largest arity of the types A_i such that $x_i : A_i$.

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

Matching is an instance of two undecidable problems, higher-order unification and higher-order β-matching. A unification problem is a similar equation $v = u$ except that free variables may occur on both sides, in v and u; so, a solution is a sequence of terms $t_1, \ldots, t_n$ such that $v\{t_1/x_1, \ldots, t_n/x_n\} =_{\beta\eta} u\{t_1/x_1, \ldots, t_n/x_n\}$. This problem is undecidable, even at order 2 (Goldfarb 1982). A β-matching problem is an equation $v = u$ where $v, u : \mathbf{0}$ are in β-normal form and u is closed; a solution is a sequence of terms $t_1, \ldots, t_n$ such that $v\{t_1/x_1, \ldots, t_n/x_n\} =_\beta u$. This problem is undecidable at order 5 and beyond (Loader 2003; Joly 2005) (and decidable at orders below 5 (Padovani 2000)). There are other closely related problems that are undecidable which if they were decidable would imply that matching is decidable; the most famous of which is lambda definability (Loader 2001; Statman 1982); the others are the type inhabitation problem in the presence of intersection types (Urzyczyn 1999) and the non-emptiness problem for a particular kind of alternating tree automata which we shall see later (Stirling 2009;

Ong and Tzevelekos 2009); this last result appeals to the undecidability of observational equivalence for finitary PCF (Loader 2001A).

In the literature there are alternative versions of the matching problem. First is the general case of an equation $v = u$ when $v, u : A$ for arbitrary A and u is closed. If $A = (A_1, \dots, A_k, \mathbf{0})$ then the term u has the form $\lambda z_1 \dots z_k.u'$; therefore, there is the matching problem $v' = u''$ when v', u'' are the lnfs of $vc_1 \dots c_k$ and $u'\{c_1/z_1, \dots, c_k/z_k\}$ and the c_i's are new constants of correct type that cannot occur in solution terms. Statman presents "the range question" (Statman 1982): given closed $v : (A_1, \dots, A_n, B)$ and $u : B$ are there terms $t_1 : A_1, \dots, t_n : A_n$ such that $vt_1 \dots t_n =_{\beta\eta} u$? This is the matching problem $v' = u$ where v' is the lnf of $vx_1 \dots x_n$ where each x_i has type A_i. In (Padovani 2000) a matching problem is a family of matching equations $v_1 = u_1, \dots, v_m = u_m$ to be solved uniformly, as all occurrences of free x_i in each equation must be instantiated with the same t_i: this problem reduces to the single equation $fv_1 \dots v_m = fu_1 \dots u_m$ where f is a constant of the appropriate type.

There is a simplified version of matching which is essentially monadic, involving a single free variable.

Definition 9. Assume $u : \mathbf{0}$ and $w_i : A_i$, $1 \leq i \leq k$, are closed terms in lnf and $x : (A_1, \dots, A_k, \mathbf{0})$. An interpolation equation is $xw_1 \dots w_k = u$. The type and order of the equation is that of x. A solution is a closed term $t : (A_1, \dots, A_k, \mathbf{0})$ in lnf such that $tw_1 \dots w_k =_{\beta\eta} u$. The term u of the equation $xw_1 \dots w_k = u$ is called its goal term.

Proposition 4. If t solves $xw_1 \dots w_k = u$ then there is a lnf u' such that $tw_1 \dots w_k \rightarrow^*_\beta u'$ and u' is α-equivalent to u.

Proof. If t solves $xw_1 \dots w_k = u$ then $tw_1 \dots w_k =_{\beta\eta} u$ and the result therefore follows from Fact 2 because the terms $t, w_1, \dots, w_k, u$ are in lnf. □

Interpolation equations appear as a component in Dowek's proof of decidability of 3rd-order matching (Dowek 1994); they are used by Padovani alongside interpolation disequations to characterise observational equivalence between terms (Padovani 1996; Padovani 1995; Padovani 2000). An interpolation disequation has the form $xw_1 \dots w_k \neq u$ and is solved by t of correct type and in lnf if $tw_1 \dots w_n \neq_{\beta\eta} u$. At each type observational equivalence has finite index; decidability of observational equivalence and its representatives implies decidability of matching (Padovani 2000). Via this route, Padovani proves decidability of 4th-order matching (including β-matching).

Conceptually, interpolation is simpler than matching because there is a single variable x that appears at the head of the equation. If $v = u$ is a matching problem with free variables $x_1 : A_1, \dots, x_n : A_n$ then its *associated* interpolation equation is $x(\lambda x_1 \dots x_n.v) = u$ where $x : ((A_1, \dots, A_n, \mathbf{0}), \mathbf{0})$. This appears to raise order by 2 as with reduction of matching to pairs of interpolation equations in (Schubert 1997). However, we only need to consider potential solution terms (in lnf of the right type) $\lambda z.zt_1 \dots t_n$ where each $t_i : A_i$ is closed and so cannot contain z: we say that such terms are *canonical* solutions.

Proposition 5. A matching problem has a solution iff its associated interpolation equation has a canonical solution.

Proof. Assume $v = u$ is a matching problem, $x_1 : A_1, \ldots, x_n : A_n$ are the free variables that occur in v and $x(\lambda x_1 \ldots x_n.v) = u$ is its associated interpolation problem. If the matching problem has solution $t_1, \ldots, t_n$ then it follows that $v\{t_1/x_1, \ldots, t_n/x_n\} =_{\beta\eta} u$. Therefore, $\lambda z.zt_1 \ldots t_n$ is a canonical solution to the associated interpolation equation because $(\lambda z.zt_1 \ldots t_n)(\lambda x_1 \ldots x_n.v) \rightarrow_\beta (\lambda x_1 \ldots x_n.v)t_1 \ldots t_n \rightarrow^*_\beta v\{t_1/x_1, \ldots, t_n/x_n\}$. Conversely, if $\lambda z.zt_1 \ldots t_n$ is a canonical solution to the equation $x(\lambda x_1 \ldots x_n.v) = u$ then $v\{t_1/x_1, \ldots, t_n/x_n\} =_{\beta\eta} u$ and, so, the closed terms $t_1, \ldots, t_n$ solve the matching problem $v = u$. □

Similarly, a β-matching problem $v = u$ also has an associated β-interpolation equation $x(\lambda x_1 \ldots x_n.v) = u$; and the problem has a solution iff the associated interpolation equation has a canonical solution $\lambda z.zt_1 \ldots t_n$ in β-normal form. Although Proposition 5 implies that interpolation equations need only have one argument, we forgo this restriction in what follows.

Example 5. The matching problem $x_1(\lambda z.x_1(\lambda z'.za)) = a$ from (Comon and Jurski 1997) is 4th-order when $z, z' : (\mathbf{0}, \mathbf{0})$ and $x_1 : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$. The associated interpolation problem is $xw = a$ where $w = \lambda x_1.x_1(\lambda z.x_1(\lambda z'.za))$ and $x : (((((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0}), \mathbf{0}), \mathbf{0})$. A canonical solution has the form $t = \lambda x.x(\lambda y.y(\lambda y_1 \ldots y(\lambda y_k.s)\ldots))$ where s is the constant a or one of the variables y_j, $1 \leq j \leq k$. To see this assume $w_1 = \lambda y.y(\lambda y_1 \ldots y(\lambda y_k.s)\ldots)$: so, $tw \rightarrow_\beta ww_1 \rightarrow_\beta w_1(\lambda z.w_1(\lambda z'.za)) \rightarrow^*_\beta w_1(\lambda z.za) \rightarrow^*_\beta a$. Conversely, assume $\lambda x.xt_1$ is a canonical solution to $xw = a$; so, $wt_1 \rightarrow^*_\beta a$ and, therefore, $t_1(\lambda z.t_1(\lambda z'.za)) \rightarrow^*_\beta a$ where $t_1 : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$ is $\lambda y.s'$ and s' is the constant $a : \mathbf{0}$ or $y(\lambda y_i.s'')$ and s'' is $a : \mathbf{0}$, a variable y_i or of the form $y(\lambda y_j.s'')$. □

It is worthwhile in the light of the undecidability of β-matching to try to distinguish between it and matching. The following is an illustrative example which we shall return to.

Example 6. Consider the following 3rd-order interpolation equation with two arguments $x(\lambda y_1y_2.y_2)a = a$ where $x : ((\mathbf{0}, \mathbf{0}, \mathbf{0}), \mathbf{0}, \mathbf{0})$. As a β-matching problem, a solution is the term $t = \lambda x_1.x_1b$ where $x_1 : (\mathbf{0}, \mathbf{0}, \mathbf{0})$ and $x_1b : (\mathbf{0}, \mathbf{0})$ because $t(\lambda y_1y_2.y_2)a \rightarrow_\beta (\lambda y_1y_2.y_2)ba \rightarrow^*_\beta a$. □

We now restrict which constants can occur in a potential solution term to an interpolation equation.

Definition 10. If E is the interpolation equation $xw_1 \ldots w_k = u$ then C_E is the set of constants that occur in the goal u together with one fresh constant $b : \mathbf{0}$ (that does not occur in u).

Fact 6. If E is an interpolation equation of type A, t solves E and $t \in T_A(C)$ then there is a $t' \in T_A(C_E)$ such that t' solves E.

In general the goal u of an interpolation equation may contain bound variables: for instance, $x(\lambda z.z) = h(\lambda x_1x_2x_3.x_1x_3)a$ has order 3 where x has type $((\mathbf{0}, \mathbf{0}), \mathbf{0})$ and the constant $h : (((\mathbf{0}, \mathbf{0}), \mathbf{0}, \mathbf{0}, \mathbf{0}), \mathbf{0}, \mathbf{0})$ assuming $x_2 : \mathbf{0}$. An example solution is the term

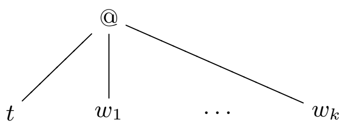


Fig. 1. An interpolation tree

$\lambda y.y(y(h(\lambda x z_1 z_2.x(y z_2))(ya)))$. For convenience, as it simplifies the presentation considerably, we restrict ourselves to the case where the goal u does not contain bound variables. A consequence is that Proposition 4 can be slightly strengthened: any solution t of $xw_1 \dots w_k = u$ can be chosen (up to α-equivalence) so that $tw_1 \dots w_k \rightarrow_\beta^* u$.

Definition 11. Assume $u : \mathbf{0}$ is closed, in lnf and does not contain bound variables.

1 The set of subterms of u, $\mathrm{Sub}(u)$, is: if $u = a : \mathbf{0}$ then $\mathrm{Sub}(u) = \{u\}$ and if $u = fu_1 \dots u_m$ then $\mathrm{Sub}(u) = \{u\} \cup \bigcup_{1 \le i \le m} \mathrm{Sub}(u_i)$.
2 The depth of u, $\mathrm{depth}(u)$, is: if $u = a : \mathbf{0}$ then $\mathrm{depth}(u) = 1$ and if $u = fu_1 \dots u_m$ then $\mathrm{depth}(u) = 1 + \max\{\mathrm{depth}(u_1), \dots, \mathrm{depth}(u_m)\}$.
3 The number of branches of u, $\mathrm{branch}(u)$, is: if $u = a : \mathbf{0}$ then $\mathrm{branch}(u) = 1$ and if $u = fu_1 \dots u_m$ then $\mathrm{branch}(u) = \mathrm{branch}(u_1) + \dots + \mathrm{branch}(u_m)$.

3. Tree automata and interpolation

3.1. *Terms as trees*

Given a potential solution term t (of the right type and in lnf) to an interpolation equation E, $xw_1 \dots w_k = u$, there is the tree in Figure 1. If $x : (A_1, \dots, A_k, \mathbf{0})$ then the explicit application operator $@ : ((A_1, \dots, A_k, \mathbf{0}), A_1, \dots, A_k, \mathbf{0})$ has its expected meaning $\lambda z_0 z_1 \dots z_k.z_0 z_1 \dots z_k$; so $@tw_1 \dots w_k \rightarrow_\beta^* tw_1 \dots w_k$.

An objective is to understand the reduction of $tw_1 \dots w_k$ to normal form by examining the tree in Figure 1, avoiding the use of substitution (and β-reduction). In fact we will present different ways of achieving this using automata and games. (Type checking with intersection types inspired by the work in (Kobayashi and Ong 2009) offers a third way which we do not expand on here.) However, only the method employing games leads to a proof of decidability of matching, as we shall see.

The terms $t, w_1, \dots, w_k$ in Figure 1 are represented as a special kind of tree (that we call *binding* trees) with dummy lambdas and an explicit binding relation. A term of the form $y : \mathbf{0}$ or $a : \mathbf{0}$ is represented as a tree with a single node labelled y or a. In the case of $u\, v_1 \dots v_n$ when u is a variable or a constant of type $(A_1, \dots, A_n, \mathbf{0})$ including @, we assume that a dummy lambda with the empty sequence of variables is placed directly above any subterm $v_i : A_i = \mathbf{0}$ in its tree representation. With this understanding, the tree for $u\, v_1 \dots v_n$ consists of a root node labelled u and n-successor trees representing $v_1, \dots, v_n$. We also use the standard vector abbreviation $\lambda\overline{y}$ for $\lambda y_1 \dots y_m$ for some $m \ge 0$, so $\overline{y}$ is possibly the empty sequence of variables in the case of a dummy lambda. The tree representation of $\lambda\overline{y}.t'$ consists of a root node labelled $\lambda\overline{y}$ with a single successor tree for $t' : \mathbf{0}$. (Thus, we do not use Böhm trees.) Therefore, in any interpolation tree, as in Figure 1, a node that is at even depth is labelled with a $\lambda\overline{y}$ and a node that is at odd depth is labelled with @, a constant or a variable.

The other elaboration is that we assume an extra binary relation $\downarrow$ between nodes in a tree that represents *binding*; that is, between a node labelled $\lambda y_1 \dots y_n$ and a node labelled y_j (that it binds). We assume the following convention for binders and variables. A binder $\lambda\overline{y}$ is such that either $\overline{y}$ is empty and therefore is a dummy lambda and cannot bind a variable occurrence or $\overline{y} = y_1 \dots y_k$ and $\lambda\overline{y}$ can only then bind variable occurrences of the form y_i, $1 \leq i \leq k$. Consequently, in the binding tree representation if $n \downarrow m$ and n is labelled $\lambda y_1 \dots y_k$ then m is labelled y_i for some $i : 1 \leq i \leq k$ (and node m must occur somwhere below n).

Example 7. An example of the tree representation of an interpolation equation with a solution term is Figure 2; its equation is in Example 8. We have identified each node with a unique integer. There are dummy lambdas such as at nodes (5) and (21). The binding relation is given by pairs such as (1) $\downarrow$ (2), (1) $\downarrow$ (4), (17) $\downarrow$ (18) and (19) $\downarrow$ (22). All nodes such as (1), (7) and (23) at even depth are labelled with a lambda. Further examples of interpolation trees with solution terms are Figures 6 and 8. $\square$

We provide a formal definition of a binding tree constructed from a finite alphabet $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ where Σ_1 are the binders, Σ_2 the bound variables and Σ_3 the remaining symbols. We adopt standard tree notation: if n is a node with k successors then these successors are $n1, \dots, nk$ (in order from left to right).

Definition 12. Assume Σ is a finite graded alphabet where each element $s \in \Sigma$ has an arity $\mathrm{ar}(s) \geq 0$. Moreover, Σ consists of three disjoint sets Σ_1 that are the binders which have arity 1, Σ_2 are (the bound) variables and Σ_3 are the remaining symbols. A binding Σ-*tree* is a finite tree where each node is labelled with an element of Σ together with a binary relation $\downarrow$ (representing binding). If node n in the tree is labelled with s and $\mathrm{ar}(s) = k$ then n has precisely k successors in the tree, the nodes $n1, \dots, nk$. Also, if a node m is labelled with a variable in Σ_2 then there is a unique node n labelled with a binder occurring above m in the tree such that $n \downarrow m$. For ease of exposition we also assume the following restrictions on binding Σ-trees: if node n is labelled with a binder then $n1$ is labelled with an element of $\Sigma_2 \cup \Sigma_3$ and if n is labelled with an element of $\Sigma_2 \cup \Sigma_3$ and ni is a successor then it is labelled with a binder.

The tree representation of an interpolation equation obeys the regulations for a binding tree. However, in actual examples such as Figure 2 we use integers to identify tree nodes instead of words.

3.2. *Tree automata*

Given an interpolation equation E of type A, $xw_1 \dots w_k = u$, is there a method for generating or recognising its set of solutions? Here is a naive technique: relative to an enumeration $t_1, t_2, \dots$ of terms in $T_A(C_E)$ we know that t_i solves E iff $t_i w_1 \dots w_k \rightarrow^*_\beta u$; so, it is enough to calculate the normal form of $t_i w_1 \dots w_k$ for each t_i in turn. This is only a decision procedure when $T_A(C_E)$ is a finite set of terms (such as for a fixed k if the order of A is at most 2 and a term contains at most k constants); otherwise, it is a semi-decision procedure.

A systematic approach to solving problems over trees is to show that their solutions are recognised by an appropriate *tree* automaton: there is an automaton that accepts a tree if, and only if, it solves the problem (Comon et als 2002). This is a decision procedure for the problem when the non-emptiness question for the automaton is decidable; that is, whether it accepts at least one tree. In this section, we investigate the application of classical bottom-up tree automata to 4th-order interpolation equations described in (Comon and Jurski 1997). (In fact, they prove the more general result that the set of solutions of a 4th-order matching problem, including β-matching, is recognisable by a tree automaton.)

First, we define bottom-up tree automata that operate on Σ-trees which are finite trees where each node is labelled with an element of Σ (Comon et als 2002).

Definition 13. Assume Σ is a finite graded alphabet where each element $s \in \Sigma$ has an arity $\mathrm{ar}(s) \geq 0$. 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 $\mathrm{ar}(s) = k$ then n has precisely k successors $n1, \ldots, nk$ in t. Let T_Σ be the set of Σ-trees.

Binding trees as in Definition 12 are a particular case of Σ-trees.

Definition 14. 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 of the form $sq_1 \ldots q_k \Rightarrow q$ where $k \geq 0$, $s \in \Sigma$, $\mathrm{ar}(s) = k$ and $q_1, \ldots, q_k, q \in Q$.

Definition 15. 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 $s \in \Sigma$, $\mathrm{ar}(s) = k \geq 0$, $sq_1 \ldots q_k \Rightarrow q \in \Delta$ and $q_1, \ldots, q_k$ label $n1, \ldots, nk$ the k successors of n, 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 has a finite set of states and transitions (which can be nondeterministic). A run of A on the Σ-tree t is a labelling of it with elements of Q that starts at the leaves (hence "bottom-up"). If n in t is a leaf then it is labelled with an $s \in \Sigma$ with $\mathrm{ar}(s) = 0$: 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 $n1, \ldots, nk$ are labelled with the states $q_1, \ldots, q_k$ and $sq_1 \ldots q_k \Rightarrow q \in \Delta$ 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 key properties of tree automata (Comon et als 2002).

Fact 7. Assume A and A' are Σ-tree automata.

1 The non-emptiness problem for A (is $\mathsf{T}_\Sigma(\mathsf{A}) \neq \emptyset$?) is decidable in linear time.
2 There is a Σ-tree automaton $\overline{\mathsf{A}}$ such that $\mathsf{T}_\Sigma(\overline{\mathsf{A}}) = \mathsf{T}_\Sigma - \mathsf{T}_\Sigma(\mathsf{A})$.
3 There is a Σ-tree automaton A'' such that $\mathsf{T}_\Sigma(\mathsf{A}'') = \mathsf{T}_\Sigma(\mathsf{A}) \cap \mathsf{T}_\Sigma(\mathsf{A}')$.

Whether a tree automaton accepts at least one tree is decidable. Its proof uses a small tree property: if an automaton accepts a tree then it accepts a tree whose depth is bounded by the number of states of the automaton. The other properties have the consequence that the sets of Σ-trees that are tree recognisable are regular (closed under the usual boolean operations).

If we can design a tree automaton to recognise exactly the solutions of an interpolation equation then matching is decidable: there is a slight gap here; more precisely, we need the slightly stronger, and easily attainable, requirement of decidability of non-emptiness of the tree automaton with respect to canonical solutions, see Proposition 5.

We start with a 3rd-order interpolation equation E, $xw_1 \dots w_k = u$. To define a tree automaton that accepts its solutions, first we need to isolate a finite alphabet Σ. A potential solution term has the form $\lambda x_1 \dots x_k.s$ where s is given by the following simple grammar

$$s ::= fs_1 \dots s_m \mid x_i s_1 \dots s_m$$

where f range over the constants in C_E, x_i over $\{x_1, \dots, x_k\}$ and $m \geq 0$. A potential solution term is, therefore, built from a fixed finite alphabet; the only constants allowed in s belong to C_E and the only free variables are the x_is. The set of terms allowed by this grammar may not be finite because of arbitrary embeddings of constants and variables, such as with $\lambda x_1 \dots x_k.x_1(x_1 \dots (x_1 x_2) \dots)$.

Example 8. Consider the interpolation equation $xw_1w_2 = faa$ from (Comon and Jurski 1997) where w_1 is $\lambda y_1 y_2.y_1$, w_2 is $\lambda y_3.fy_3y_3$, $y_1, y_2, y_3 : \mathbf{0}$, $f : (\mathbf{0}, \mathbf{0}, \mathbf{0})$ and $x : ((\mathbf{0}, \mathbf{0}, \mathbf{0}), (\mathbf{0}, \mathbf{0}), \mathbf{0})$ is 3rd-order. The alphabet Σ is $\{a, b, f, x_1, x_2, \lambda, \lambda x_1 x_2\}$; f and x_1 have arity 2, λ, $\lambda x_1 x_2$ and x_2 have arity 1, and a, b arity 0. A solution term $t = \lambda x_1 x_2.x_1(x_2(x_1(x_1ab)b))b$ is shown in Figure 2: $tw_1w_2 \rightarrow^*_\beta w_1(w_2(w_1(w_1ab)b))b \rightarrow^*_\beta w_1(w_2a)b \rightarrow_\beta w_1(faa)b \rightarrow_\beta faa$. □

We describe how to define a simple tree automaton that captures all solution terms of E (represented as trees). Consider any node n within a solution term t that is labelled with a variable or a constant. The subterm rooted at that node has type $\mathbf{0}$; it may contain occurrences of free variables belonging to $\{x_1, \dots, x_k\}$. The first issue is whether this node in t contributes to the normal form of $tw_1 \dots w_k$. If it does not then it is said to be *inaccessible* (Padovani 2000): that is, replacing the subtree rooted at n with any other term of type $\mathbf{0}$, and in particular the constant $b : \mathbf{0}$ (which does not occur in u), will not change the normal form; so, adopting the notation $t[b/n]$ meaning t where the subtree at node n is replaced with the single node tree labelled b, $(t[b/n])w_1 \dots w_k$ has the same normal form, u, as $tw_1 \dots w_k$. Otherwise n does contribute and so the node is then said to be accessible; next, we examine its *evaluation* (Padovani 2000; Comon and Jurski 1997), the normal form of $t'\{w_1/x_1, \dots, w_k/x_k\}$ when t' is the subterm rooted at n which must be a subterm of u, an element of $\mathrm{Sub}(u)$. For instance, nodes (12), (14) and (16) of Figure 2 are inaccessible. Node (6) is accessible; it is the root of the subterm $x_1(x_1ab)b$ whose evaluation is the normal form of $x_1(x_1ab)b\{\lambda y_1 y_2.y_1/x_1, \lambda y_3.fy_3y_3/x_2\}$ which is a. In contrast, the evaluation of (2) is faa. This analysis is extended to nodes of t that are labelled with dummy lambdas and the root node labelled $\lambda x_1 \dots x_k$; such a node

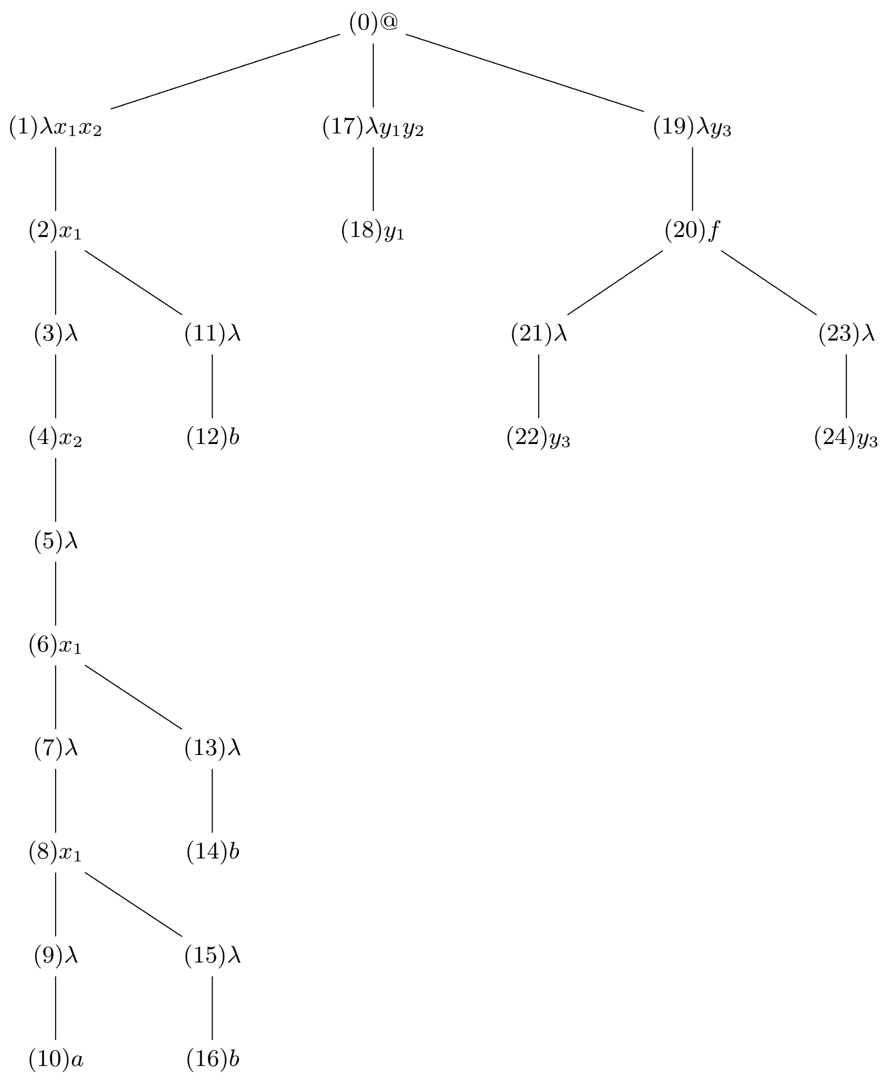


Fig. 2. A 3rd-order example

n is inaccessible if its immediate successor $n1$ is inaccessible; otherwise, its evaluation is the value at $n1$ prefaced with the lambda at n.

States of the tree automaton for E are defined from these values. We need the elements $\mathrm{Sub}(u)$ where u is the goal and one other state for inaccessible nodes plus their versions for lambda nodes.

$$Q = \{[e], [\lambda e] \mid e \in \mathrm{Sub}(u) \cup \{-\}\} \cup \{[\lambda x_1 \dots x_k u]\}$$

The states $[-]$ and $[\lambda -]$, "dont care" states, are for inaccessible nodes; the states $[u']$ and $[\lambda u']$ are for accessible nodes whose evaluation or successor's evaluation is $u' \in \mathrm{Sub}(u)$ and $[\lambda x_1 \dots x_k u]$ is the only final state of the automaton for the root node.

The final component of the automaton is the transitions. The intention is that t is a solution of E if there is a labelling of t with states such that its root is labelled with the final state $[\lambda x_1 \dots x_k u]$. Transitions are of the form $sq_1 \dots q_k \Rightarrow q$ where $s \in \Sigma$ has arity k and $q_1, \dots, q_k, q$ are states. First when s is a constant or a λ there are the following transitions.

$$\begin{array}{ll} f[\lambda u_1] \dots [\lambda u_m] \Rightarrow [f u_1 \dots u_m] & m \geq 0 \\ f[\lambda -] \dots [\lambda -] \Rightarrow [-] & \\ \lambda[u'] \Rightarrow [\lambda u'] & \\ \lambda[-] \Rightarrow [\lambda -] & \\ \lambda x_1 \dots x_k[u] \Rightarrow [\lambda x_1 \dots x_k u] & \end{array}$$

Transitions are of two kinds, for accessible and inaccessible nodes. If $m = 0$ then these transitions have the form $a \Rightarrow [a]$ and $a \Rightarrow [-]$. Next we examine the transitions for a node n labelled with a variable $x_i \in \{x_1, \dots, x_k\}$. If n is accessible then its evaluation employs w_i for x_i; therefore, we are interested in the normal form of $w_i e_1 \dots e_m$ when for each j state $[\lambda e_j]$ labels nj; $e_j \in \mathrm{Sub}(u) \cup \{-\}$; so, if e_j is dont care then it does not contribute to the normal form.

$$\begin{array}{ll} x_i[\lambda e_1] \dots [\lambda e_m] \Rightarrow [u'] & m \geq 0 \text{ and } w_i e_1 \dots e_m \to_\beta^* u' \in \mathrm{Sub}(u) \\ x_i[\lambda -] \dots [\lambda -] \Rightarrow [-] & \end{array}$$

Example 9. The automaton for the equation in Example 8 has, as we have seen, alphabet $\Sigma = \{a, b, f, x_1, x_2, \lambda, \lambda x_1 x_2\}$. Q is $\{[faa], [\lambda faa], [a], [\lambda a], [-], [\lambda -], [\lambda x_1 x_2 faa]\}$ and its last element is the final state. Transitions are as follows where e ranges over $\{a, faa, -\}$.

$$\begin{array}{ll} a \Rightarrow [a] & a \Rightarrow [-] \quad b \Rightarrow [-] \\ f[\lambda a][\lambda a] \Rightarrow [faa] & f[\lambda -][\lambda -] \Rightarrow [-] \\ x_1[\lambda e][\lambda -] \Rightarrow [e] & \\ x_2[\lambda a] \Rightarrow [faa] & x_2[\lambda -] \Rightarrow [-] \\ \lambda[e] \Rightarrow [\lambda e] & \\ \lambda x_1 x_2[faa] \Rightarrow [\lambda x_1 x_2 faa] & \end{array}$$

In the rule for x_1, the second argument of $w_1 e_1 e_2$ does not contribute to normal form because w_1 is $\lambda y_1 y_2 . y_1$. The solution term in Figure 2 is accepted as follows. For the leaves node (10) is labelled with $[a]$ and (12), (14), (16) with $[-]$. Using transitions $\lambda[e] \Rightarrow [\lambda e]$, node (9) is labelled with $[\lambda a]$ and (15), (13), (11) with $[\lambda -]$. The rule $x_1[\lambda a][\lambda -] \Rightarrow [a]$

allows (8) to be labelled $[a]$; so (7) can be labelled $[\lambda a]$; therefore, node (6) can also be labelled $[a]$ and, therefore, (5) can be labelled $[\lambda a]$. The transition $x_2[\lambda a] \Rightarrow [faa]$ is now applied and (4) is labelled $[faa]$ and (3) $[\lambda faa]$. Using $x_1[\lambda faa][\lambda -] \Rightarrow [faa]$ node (2) is labelled $[faa]$ and, therefore, (1) is labelled with the final state. We leave the reader to verify that this automaton recognises the solutions of the equation in Example 8. $\square$

Extending the construction to a 4th-order interpolation equation E requires care with variables (Comon and Jurski 1997). If E, $xw_1 \dots w_k = u$, is 4th-order then a solution term has the form $\lambda x_1 \dots x_k.s$ where s is given by the following grammar.

$$\begin{aligned} s &::= z_j \mid fs_1 \dots s_m \mid x_i v_1 \dots v_m \\ v &::= s \mid \lambda z_1 \dots z_m.s \end{aligned}$$

Again f ranges over C_E, x_i over $\{x_1, \dots, x_k\}$ and $m \geq 0$: a variable x_i may take as argument a term of the form $\lambda z_1 \dots z_m.s$ where the z_js are of ground type. Therefore, with this grammar we can write 4th-order terms which use arbitrarily many variables such as

$$\lambda x_1 x_2.x_1(\lambda z_1.x_1(\lambda z_2.x_1 \dots x_1(\lambda z_n.x_2(x_2 \dots (x_2(x_2 z_1 z_2)z_3)\dots)z_n)\dots))$$

where $z_i : \mathbf{0}$ for each i, $x_1 : ((\mathbf{0},\mathbf{0}),\mathbf{0})$ and $x_2 : (\mathbf{0},\mathbf{0},\mathbf{0})$.

However, there is an upper bound on the number of variables that are needed to label the accessible nodes of any solution term. Any occurrence of a free variable z_j in a subterm has ground type and, therefore, occurs at a leaf of the tree. If we now examine an accessible node of a solution term that is labelled with a constant or a variable then the term t' rooted there has gound type. Its evaluation, the normal form of $t'\{w_1/x_1, \dots, w_k/x_k\}$, may contain free variables but only z_js of ground type: therefore, it is either an element of $\mathrm{Sub}(u)$ or what happens to an element of $\mathrm{Sub}(u)$ when subterms of it are replaced by z_js. Consequently, if $t'\{w_1/x_1, \dots, w_k/x_k\}$ contains a large number of free variable occurrences then they are mostly at inaccessible nodes. There is, therefore, an upper bound on the number of accessible leaves labelled with a free variable z_j in any subterm of a solution term; this bound is $|u|$, the size of u. Consequently, by reusing variables there is an overall upper bound on the number of z_js needed in the abstract syntax for a solution term $\lambda x_1 \dots x_k.s$, as described above, which leads to a finite alphabet Σ for the tree automaton.

The states of the automaton consist of elements $[e]$, $[\lambda e]$, $[\lambda z_1 \dots z_m e]$ and the final state $[\lambda x_1 \dots x_k u]$ where e not only ranges over subterms of u and $-$ but also over subterms of u when their subterms are replaced with z_js. The full definition of the transition relation of the automaton is left as an exercise for the reader; we illustrate it with an example.

Example 10. The equation $x\,w_1 = ga$ is 4th-order where $w_1 = \lambda y_1 y_2.y_1 y_2$, $g : (\mathbf{0},\mathbf{0})$, $a : \mathbf{0}$ and $x : (((\mathbf{0},\mathbf{0}),\mathbf{0},\mathbf{0}),\mathbf{0})$. Any solution term with $b : \mathbf{0}$ at an inaccessible non-lambda node has the form $\lambda x_1.s$ where s is constrained by the following finite alphabet.

$$s ::= a \mid b \mid z \mid gs \mid x_1(\lambda z.s)s \mid x_1(\lambda y.s)s$$

So, Σ is $\{a, b, z, g, x_1, \lambda, \lambda z, \lambda y, \lambda x_1\}$. The states of the automaton are as follows where

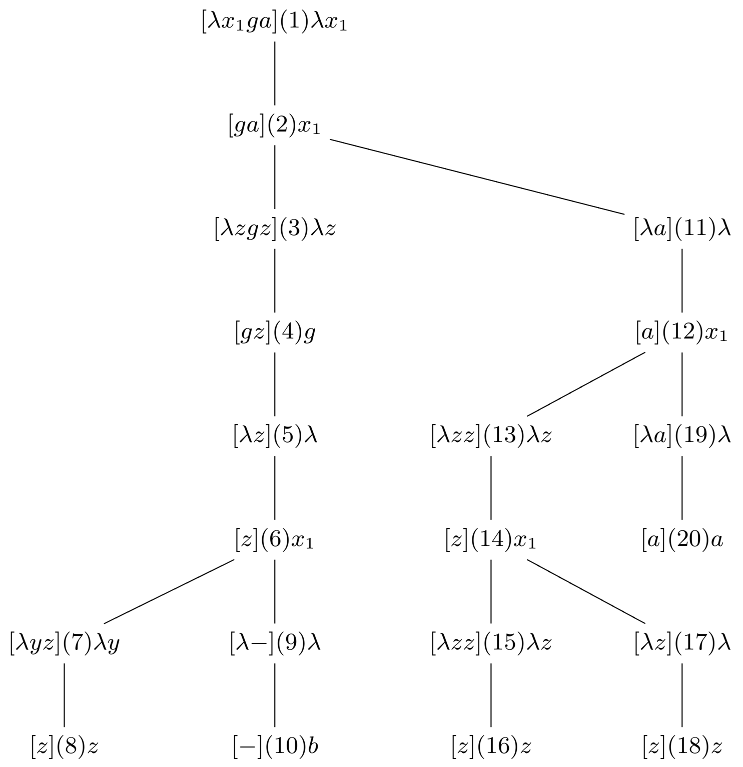


Fig. 3. A successful run of the automaton of Example 10

the final state is $[\lambda x_1 ga]$.

$$Q = \{[e], [\lambda e], [\lambda ye], [\lambda ze], [\lambda x_1 ga] \mid e \in \{-, z, a, gz, ga\}\}$$

Below are the transitions.

$$\begin{array}{ll}
a \Rightarrow [a] & a \Rightarrow [-] \qquad b \Rightarrow [-] \\
z \Rightarrow [z] & z \Rightarrow [-] \\
g[\lambda e] \Rightarrow [ge] \;\; e \in \{a, z\} & \lambda[e] \Rightarrow [\lambda e] \;\; e \in \{-, a, z, ga, gz\} \\
\lambda y[e] \Rightarrow [\lambda ye] \;\; e \in \{a, z, ga, gz\} & \lambda z[e] \Rightarrow [\lambda ze] e \in \{a, z, ga, gz\} \\
\lambda x_1[ga] \Rightarrow [\lambda x_1 ga] & x_1[\lambda ye][\lambda e'] \Rightarrow [e] \;\; e \in \{a, z, ga, gz\} \\
x_1[\lambda ze][\lambda e'] \Rightarrow [e] \;\; e \in \{a, ga\} & x_1[\lambda zz][\lambda e] \Rightarrow [e] \;\; e \in \{a, z, ga, gz\} \\
x_1[\lambda zgz][\lambda e] \Rightarrow [ge] \;\; e \in \{z, a\} &
\end{array}$$

The interesting transitions cover the cases for x_1 whose interpretation is w_1: therefore, for instance, $w_1(\lambda y.e)e' \rightarrow^*_\beta e$ and $w_1(\lambda z.z)e \rightarrow^*_\beta e$. An example solution term is $\lambda x_1.x_1(\lambda z.gx_1(\lambda y.z)b)(x_1(\lambda z.x_1(\lambda z.z)z)a)$. The successful run of the automaton on its tree is presented in Figure 3. In contrast, the automaton will not accept the term when the label at node (7) of Figure 3 is changed to λz; then we would have state $[\lambda zz]$ at node (7) and $[\lambda -]$ at node (9); now the run of the automaton becomes stuck as there is no transition that allows (6) to be labelled with a state. $\square$

The following result is now clear, see (Comon and Jurski 1997).

Fact 8. The tree automaton associated with a 4th-order interpolation equation E accepts the term t iff t solves E.

There are serious problems with extending the definition of a tree automaton associated with an interpolation equation beyond 4th-order. If E, $xw_1 \dots w_k = u$, is 5th-order, then a solution term has the form $\lambda x_1 \dots x_k.s$ where s is given by the following grammar

$$\begin{aligned} s &::= z_j s_1 \dots s_m \mid f s_1 \dots s_m \mid x_i v_1 \dots v_m \\ v &::= s \mid \lambda z_1 \dots z_m.s \end{aligned}$$

where f ranges over C_E, x_i over $\{x_1, \dots, x_k\}$ and $m \geq 0$. Now it is possible to produce terms whose accessible nodes involve subterms containing arbitrary many different free z_j variables because they can be 2nd-order. A first problem is that the alphabet Σ for the automaton need not be finite. A second problem is related; how to restrict the state set Q of the automaton to be finite. Even if the syntax is restricted to be finite there need not be an upper bound on the number of different evaluations of accessible nodes because the same free variable z_j may occur embedded multiple times. Comon and Jurski illustrate the problems with the following example (Comon and Jurski 1997).

Example 11. The equation $x\,\lambda yz.y(\lambda z'.zz') = a$ where $z' : \mathbf{0}$, $z : (\mathbf{0}, \mathbf{0})$, $y : ((\mathbf{0}, \mathbf{0}), \mathbf{0})$ and $x : ((((\mathbf{0}, \mathbf{0}), \mathbf{0}), (\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$ is 5th-order. Solutions include the terms

$$\lambda x_1.x_1(\lambda z_1.x_1(\dots x_1(\lambda z_n.z_{i_1}(z_{i_2}(\dots(z_{i_k}a)\dots)))u_n \dots)u_2)u_1$$

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. The point is that the z_i variables are 2nd-order; and so they can be "stacked". To recognise this family of terms requires an unbounded alphabet. Moreover, even if we do restrict the number of different z_j's: for instance, assume that they are all z_n and consider the evaluation at the position of the first z_n which is $z_n(z_n(\dots(z_n a)\dots))$.

Similarly, the monster type M, $((((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0}), \mathbf{0}, \mathbf{0})$, has, as we have seen, order 5. Assume $x : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$, $y : \mathbf{0}$ and $z_i : (\mathbf{0}, \mathbf{0})$ for $i \geq 1$. The following family of terms in lnf $\lambda xy.x(\lambda z_1.x(\lambda z_2 \dots (\lambda z_n.z_n(z_{n-1}(\dots z_1(y))\dots))\dots))$ for $n \geq 0$ belong to $T_M(\emptyset)$. To write down this subset of terms requires an alphabet of unbounded size. □

A solution to the problem of a potentially infinite alphabet is to represent terms as *binding* trees as defined in Section 3.1. For instance, there is a straightforward representation of the family of terms of type M in Example 11 as binding trees (with dummy lambdas). Nodes are labelled with binders λxy, λz, λ, or with variables x, z of arity 1 and y of arity 0: in linear form

$$\lambda xy.x(\lambda z.x(\lambda z \dots x(\lambda z.z(\lambda.z(\dots\lambda.z(\lambda.y))\dots))\dots))$$

where there is an edge $\downarrow$ from the node labelled λxy to each node labelled x or y, and an edge $\downarrow$ from the first node labelled λz to the last node labelled z, and so on.

Fact 9. For any type A and finite C, there is a finite alphabet Σ such that every $t \in T_A(C)$ up to α-equivalence is represented as a binding Σ-tree.

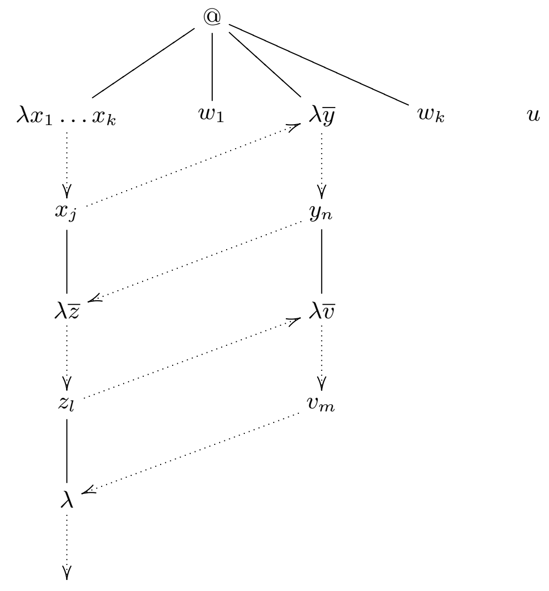


Fig. 4. Game-theoretic view

However this does not help with the issue of how to define an automaton that only involves a finite number of states. To understand this better, we need a finer analysis than that of the evaluation of an accessible node. Indeed, we shall now present a finer static analysis of β-reduction using games.

4. Games and interpolation

We now describe a game theoretic characterisation of whether t is a solution to an interpolation equation $x\,w_1 \ldots w_k = u$. The game board is the interpolation tree of Figure 1. The idea is as with game semantics (Ong 2006), see Figure 4. Player Opponent, $\forall$, chooses a branch of u. Then, there is a finite play that starts at the root of t labelled $\lambda x_1 \ldots x_k$ which may repeatedly jump from t into a w_j and back again. At a constant $a : \mathbf{0}$ play ends. At other constants f, player Proponent, *exists*, tries to match Opponent's choice of branch. Proponent wins, when the play finishes, if the sequence of constants encountered matches the branch chosen by Opponent. Play, for example, may reach a node labelled x_j in t and then jump to the root labelled $\lambda\overline{y}$ of w_j because w_j is the jth argument that is applied to $\lambda x_1 \ldots x_k$ at the root of t, and then when at a node labelled y_n in w_j play returns to the nth successor of x_j in t, to the node labelled $\lambda\overline{z}$; play then may proceed to a node z_l and return to the lth successor of the node labelled y_n in w_j, and so on. The game models β-reduction on the fixed structure of Figure 4 without changing it using substitution. The game we now develop avoids questions, answers and justification pointers of game semantics and uses iteratively defined look-up tables as developed in (Stirling 2005; Stirling 2006; Stirling 2007).

Current position is $n[r]\theta$. The next position is by cases on the label at node n:

— @ then $n1[r]\theta'$ where $\theta' = \theta\{((n2, \dots, n(k+1)), \theta)/n1\}$

— $\lambda\overline{y}$ then $n1[r]\theta$

— $a : \mathbf{0}$ if $r = a$ then $n[\exists]\theta$ else $n[\forall]\theta$

— $f : (B_1, \dots, B_p, \mathbf{0})$ if $r = fr_1 \dots r_p$ then $\forall$ chooses $j \in \{1, \dots, p\}$ and $nj\,[r_j]\theta$ else $n[\forall]\theta$

— $y_j : \mathbf{0}$ if $m \downarrow n$ and $\theta(m) = ((m_1, \dots, m_l), \theta')$ then $m_j\,[r]\theta'$

— $y_j : (B_1, \dots, B_p, \mathbf{0})$ if $m \downarrow n$ and $\theta(m) = ((m_1, \dots, m_l), \theta')$ then $m_j\,[r]\theta''$ where $\theta'' = \theta'\{((n1, \dots, np), \theta)/m_j\}$

Fig. 5. Game moves

4.1. *Interpolation games*

Assume E is the equation $xw_1 \dots w_k = u$ and t is a potential solution term. We define the game $\mathsf{G}(t, E)$ played by one participant, player $\forall$, the *refuter* who attempts to show that t is not a solution of E. The game is played on the interpolation tree $@tw_1 \dots w_k$ of Figure 1, presented as a binding tree, Definition 12, where the alphabet is the disjoint union of Σ_1, the set of binders, Σ_2, the bound variables, Σ_3, the constants and @, and where $\downarrow$ is the binding relation between nodes.

Definition 16. N is the set of nodes of the interpolation tree $@\,tw_1 \dots w_k$ labelled with elements of $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ and S is the set $\{[\,r\,] \,|\, r \in \mathrm{Sub}(u) \cup \{\forall, \exists\}\}$ of game states. $[\forall]$ and $[\,\exists]$ are the final game states. Let N_1 be the subset of nodes N whose labels belong to Σ_1 (the binders). If s is $\lambda y_1 \dots y_p$ then its length, $|s|$, is p. 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 from $N_1 \to (\bigcup_{s \in \Sigma_1} N^{|s|} \times \bigcup_{j \leq i} \Theta_j)$.

Definition 17. A play of $\mathsf{G}(t, E)$ is a finite sequence $n_1q_1\theta_1, \dots, n_lq_l\theta_l$ of positions where each $n_i \in N$, each $q_i \in S$ with q_l a final state and each $\theta_i \in \Theta_i$ is a look-up table. For the initial position n_1 is the root of the interpolation tree labelled @, $q_1 = [\,u\,]$ where u is the goal term of E and θ_1 is the empty look-up table. Player $\forall$ wins the play if the final state is $[\forall]$ and loses it otherwise (when the final state is $[\,\exists]$).

The game $\mathsf{G}(t, E)$ appeals to a finite set of states S with elements states $[\,r\,]$, $r \in \mathrm{Sub}(u)$ where u is the goal term of E, and *final* states, $[\forall]$, winning for the refuter, and $[\,\exists]$, losing for the refuter. The central feature of a play of $\mathsf{G}(t, E)$, as depicted in Figure 4, is that repeatedly control may jump from a node of t to a node of w_j for some j and then later jump back into t. Therefore, as play proceeds, one needs an account of the meaning of free variables in subtrees. A free variable in a subtree of t is associated with a subtree of w_j for some j; similarly, a free variable in a subtree of w_j is associated with a subtree of t. This is the role of the look-up table $\theta_k \in \Theta_k$ at position $k \geq 1$. If $n \in N_1$ is a binder labelled $\lambda y_1 \dots y_p$ and $\theta_k(n)$ is defined then it has the form $((n_1, \dots, n_p), \theta_j)$ which tells us that any node m labelled y_i such that $n \downarrow m$ is associated with the subtree rooted at node n_i: that subtree may itself contain free variables, hence, the presence of a previous look-up table θ_j. Formally, as we shall see, one can view a look-up table as a substitution.

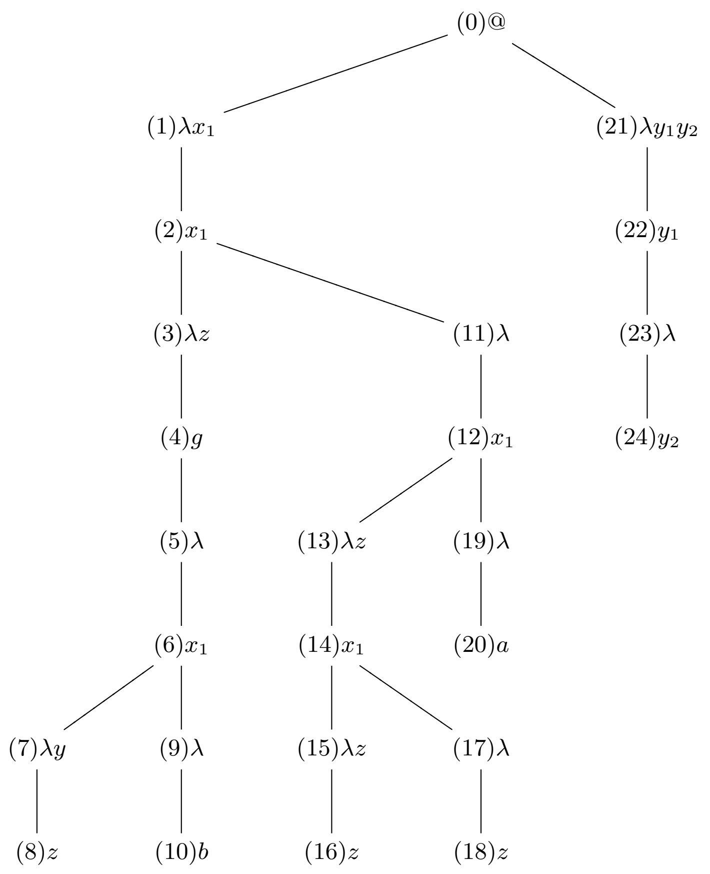


Fig. 6. The interpolation tree of Example 12

Definition 18. If the current position in a play of $\mathsf{G}(t, E)$ is $n[r]\theta$ and $[r]$ is not final then the next position is determined by a unique move in Figure 5 according to the label at node n.

At the initial node labelled @, play proceeds to its first successor labelled with $\lambda x_1 \dots x_k$ and the look-up table is updated and this binder is associated with the other successors of the root; so any free variable x_j in the subtree below $\lambda x_1 \dots x_k$ is associated with the subtree w_j rooted at the $(j+1)$th successor of the root and the empty look-up table. Later, if play reaches a node labelled x_j (bound by this initial binder) then there is a jump to the $(j+1)$th successor of the root node; it is this jumping that, thereby, simulates the substitution of w_j for x_j. Standard update notation is assumed: $\gamma\{((m1, \dots, ml), \gamma')/n\}$ is the partial function similar to γ except that $\gamma(n) = ((m1, \dots, ml), \gamma')$ where n is labelled $\lambda z_1 \dots z_l$ for some z. If play is at a node labelled $\lambda\overline{y}$, where $\overline{y}$ can be empty, then the next position is at its successor. At a node labelled with the constant $a : \mathbf{0}$, the refuter loses if the state is $[a]$ and wins otherwise. At a node labelled with a constant f

$(0)[ga]\theta_1 = \emptyset$	$(8)[a]\theta_6$	$(16)[a]\theta_{10}$
$(1)[ga]\theta_2 = \{((21), \theta_1)/1\}$	$(23)[a]\theta_3$	$(23)[a]\theta_9$
$(2)[ga]\theta_2$	$(24)[a]\theta_3$	$(24)[a]\theta_9$
$(21)[ga]\theta_3 = \{((3, 11), \theta_2)/21\}$	$(11)[a]\theta_2$	$(17)[a]\theta_8$
$(22)[ga]\theta_3$	$(12)[a]\theta_2$	$(18)[a]\theta_8$
$(3)[ga]\theta_4 = \theta_2\{((23), \theta_3)/3\}$	$(21)[a]\theta_7 = \{((13, 19), \theta_2)/21\}$	$(23)[a]\theta_7$
$(4)[ga]\theta_4$	$(22)[a]\theta_7$	$(24)[a]\theta_7$
$(5)[a]\theta_4$	$(13)[a]\theta_8 = \theta_2\{((23), \theta_7)/13\}$	$(19)[a]\theta_2$
$(6)[a]\theta_4$	$(14)[a]\theta_8$	$(20)[a]\theta_2$
$(21)[a]\theta_5 = \{((7, 9), \theta_4)/21\}$	$(21)[a]\theta_9 = \{((15, 17), \theta_8)/21\}$	$(20)[\exists]\theta_2$
$(22)[a]\theta_5$	$(22)[a]\theta_9$	
$(7)[a]\theta_6 = \theta_4\{((23), \theta_5)/7\}$	$(15)[a]\theta_{10} = \theta_2\{((23), \theta_9)/15\}$	

Fig. 7. The single play of Example 12

with arity more than 0, $\forall$ immediately wins if the state is not of the form $[fr_1 \dots r_m]$. Otherwise $\forall$ chooses a successor j and play moves to its jth successor with the new state $[r_j]$. If play is at node n labelled with variable $y_j : \mathbf{0}$ and $\theta(m) = ((m_1, \dots, m_l), \theta')$ when $m \downarrow n$ then play jumps to m_j which is labelled with a dummy lambda and θ' is then the look-up table. If n is labelled $y_j : (B_1, \dots, B_p, \mathbf{0})$ and $\theta(m) = ((m_1, \dots, m_l), \theta')$ when $m \downarrow n$ then play jumps to m_j which is labelled $\lambda z_1 \dots z_p$ for some z and the look-up table is θ' plus the entry $((n1, \dots, np), \theta)$ for m_j: besides the jump to node m_j with the look-up table θ', there is also the extra association of the ith successor of n and θ with any free occcurrence of z_i beneath m_j for each $i : 1 \leq i \leq p$; node n must have precisely p successors because of the type of y_j.

Definition 19. Player $\forall$ wins the game $\mathsf{G}(t, E)$ if there is a play that she wins; otherwise she loses the game (when she loses every play).

4.2. *Example games*

Example 12. Consider $x\,w_1 = ga$ of Example 10 where $w_1 = \lambda y_1 y_2 . y_1 y_2$. The interpolation tree with the solution term described in Example 10 is presented in Figure 6. We now examine the game for this tree which consists of the single play in Figure 7. The positions are presented in three columns; the second column follows after the first and the third after the second. Each position has the form $n[r]\theta$ where n is a node of the interpolation tree, $r \in \{ga, a, \exists, \forall\}$ and θ is a look-up table.

The initial position is at the root (0). Play descends to node (1) which is a binder; any node n such that $(1) \downarrow n$ is, therefore, associated with (the subtree at) (21) and the empty look-up table θ_1; this means that the look-up table θ_2 at this position has the single entry $\theta_2(1) = ((21), \theta_1)$: in examples, we reduce the number of brackets by abbreviating nodes such as (1) and (21) to 1 and 21 in look-up tables. Play descends to node (2) which is bound by (1); therefore, play jumps to node (21) with the look-up table θ_1 which is updated because (21) is a binder; therefore, the look-up table θ_3 in this position at (21) has the single entry $\theta_3(21) = ((3, 11), \theta_2)$ which represents that any node labelled y_1 bound by (21) is associated with (3), the first successor of (2), and look-up table θ_2, and any node labelled y_2 bound by (21) is associated with (11), the second successor of (2), and θ_2. Play descends from (21) to (22) and so jumps to (3) with θ_2 which is updated because (3) is a binder; therefore, at this position any node bound by (3) is associated with (23), the successor of (22), and the look-up table θ_3. Play descends from (3) to (4) which is labelled with a constant g; because the goal state is $[ga]$, play moves to node (5) with state $[a]$ as this is the only choice available to $\forall$.

From (5) play proceeds to (6), and, therefore, jumps to (21) because (1) $\downarrow$ (6) with look-up table θ_5 which consists of the single entry $\theta_5(21) = ((7, 9), \theta_4)$; so any node labelled y_1, bound by (21), is associated with (7), the first successor of (6), and θ_4 and any node labelled y_2 is associated with (9), the other successor of (6), and θ_4. Play proceeds to (22) and, therefore, jumps to node (7). The look-up table θ_6 at this position is $\{((21), \theta_1)/1, ((23), \theta_3)/3, ((23), \theta_5)/7\}$; it has exactly one entry for every binding node above and including (7); the nodes (1) labelled λx_1, (3) labelled λz and (7) labelled λy; a node labelled with a dummy lambda does not bind and, therefore, does not have an entry in a look-up table. Consequently, when play decends from (7) to (8) it jumps to (23) because (3) $\downarrow$ (8); the look-up table at this position is, therefore, θ_3 and so when play proceeds to (24) it jumps to (11) which is the other successor of (2); if node (8) were labelled y, play would still jump to (23) because (7) $\downarrow$ (8) and descend to (24); but then the look-up table would be θ_5 and so play would jump to (9), the other successor of (6), and descend to (10) and the refuter would win.

After play jumps to (11), it descends to (12) and, therefore, jumps to (21), proceeds to (22), jumps to (13), proceeds to (14), jumps to (21), descends to (22), jumps to (15), down to (16), jumps to (23), proceeds to (24), jumps to (17), down to (18), and, therefore, jumps to (23) and proceeds to (24) before finally jumping to (19), finishing at (20) and so refuter loses because the state is then $[a]$. □

Example 13. The equation, $x(\lambda y_1 y_2.y_1(\lambda y_3.y_2(y_1(\lambda y_4.y_3))) = h(g(h(ha)))$, is 5th-order with $x : ((((\mathbf{0}, \mathbf{0}), \mathbf{0}), (\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$ and $g, h : (\mathbf{0}, \mathbf{0})$. A solution term is presented as part of the interpolation tree in Figure 8. There is a single play on this tree which is presented in Figure 9; here $u = h(g(h(ha)))$, $u_1 = g(h(ha))$, and $u_2 = h(ha)$. The reader is invited to examine the play in detail. □

4.3. *Properties of game playing*

A key feature of the game is that terms must be in lnf. Consider what happens if we allow β-normal forms instead as in Example 6 with interpolation equation $x(\lambda y_1 y_2.y_2)a = a$ where $x : ((\mathbf{0}, \mathbf{0}, \mathbf{0}), \mathbf{0}, \mathbf{0})$. As a β-matching problem, one solution is $t = \lambda x_1.x_1 b$ and another is its lnf; these interpolation trees are illustrated in Figure 10. The tree on the left is in β-normal form whereas the tree on the right is in lnf. It is clear how play proceeds on the second tree; there is a jump from the node labelled x_1 to the node labelled $\lambda y_1 y_2$, and a return jump from the node labelled y_2 to the dummy lambda above x_2, so that at x_2 play jumps to the third successor of @ and so $\forall$ loses. However, in the first tree it is unclear how play should proceed because of incomplete information; what should

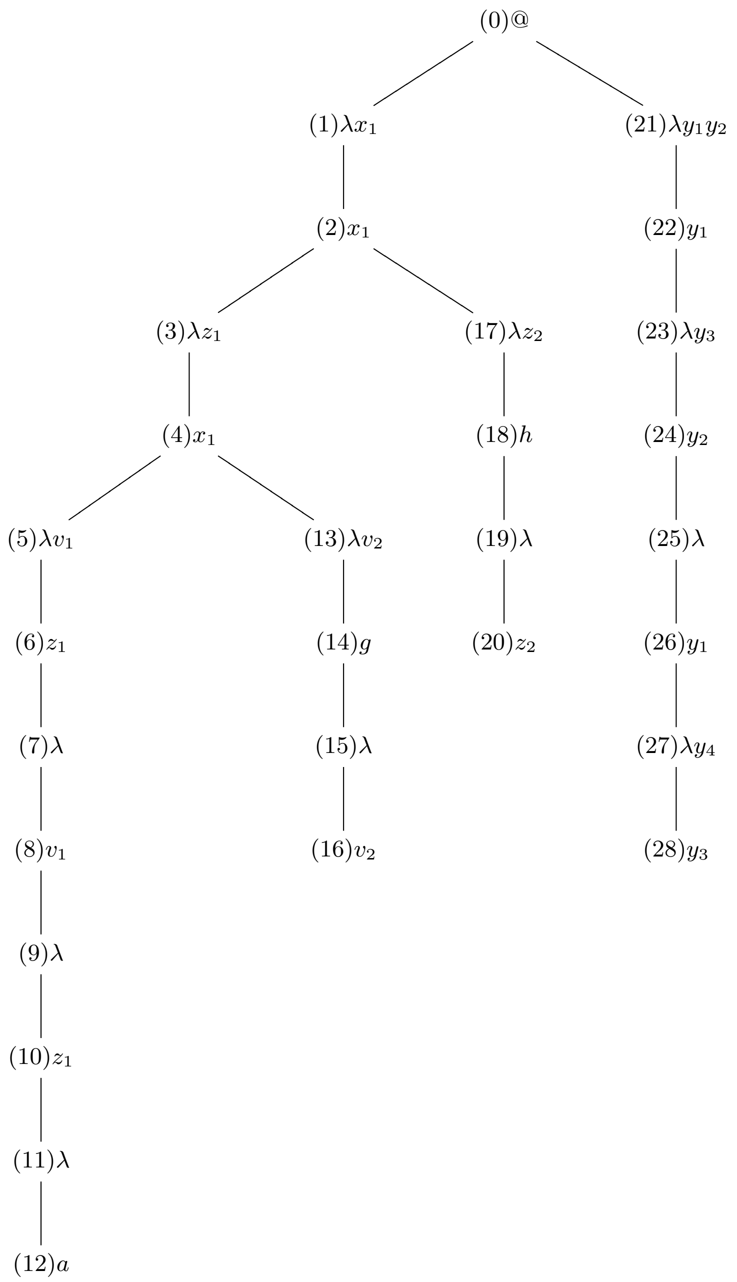


Fig. 8. The interpolation tree of Example 13

$(0)[u]\theta_1 = \emptyset$
$(1)[u]\theta_2 = \{((21), \theta_1)/1\}$
$(2)[u]\theta_2$
$(21)[u]\theta_3 = \{((3, 17), \theta_2)/21\}$
$(22)[u]\theta_3$
$(3)[u]\theta_4 = \theta_2\{((23), \theta_3)/3\}$
$(4)[u]\theta_4$
$(21)[u]\theta_5 = \{((5, 13), \theta_4)/21\}$
$(22)[u]\theta_5$
$(5)[u]\theta_6 = \theta_4\{((23), \theta_5)/5\}$
$(6)[u]\theta_6$
$(23)[u]\theta_7 = \theta_3\{((7), \theta_6)/23\}$
$(24)[u]\theta_7$
$(17)[u]\theta_8 = \theta_2\{((25), \theta_7)/17\}$
$(18)[u]\theta_8$
$(19)[u_1]\theta_8$
$(20)[u_1]\theta_8$
$(25)[u_1]\theta_7$
$(26)[u_1]\theta_7$
$(3)[u_1]\theta_8 = \theta_2\{((27), \theta_7)/3\}$
$(4)[u_1]\theta_8$
$(21)[u_1]\theta_9 = \{((5, 13), \theta_8)/21\}$
$(22)[u_1]\theta_9$
$(5)[u_1]\theta_{10} = \theta_8\{((23), \theta_9)/5\}$
$(6)[u_1]\theta_{10}$
$(27)[u_1]\theta_{11} = \theta_7\{((27), \theta_{10})/27\}$
$(28)[u_1]\theta_{11}$
$(7)[u_1]\theta_6$
$(8)[u_1]\theta_6$
$(23)[u_1]\theta_{12} = \theta_5\{((9), \theta_6)/23\}$
$(24)[u_1]\theta_{12}$
$(13)[u_1]\theta_{13} = \theta_4\{((25), \theta_{12})/13\}$
$(14)[u_1]\theta_{13}$
$(15)[u_2]\theta_{13}$
$(16)[u_2]\theta_{13}$
$(25)[u_2]\theta_{12}$
$(26)[u_2]\theta_{12}$
$(5)[u_2]\theta_{14} = \theta_4\{((27), \theta_{12})/5\}$
$(6)[u_2]\theta_{14}$
$(23)[u_2]\theta_{15} = \theta_3\{((7), \theta_{14})/23\}$
$(24)[u_2]\theta_{15}$
$(17)[u_2]\theta_{16} = \theta_2\{((25), \theta_{15})/17\}$
$(18)[u_2]\theta_{16}$
$(19)[ha]\theta_{16}$
$(20)[ha]\theta_{16}$
$(25)[ha]\theta_{15}$
$(26)[ha]\theta_{15}$
$(3)[ha]\theta_{17} = \theta_2\{((27), \theta_{15})/3\}$
$(4)[ha]\theta_{17}$
$(21)[ha]\theta_{18} = \{((5, 13), \theta_{17})/21\}$
$(22)[ha]\theta_{18}$
$(5)[ha]\theta_{19} = \theta_{17}\{((23), \theta_{18})/5\}$
$(6)[ha]\theta_{19}$
$(27)[ha]\theta_{20} = \theta_{15}\{((7), \theta_{19})/27\}$
$(28)[ha]\theta_{20}$
$(7)[ha]\theta_{14}$
$(8)[ha]\theta_{14}$
$(27)[ha]\theta_{21} = \theta_{12}\{((9), \theta_{14})/27\}$
$(28)[ha]\theta_{21}$
$(9)[ha]\theta_6$
$(10)[ha]\theta_6$
$(23)[ha]\theta_{22} = \theta_3\{((11), \theta_6)/23\}$
$(24)[ha]\theta_{22}$
$(17)[ha]\theta_{23} = \theta_2\{((25), \theta_{22})/17\}$
$(18)[ha]\theta_{23}$
$(19)[a]\theta_{23}$
$(20)[a]\theta_{23}$
$(25)[a]\theta_{22}$
$(26)[a]\theta_{22}$
$(3)[a]\theta_{24} = \theta_2\{((27), \theta_{22})/3\}$
$(4)[a]\theta_{24}$
$(21)[a]\theta_{25} = \{((5, 13), \theta_{24})/21\}$
$(22)[a]\theta_{25}$
$(5)[a]\theta_{26} = \theta_{24}\{((23), \theta_{25})/5\}$
$(6)[a]\theta_{26}$
$(27)[a]\theta_{27} = \theta_{22}\{((7), \theta_{26})/27\}$
$(28)[a]\theta_{27}$
$(11)[a]\theta_6$
$(12)[a]\theta_6$
$(12)[\exists]\theta_6$

Fig. 9. The single play of Example 13

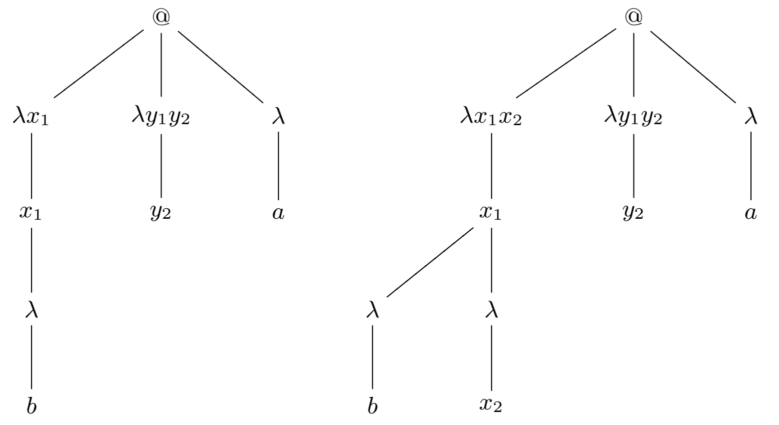


Fig. 10. Why long normal forms matter

happen after the jump from (the node labelled) x_1 to $\lambda y_1 y_2$ and the descent to y_2?; there is not a place to jump to in the first subtree.

We examine properties of game playing and prove that the game $\mathsf{G}(t, E)$ characterises when t solves E. Further properties are described which are useful for defining tree automata that recognise the set of solutions to interpolation equations. Also, these properties underpin uniformity features of game playing that account for the decidability of matching as described in Section 6.5.

We begin with some some simple observations.

Fact 10. Assume u is the goal term of E and t solves E then the number of plays in $\mathsf{G}(t, E)$ is branch(u).

If t doesnt solve E then the number of plays in $\mathsf{G}(t, E)$ can be less than branch(u).

Fact 11. Assume $n_1[r_1]\theta_1, \ldots, n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$.

1 If $i < l$ and $i = 2k$ for $k > 0$ then n_i is labelled with a $\lambda\overline{y}$ for some $\overline{y}$.
2 If $i = 2k + 1$ for $k > 0$ then n_i is labelled with a variable or a constant.
3 The node n_l is labelled with a constant and $r_l \in \{\forall, \exists\}$.
4 If $i < j < l$ then $r_j \in \mathrm{Sub}(r_i)$.

Figure 7 illustrates these features; because branch(ga) = 1 there is just a single play; each even position that is not final is at a node labelled with a lambda such as positions 6 at node (3) and 28 at node (17); odd positions are at nodes labelled with variables or constants with the initial position at the root labelled @. A term in a later state is a subterm of a term in an earlier state: after position 7 in Figure 7, the term is a and until then it is ga. Play must end at a node labelled with a constant.

The same look-up table may occur multiple times in a play. To be exact, two look-up tables θ, θ' are equal, $\theta = \theta'$, if they have the same entries. That is, they are defined for the same nodes and for each m, $\theta(m) = ((n_1, \ldots, n_p), \theta_i)$ iff $\theta'(m) = ((n_1, \ldots, n_p), \theta'_i)$ and $\theta_i = \theta'_i$: this is well-defined (and not "circular") because for each play position j, if

$\theta_j(m) = ((n_1, \ldots, n_p), \theta')$ then θ' is the look-up table at some earlier position $i < j$ (as we now show).

Proposition 12. Assume $n_1[r_1]\theta_1, \ldots, n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$.

1 $\theta_j(m)$ is defined iff node m is either above n_j or is n_j and m is labelled $\lambda y_1 \ldots y_p$ for some $y_1, \ldots, y_p$ with $p > 0$.

2 If $\theta_j(m) = ((m_1, \ldots, m_p), \theta')$ then m is labelled $\lambda y_1 \ldots y_p$ for some $y_1, \ldots, y_p$ and there is a $i < j$ such that $\theta' = \theta_i$ and either $i = 1$ and each $m_k = n_1(k + 1)$ or n_i is labelled with a variable and each $m_k = n_i k$.

Proof. For 1, the proof proceeds by simultaneous induction on where positions occur in the play with it and the additional property if $\theta(m) = ((m'_1, \ldots, m'_q), \theta')$ then $\theta'(m')$ is defined iff m' is above each m'_i, $1 \le i \le q$, and m' is labelled $\lambda y_1 \ldots y_{p'}$ for some $y_1, \ldots, y_{p'}$ with $p' > 0$. Clearly, both properties hold for θ_1 as it is the empty look-up table and n_1 is the root node of the interpolation tree. Assume the properties for all positions before $n_s[r_s]\theta_s$. We now show that they hold for $n_s[r_s]\theta_s$ by cases on the label at node $n = n_{s-1}$. If the label is @ then $n_s = n1$ and $\theta_s = \{((n2, \ldots, n(k+1)), \theta_1)/n1\}$; therefore, both properties hold. For the cases of when the label is $\lambda\overline{y}$, $a : \mathbf{0}$ and f the properties at $n_{s-1}[r_{s-1}]\theta_{s-1}$ are preserved at $n_s[r_s]\theta_s$. If n is labelled y_j then for some node m, $m \downarrow n$ and $\theta_{s-1}(m) = ((m_1, \ldots, m_{p'}), \theta')$ for some p', θ'; however, θ' is $\theta_{s'}$ for some $s' < s - 1$ and so obeys the second property. For the next position $n_s = m_j$ and either $\theta_s = \theta'$ and so obeys the first property or $\theta_s = \theta'\{((n1, \ldots, np''), \theta)/m_j\}$ for some p'' which obeys both properties. For 2, the only time that an entry for a node is added to a look-up table is when play is at that node and it is a binder $\lambda y_1 \ldots y_p$ for some $p > 0$: see the next positions in Figure 5 when the current node is labelled @ or $y : (B_1, \ldots, B_q, \mathbf{0})$. Moreover, the nodes in the entry are successor nodes of the root labelled @ or of the node labelled y and the associated look-up table in the entry belongs to the previous move. □

Node (8) in Figure 8 has the binders (5)λv_1, (3)λz_1 and (1)λx_1 above it. At position (8)$[u_1]\theta_6$ of the play in Figure 9 the look-up table θ_6 has precisely three entries $\{((21), \theta_1)/1, ((23), \theta_3)/3, ((23), \theta_5)/5\}$. Consider position 40, (23)$[u_2]\theta_{15}$: $\theta_{15}(21) = ((3, 17), \theta_2)$ and (21) is labelled $\lambda y_1 y_2$, nodes (3) and (17) are successors of (2) labelled x_1 and θ_2 is the look-up table at position 2.

Although the game abstracts from β-reduction there is an intimate relationship between positions and stages of β-reduction when look-up tables are interpreted as substitutions. Given a non-final position $n[r]\theta$ we can define its evaluation as the normal form of $n\theta$ which is to be understood as $t'\theta$ where t' is the subterm rooted at node n and θ is a substitution: player $\forall$ loses the game iff $n\theta \to^*_\beta r$. As an illustration, consider the evaluation of the position (6)$[a]\theta_4$ of Figure 7. The subterm at (6) in Figure 6 is $x_1(\lambda y.z)b$ and $\theta_4(x_1)$ is $w_1\theta_1 = w_1$ where $w_1 = \lambda y_1 y_2.y_1 y_2$ because θ_1 is empty, $\theta_4(z)$ is $y_2\theta_3$, $\theta_3(y_2)$ is $(x_1(\lambda z.x_1(\lambda z.z)z)a)\theta_2$ and $\theta_2(x_1)$ is w_1. Therefore the evaluation of (6)$[a]\theta_4$ is: $w_1(\lambda y.w_1(\lambda z.w_1(\lambda z.z)z)a)b \to^*_\beta (\lambda y.w_1(\lambda z.w_1(\lambda z.z)z)a)b \to_\beta w_1(\lambda z.w_1(\lambda z.z)z)a \to^*_\beta w_1(\lambda z.z)a \to^*_\beta (\lambda z.z)a \to_\beta a$.

This relationship between positions and reduction underlies the proof that the game *characterises* interpolation.

Theorem 13. Player $\forall$ loses $\mathsf{G}(t, E)$ iff t solves E.

Proof. Assume $x : A$, the equation E is $xw_1 \ldots w_k = u$ and $t : A$ is in lnf. Either t solves E or t fails to solve E and because the simply typed λ-calculus is strongly normalising and confluent, see Fact 1, it follows that there is an $m \geq 0$ such that $tw_1 \ldots w_k$ reduces to normal form using at most m β-reductions (whatever the reduction strategy). For any position $n[r]\theta$ of a play of $\mathsf{G}(t, E)$ we say that it m-holds (m-fails) if $r = \exists$ ($r = \forall$) or if $[r]$ is not final then as follows by cases on the label at n (where look-up tables become delayed substitutions and we elide between a node of an interpolation tree, the subtree rooted at that node and the subterm at that node):

— @ then $n1n2 \ldots n(k+1) =_\beta r$ ($n1n2 \ldots n(k+1) \neq_\beta r$) and $n1n2 \ldots n(k+1)$ reduces to normal form with at most m β-reductions;
— λ then $n1\theta =_\beta r$ ($n1\theta \neq_\beta r$) and $n1\theta$ reduces to normal form with at most m β-reductions;
— $\lambda y_1 \ldots y_p$ then $n1\theta =_\beta r$ ($n1\theta \neq_\beta r$) and $n1\theta$ reduces to normal form with at most $(m - p)$ β-reductions;
— f then $n\theta =_\beta r$ ($n\theta \neq_\beta r$) and $n\theta$ reduces to normal form with at most m β-reductions;
— $y_j : \mathbf{0}$ if $n' \downarrow n$ and $\theta(n') = ((n_1, \ldots, n_l), \theta')$ then $n_j\theta' =_\beta r$ ($n_j\theta' \neq_\beta r$) and $n_j\theta'$ reduces to normal form with at most m β-reductions;
— $y_j : (B_1, \ldots, B_p, \mathbf{0})$ if $n' \downarrow n$ and $\theta(n') = ((n_1, \ldots, n_l), \theta')$ then $t' =_\beta r$ ($t' \neq_\beta r$) where $t' = (n_j\theta')n1\theta \ldots np\theta$ and t' reduces to normal form with at most m β-reductions.

Initially, play is at n labelled @ with state [u] and the empty look-up table: therefore, as either $tw_1 \ldots w_k =_\beta u$ or $tw_1 \ldots w_k \neq_\beta u$ it follows that for some m, either $n[u]\theta_1$ m-holds or m-fails. The following invariants are easy to show by case analysis.

1If $n[r]\theta$ m-holds (m-fails), n labels $\lambda y_1 \ldots y_p$ and $n'[r']\theta'$ is the next position then it $(m - p)$-holds ($(m - p)$-fails).
2If $n[r]\theta$ m-holds (m-fails), n labels λ and $n'[r']\theta'$ is the next position then it m-holds (m-fails).
3If $n[r]\theta$ m-holds and n labels f and $n'[r']\theta'$ is any next position then it m'-holds for $m' \leq m$.
4If $n[r]\theta$ m-fails and n labels f then some next position $n'[r']\theta'$ m'-fails for some $m' \leq m$.
5If $n[r]\theta$ m-holds (m-fails) and n labels y_j and $n'[r']\theta'$ is the next position then it m-holds (m-fails).

From these invariants it follows first that if a non-final position m-holds then any next position m'-holds for some $m' \leq m$ and second if a non-final position $n[r]\theta$ m-fails then there is a next position that m'-fails for some $m' \leq m$. There cannot be an infinite sequence of positions and play cannot get stuck as we now show. The index m strictly decreases with a move at a node labelled $\lambda y_1 \ldots y_p$, $p > 0$, and must be 0 at a node labelled with a constant $a : \mathbf{0}$. The index m at a node labelled $y_j : (B_1, \ldots, B_p, \mathbf{0})$, $p > 0$, reduces within two positions as the next position is at a node labelled $\lambda z_1 \ldots z_p$ for some $z_1, \ldots, z_p$: via Proposition 12 there has to be a next position. Consequently, the index remains the same for subsequent positions at a dummy lambda and at a variable $y_j : \mathbf{0}$.

$$\begin{tikzcd}
\lambda y_1 \dots y_p & m[r]\theta \arrow[d, dotted, dash] & \theta(m) = ((n_1, \dots, n_p), \theta'') \\
y_j & n[r']\theta' & \theta'(m) = ((n_1, \dots, n_p), \theta'')
\end{tikzcd}$$

Fig. 11. Position at n is a child of the position at m

Also it may remain the same at a next position, if there is one, when at a node labelled with a constant f. To complete the argument we use Fact 11 and Proposition 12. The starting term u in the initial state has a fixed size. Therefore, there are boundedly many positions at nodes labelled with a constant because terms in the subsequent positions are always proper subterms. Furthermore, the look-up table at the subsequent position is unchanged when at a node labelled with a constant. Finally, there can only be a bounded sequence of positions that are at nodes labelled with variables $y : \mathbf{0}$, dummy lambdas and constants: if the kth position in a play is at a node labelled $y : \mathbf{0}$ then the look-up table at the next position at a dummy lambda is that of an earlier position and one takes this earlier position to determine that the next time play is at a variable $z : \mathbf{0}$ the look-up table at the subsequent position is that of an even earlier position, and so on. $\square$

We can use this characterisation to show further properties of look-up tables such as the same look-up table cannot occur in different positions at the same node.

Proposition 14. If $n[r]\theta$ and $n[r']\theta'$ are two different positions in a play of $\mathsf{G}(t, E)$ then $\theta \neq \theta'$.

Proof. Let E be the equation $xw_1 \dots w_k = u$ and assume that in $\mathsf{G}(t, E)$ there is a play $n_1[r_1]\theta_1, \dots, n_l[r_l]\theta_l$ such that $n_i = n_j$ and $\theta_i = \theta_j$ for $i \neq j$. We now show that this contradicts Theorem 13 and, in particular, the association of length of a play and the evaluation of a position in terms of a bounded m such that $tw_1 \dots w_k$ reduces to normal form in at most m β-reductions (whatever the strategy). First, if $r_i = r_j$ then the association is immediately broken because of the infinite play $n_1[r_1]\theta_1, \dots, n_i[r_i]\theta_i, \dots, n_i[r_i]\theta_i, \dots$. If $r_i \neq r_j$ then we can choose larger and larger starting terms r_1 (different from u) which allow particular plays to be arbitrarily long with prefixes of the form $n_1[r_1]\theta_1$, …, $n_i[r_i]\theta_i$, …, $n_i[r_{i_1}]\theta_i$, …, $n_i[r_{i_k}]\theta_i$: the number of β-reductions for the evaluation of a position to be in normal form does not depend on the state at that position. $\square$

4.4. *Parent and child positions*

When a position is at a node n labelled with a variable, we identify the earlier position at the node m that binds n and which defines where play will jump to next (from n).

Definition 20. The position $n[r']\theta'$ is a child of the earlier position $m[r]\theta$ in a play of $\mathsf{G}(t, E)$ if $m \downarrow n$ and $\theta(m) = \theta'(m)$.

Besides the condition that m binds n, there is also the requirement that the look-up tables agree on their entry for the binder m. Node m is labelled with $\lambda y_1 \dots y_p$ for some

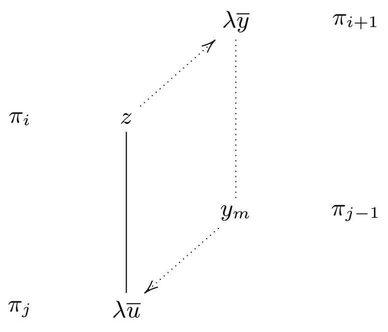


Fig. 12. Position π_j is a child of the position π_i

$p > 0$ and n is labelled with y_j for some j. We also say that position $m[r]\theta$ is the *parent* of $n[r']\theta'$ when the latter is a child of the former; a situation depicted in Figure 11.

In Figure 8 node (5) labelled λv_1 binds (8) labelled v_1. In the play in Figure 9 position $(8)[u_1]\theta_6$ is a child of $(5)[u]\theta_6$ but not of $(5)[u_1]\theta_{10}$ because $\theta_{10}(5) \neq \theta_6(5)$: $\theta_{10}(5) = ((23), \theta_9)$ and $\theta_6(5) = ((23), \theta_5)$ and $\theta_9 \neq \theta_5$ because $\theta_9(21) = ((5, 13), \theta_8)$ and $\theta_5(21) = ((5, 13), \theta_4)$ and $\theta_4(3) = ((23), \theta_3)$ whereas $\theta_8(3) = ((27), \theta_7)$.

Proposition 15. If a position in a play of $\mathrm{G}(t, E)$ is at a node labelled with a variable then there is a unique earlier position in the play that is its parent.

Proof. Assume $m \downarrow n$ and $n[r']\theta'$ is a position in a play of $\mathsf{G}(t, E)$ and $\theta'(m) = ((n_1, \ldots, n_l), \theta'')$. The first possibility is that there is not an earlier position $m[r]\theta$ such that $\theta(m) = \theta'(m)$. However, the only time when an entry for a binding node m enters a look-up table is when play is at m; see the two cases @ and $y_j : (B_1, \ldots, B_p, \mathbf{0})$ of Figure 5. Therefore there must be at least one position $m[r]\theta$ with $\theta(m) = \theta'(m)$. Next assume that there is more than one, $m[r_1]\theta_i$ and $m[r_2]\theta_j$ where $\theta_i(m) = \theta'(m) = \theta_j(m)$. However, both their previous positions must be at the same node (labelled with a variable or @) as $n_1, \ldots, n_l$ are its successors with the same look-up table θ''; this contradicts Proposition 14. □

We now extend the definition of child/parent to all positions except the initial and final positions; the following covers the cases other than that of Definition 20.

Definition 21. Assume $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$. For $1 < j < l$ we define when π_i is the parent of π_j by cases on the label at the node n' directly above n_j or the label at n_j;

— n' is labelled @: $i = 1$;
— n' or n_j is labelled with a constant: $i = j - 1$;
— n' is labelled with a variable: i is such that π_{i+1} is the parent of π_{j-1}.

The third case of Definition 21 is illustrated in Figure 12. Position π_i is at a node labelled with a variable z and π_j is at its mth successor; position π_{j-1} is at a variable y_m and it is the child of position π_{i+1} which is at a node labelled $\lambda\overline{y}$.

Proposition 16. Assume $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$. If $1 < j < l$ then there is a unique $i < j$ such that π_i is the parent of π_j.

Proof. Let $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ be a play of $\mathsf{G}(t, E)$. Consider any position π_j, $1 < j < l$. We show that there is a unique parent by case analysis on the label at n_j. Clearly, it cannot be @. If it is a constant then π_{j-1} is the parent of π_j and n_j is a successor node of n_{j-1}. If it is a variable then π_j has a unique parent by Proposition 15. Otherwise the label is $\lambda\overline{u}$. Let n' be the node directly above n_j in the tree; we now examine its label. If it is @ then π_1 is the parent of π_j; if it is a constant f then π_{j-1} is the parent of π_j. Finally, if it is a variable z then $z : (B_1, \ldots, B_p, \mathbf{0})$ for some p because n' has successors. Consider the moves in Figure 5; the only time that n_j occurs within an entry in a look-up table is at a subsequent position after a position at n'; this is at a node labelled $\lambda\overline{y}$ for some $\overline{y}$. Moreover, the look-up table θ_j is determined by the previous move of the parent of π_{j-1} which is unique by Proposition 15. □

Example 14. Consider the play in Figure 7 on the tree of Figure 6. Positions $(1)[ga]\theta_2$, $(21)[ga]\theta_3$ and $(21)[a]\theta_5$ are some of the children of the initial position. The positions $(3)[ga]\theta_4$ and $(11)[a]\theta_2$ are children of $(2)[ga]\theta_2$; for the second of these, this is because position $(24)[a]\theta_3$ is a child of $(21)[ga]\theta_3$. □

The following is a consequence of uniqueness of a parent position.

Fact 17. If $\theta(m), \theta'(m)$ are defined, $\theta(m) = \theta'(m)$ and m' is a binder above m then $\theta(m') = \theta'(m')$.

Definition 22. The look-up table θ' *extends* θ, written $\theta \leq \theta'$, if for all nodes m if $\theta(m)$ is defined then $\theta'(m) = \theta(m)$.

Proposition 18. If $m[r]\theta$ is the parent of $n[r']\theta'$ in a play then $\theta \leq \theta'$.

Proof. Assume $m[r]\theta$ is the parent of $n[r']\theta'$ and node n' is directly above n. We proceed by case analysis on the labels at n' and n. If n' is labelled @ then $\theta = \theta_1$ which is the initial empty look-up table; so $\theta \leq \theta'$. If n' or n is labelled with a constant then $\theta = \theta'$. If n' is labelled with a variable then consider the label of n; if it is a dummy lambda then $\theta = \theta'$; otherwise the label is $\lambda z_1 \ldots z_p$ for $p > 0$ and $\theta' = \theta\{((m_1, \ldots, m_p), \theta'')/n\}$; by Proposition 12, θ cannot have an entry for node n and so $\theta < \theta'$. The remaining case is when $m \downarrow n$ and $\theta(m) = \theta'(m)$. Consider the sequence of nodes $mm_1m_2 \ldots m_{2k-1}m_{2k}n$ which is the branch between m and n for some $k \geq 0$ and the positions $m_1[r'_1]\theta'_1, \ldots, m_{2k}[r'_{2k}]\theta'_{2k}$ such that $m_1[r'_1]\theta'_1$ is the next position after $m[r]\theta$; $m_{2j-1}[r'_{2j-1}]\theta'_{2j-1}$ is the parent of $m_{2j}[r'_{2j}]\theta'_{2j}$ for $1 < j \leq k$; $m_{2j+1}[r'_{2j+1}]\theta'_{2j+1}$ is the next position after $m_{2j}[r'_{2j}]\theta'_{2j}$ for $1 \leq j < k$; and $n[r']\theta'$ is the position after and $m_{2k}[r'_{2k}]\theta'_{2k}$. Clearly, $\theta \leq \theta'_1 \leq \ldots \leq \theta'_{2k} \leq \theta'$; so, θ' extends θ because $\leq$ is transitive. □

A key consequence of Proposition 18 is that the meaning of a binder at a parent position is preserved at its children. We shall exploit this in Section 6.1 when exhibiting uniformity features of game playing.

$(0)@$	$[u]\theta_1$	$\theta_1 = \emptyset$
$(1)\lambda x_1$	$[u]\theta_2$	$\theta_2 = \{((21), \theta_1)/1\}$
$(2)x_1$	$[u]\theta_2$	
$(3)\lambda z_1$	$[u]\theta_4$	$\theta_4 = \theta_2\{((23), \theta_3)/3\}$
$(4)x_1$	$[u]\theta_4$	
$(5)\lambda v_1$	$[u]\theta_6$	$\theta_6 = \theta_4\{((23), \theta_5)/5\}$
$(6)z_1$	$[u]\theta_6$	
$(7)\lambda$	$[u_1]\theta_6$	
$(8)v_1$	$[u_1]\theta_6$	
$(9)\lambda$	$[ha]\theta_6$	
$(10)z_1$	$[ha]\theta_6$	
$(11)\lambda$	$[a]\theta_6$	

Fig. 13. The sequence of branch positions for $(11)[a]\theta_6$

Proposition 19. Assume $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$ and $m_1, \ldots, m_p$ where $m_1 = n_1$ and $m_p = n_j$ is the branch from the root of the interpolation tree to node n_j where $j < l$. Then for $1 = j_1 < \ldots < j_p = j$ there is a unique sequence of positions $\pi_{j_1}, \ldots, \pi_{j_p}$, such that for each $i : 1 \leq i \leq p$

1 $n_{j_i} = m_i$,
2 if i is even then π_{j_i} is the child of $\pi_{j_{i-1}}$,
3 if $i > 1$ is odd then $j_i = j_{i-1} + 1$,
4 if $m_i \downarrow m_k$ then π_{j_k} is the child of π_{j_i},
5 $\theta_{j_1} \leq \ldots \leq \theta_{j_p}$.

Proof. Let $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ be a play of $\mathsf{G}(t, E)$ and $j < l$. Let $m_1, \ldots, m_p$ be the branch from the root of the interpolation tree to node n_j where $m_1 = n_1$ and $m_p = n_j$. We define the positions $\pi_{j_p}, \ldots, \pi_{j_1}$ iteratively such that $n_{j_i} = m_i$ for all $i : 1 \leq i \leq p$ as follows. Initially $\pi_{j_p} = \pi_j$. Assume for $k > 1$, $\pi_{j_{k+1}}$ at m_{k+1} is defined. We now proceed by cases on the label at m_{k+1}; if it is a variable or a constant then $\pi_{j_k} = \pi_{(j_{k+1})-1}$; if it is a $\lambda\overline{y}$ then π_{j_k} is the parent of $\pi_{j_{k+1}}$. Next we show that properties $2 - 5$ all hold; by definition 2 and 3 hold. Clearly, 5 also holds using Proposition 18. 4 follows from 5 and Propositions 12 and 15; if $m_i \downarrow m_k$ then $\theta_{j_i} \leq \theta_{j_k}$ and $\theta_{j_i}(m_i)$ is defined and, therefore, $\theta_{j_i}(m_i) = \theta_{j_k}(m_i)$ which guarantees that π_{j_k} is the child of π_{j_i}. □

Definition 23. Assume $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$.

1 The sequence of branch positions for π_j, $j < l$, is $\pi_{j_1}, \ldots, \pi_{j_p} = \pi_j$ as determined in Proposition 19.
2 The sequence of branch positions for π_j to π_i is the suffix of the sequence for π_j, $\pi_{j_m}, \ldots, \pi_{j_p}$ when $\pi_i = \pi_{j_m}$.
3 Assume $j < k$, $\pi_{j_1}, \ldots, \pi_{j_p}$ is the sequence of branch positions for π_j and $\pi_{k_1}, \ldots, \pi_{k_q}$ is the sequence for π_k. Position π_j is compatible with π_k if when i is the largest index such that $n_{j_i} = n_{k_i}$ this implies $\pi_{j_i} = \pi_{k_i}$.

The sequence of branch positions for $(11)[a]\theta_6$ of the play of Figure 9 on the tree in Figure 8 is given in Figure 13 as a linear sequence. To aid the reader, the entries in the look-up tables are given. The positions, for instance, at (6) and (10) are children of the position at (3). In this example, position $(13)[u_1]\theta_{13}$ is compatible with $(11)[a]\theta_6$.

Fact 20. Assume $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$. If n_j is above n_k and π_j is compatible with π_k then π_j occurs in the sequence of branch positions for π_k.

5. Tree automata revisited

In Section 3.2 we defined tree automata following (Comon and Jurski 1997) for recognising solutions of 4th-order interpolation equations. The alphabet of the automaton ranges over the variables, constants and binders of potential solution terms and the states of the automaton uses evaluation to normal form of a node of the tree of such a term. There are two problems with extending the automata beyond 4th-order; first, a set of solutions may require an infinite alphabet; second, the set of states of the automaton may also be infinite even when the alphabet is restricted to be finite. As mentioned in Section 3.2 we can overcome the first problem by representing terms as binding trees.

In this section, using the game theoretic characterisation of interpolation, we define tree automata that work at all orders. There are two stages. First is a restricted result: for any interpolation equation at any order and for a fixed finite alphabet, there is a classical tree automaton that recognises the solutions to the equation that are built from that alphabet. As with the game, we define the tree automata to recognise interpolation trees of Figure 1. The second stage involves new kinds of tree automata that operate on binding trees called *dependency* tree automata. The set of solutions of an interpolation equation is characterised precisely using *alternating* dependency tree automata. However, the non-emptiness problem for such automata is in general undecidable as shown in (Ong and Tzevelekos 2009); so, this characterisation does not lead to decidability of matching.

5.1. *Variable profiles*

In the game, play jumps around the interpolation tree as illustrated in Examples 12 and 13. The question is how to abstract from this motion statically using states of a tree automaton. Our solution is based on Ong (Ong 2006) (which is a different setting, with a fixed infinite lambda term and an alternating parity tree automaton). We break apart the *motion of jumping into and out of components* into constituents using variable profiles.

Definition 24. Assume V is a finite set of variables and $u : \mathbf{0}$ is a goal term. For each $y \in V$, $\Gamma(y)$ is the set of y profiles defined inductively: if $y : \mathbf{0}$ then $\Gamma(y) = \{(y, r, \emptyset) \,|\, r \in \mathrm{Sub}(u)\}$; if $y : (B_1, \ldots, B_p, \mathbf{0})$ then $\Gamma(y) = \{(y, r, \Gamma) \,|\, r \in \mathrm{Sub}(u), \Gamma \subseteq \bigcup_{1 \leq i \leq p} \bigcup_{z : B_i \in V} \Gamma(z)\}$. A mode is a pair (r, Γ) where $r \in \mathrm{Sub}(u)$ and $\Gamma \subseteq \bigcup_{y \in V} \Gamma(y)$.

Although a variable profile is defined independently of the interpolation game, it is intended to be an abstraction from subsequences of positions. Consider the play in Figure 9 of Example 13 and its interpolation tree in Figure 8. A $y_3 : \mathbf{0}$ profile has the form

$$\begin{array}{llll}
(x_1, u, \{ & (y_1, u, \{ & (z_1, u, \{ & (y_3, u_1, \emptyset)\}) \\
 & & (z_1, u_2, \{ & (y_3, ha, \emptyset)\}) \\
 & & (z_1, ha, \{ & (y_3, a, \emptyset)\})\}) \\
 & (y_1, u_1, \{ & (z_1, u_1, & \emptyset)\}) \\
 & (y_1, ha, \{ & (z_1, ha, & \emptyset)\}) \\
 & (y_1, a, \{ & (z_1, a & \emptyset)\}) \\
 & (y_2, u, \{ & (z_2, u_1, & \emptyset)\}) \\
 & (y_2, u_2, \{ & (z_2, ha, & \emptyset)\}) \\
 & (y_2, ha, \{ & (z_2, a, & \emptyset)\})\})
\end{array}$$

Fig. 14. A z profile for Example 13

$(y_3, r, \emptyset)$ where $r \in \text{Sub}(u)$, which captures a game position at node (28) labelled y_3; there are three such positions $(28)[u_1]\theta_{11}$, $(28)[ha]\theta_{20}$ and $(28)[a]\theta_{27}$; the associated profiles are $(y_3, u_1, \emptyset)$, $(y_3, ha, \emptyset)$ and $(y_3, a, \emptyset)$. The key point is that there is an upper bound on the number of such profiles, the size of $\text{Sub}(u)$; however, there is no upper bound on the number of times a play may be at a node labelled y_3. In this strong sense a variable profile is an *abstraction.* A $z_1 : (\mathbf{0}, \mathbf{0})$ profile captures those positions at nodes (6) and (10) and has the form (z_1, r, Γ') where Γ' is a set of profiles for variables $y : \mathbf{0}$. Assume a position $n[r]\theta$ at a node labelled z_1 and a next position $n'[r]\theta'$. The associated profile is (z_1, r, Γ') where Γ' is (the possibly empty) set of associated profiles of any position that is a child of $n'[r]\theta'$; for instance, in the play of Figure 9 there is the position $(6)[u]\theta_6$ and its corresponding profile is $(z_1, u, \{(y_3, u_1, \emptyset)\})$ because the next position is $(23)[u]\theta_7$ whose only child is $(28)[u_1]\theta_{11}$. Similarly, $(z_1, u_1, \emptyset)$ is associated with $(6)[u_1]\theta_{10}$ because the next position is $(27)[u_1]\theta_{11}$ which has no children.

For higher-order variables such as $x_1 : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), (\mathbf{0}, \mathbf{0}), \mathbf{0})$ of Figure 8 the pattern of a profile is the same. For instance, Figure 14 is the profile associated with the position $(2)[u]\theta_2$ of the play in Figure 9; the y_1 and y_2 profiles are those associated with the children of the position $(21)[u]\theta_3$ at nodes (22), (24) and (26); in turn, for example, for the first of the y_1 profiles the z_1 profiles are those associated with the children of the position $(3)[u]\theta_4$ and so on.

A mode is a pair (r, Γ) where $r \in \text{Sub}(u)$ and Γ is a set of variable profiles. Because $\text{Sub}(u)$ is finite and Σ is fixed, there can only be boundedly many different modes (r, Γ).

5.2. *Tree automata for interpolation at all orders*

We now come to the formal definition of the tree automaton whose states are sets of modes. For ease of exposition, we assume *well-named* interpolation trees where each occurrence of a variable in a binder is distinct. This means that binders can not share variables; for instance, the tree in Figure 6 is not well-named because of the multiple occurrences of λz; there is a well-named tree that is α-equivalent to it (with a larger alphabet). It is possible to avoid this assumption but at the expense of a more complicated definition of the tree automaton.

Definition 25. Assume an interpolation equation E, $xw_1 \dots w_k = u$ and a fixed finite alphabet Σ, the labels for interpolation trees $@tw_1 \dots w_k$ where t has the form $\lambda x_1 \dots x_k.t'$. Let $V \subset \Sigma$ be the variables in Σ. The Σ-tree automaton for E is $\mathsf{A}_E = (Q, \Sigma, F, \Delta)$ where each $q \in Q$ is a set of modes $\{(r_1, \Gamma_1), \dots, (r_m, \Gamma_m)\}$, $m \geq 0$, $r_i \in \text{Sub}(u)$ and $\Gamma_i \subseteq \bigcup_{y \in V} \Gamma(y)$, $F = \{\{(u, \emptyset)\}\}$ and the transition relation Δ is defined on nodes of the interpolation tree by cases on its label in Σ.

— $a : \mathbf{0}$; then $a \Rightarrow \Lambda$ for each $\Lambda \subseteq \{(a, \emptyset)\}$

— $y : \mathbf{0}$; then $y \Rightarrow \Lambda$ for each $\Lambda \subseteq \{(r, \{(y, r, \emptyset)\}) \mid r \in \text{Sub}(u)\}$

— $f : (A_1, \dots, A_p, \mathbf{0})$; then $f\Lambda_1 \dots \Lambda_p \Rightarrow \Lambda$ for each $\Lambda_1, \dots, \Lambda_p, \Lambda$ that obeys

1 if $(r, \Gamma) \in \Lambda$ then $r = fr_1 \dots r_p$ and for each $i : 1 \leq i \leq p$ there is $(r_i, \Gamma_i) \in \Lambda_i$ and $\Gamma = \bigcup \Gamma_i$; and

2 if $(r_i, \Gamma_i) \in \Lambda_i$ then there is $(fr_1 \dots r_p, \Gamma) \in \Lambda$ and $\Gamma_i \subseteq \Gamma$.

— $y : (A_1, \dots, A_p, \mathbf{0})$; then $y\Lambda_1 \dots \Lambda_p \Rightarrow \Lambda$ for each $\Lambda_1, \dots, \Lambda_p, \Lambda$ that obeys

1 if $(r, \Gamma) \in \Lambda$ then there is a $(y, r, \Gamma') \in \Gamma$ and

$$\begin{array}{rcll} \Gamma' & = & \{ & (z_1, r_{11}, \Gamma'_{11}), \dots, (z_1, r_{1m_1}, \Gamma'_{1m_1}) \\ & & & \vdots \\ & & & (z_p, r_{p1}, \Gamma'_{p1}), \dots, (z_p, r_{pm_p}, \Gamma'_{pm_p})\} \end{array}$$

for some $z_1, \dots, z_p$ and $m_i \geq 0$ and for each $i : 1 \leq i \leq p$ and $j : 1 \leq j \leq m_j$, $(r_{ij}, \Gamma'_{ij} \cup \Sigma_{ij}) \in \Lambda_i$ when $\Gamma = \bigcup \bigcup \Sigma_{ij} \cup \{(y, r, \Gamma')\}$; and

2 if $(r_i, \Gamma_i) \in \Lambda_i$ then there is a $(r, \Gamma) \in \Lambda$ and $(y, r, \Gamma') \in \Gamma$ such that $(z_i, r_i, \Gamma'') \in \Gamma'$ for some z_i and $\Gamma'' \subseteq \Gamma_i$ and $\Gamma_i - \Gamma'' \subseteq \Gamma$.

— $\lambda\overline{y}$; then $\lambda\overline{y}\,\Lambda \Rightarrow \Lambda$ for each Λ.

— $@ : ((A_1, \dots, A_k, \mathbf{0}), A_1, \dots, A_k, \mathbf{0})$; then $@\Lambda_1 \dots \Lambda_{k+1} \Rightarrow \{(u, \emptyset)\}$ for each $\Lambda_1, \dots, \Lambda_{k+1}$ that obeys

1 $\Lambda_1 = \{(u, \Gamma)\}$ for some Γ; and

2 if $(z, r', \Gamma') \in \Gamma$ then $z = x_i$ for some i and $(r', \Gamma') \in \Lambda_{i+1}$; and

3 if $(r_i, \Gamma_i) \in \Lambda_{i+1}$ then $(x_i, r_i, \Gamma_i) \in \Gamma$.

A run of the automaton on an interpolation tree accumulates modes: the idea is that positions in a game at a node are associated with modes at that node; and vice versa. For example, if (r, Γ) is at node n labelled f then r is $fr_1 \dots r_p$, for some $r_1, \dots, r_p$ and for each i, there is a mode (r_i, Γ_i) at ni so that $\Gamma = \bigcup \Gamma_i$. The only allowed variable profiles in Γ of a mode (r, Γ) at node n are free variable occurrences that occur in the subtree rooted at n or of $n1$ when n is labelled with a $\lambda\overline{z}$. When (r, Γ) is at n labelled with a variable $y : (A_1, \dots, A_p, \mathbf{0})$ and $n[r]\theta$ is its associated position, there is a variable profile $(y, r, \Gamma') \in \Gamma$ such that Γ' is the set of variable profiles associated with the children of the next position after $n[r]\theta$: for this next position is at a node labelled $\lambda z_1 \dots z_p$ for some $z_1, \dots, z_p$. For the case that $(u, \emptyset)$ is at the root n labelled $@$ then (u, Γ) is at the first successor $n1$ labelled $\lambda x_1 \dots x_k$ and each profile in Γ has the form (x_i, r_i, Γ_i) representing a later position $m[r_i]\theta'$ at a node labelled x_i and so (r_i, Γ_i) is a mode at $n(i+1)$ as play jumps from m to $n(i+1)$.

Example 15. We describe the automaton for Example 8 (which should be compared with the Comon and Jurski tree automaton of Example 9). A potential solution term such as the one depicted in Figure 2 has the form $\lambda x_1 x_2.w$. So the alphabet Σ is the set $\{a, b, f, x_1, x_2, y_1, y_3, \lambda, \lambda y_3, \lambda y_1 y_2, \lambda x_1 x_2\}$. The variable set V is $\{x_1, x_2, y_1, y_3\}$ and $\mathrm{Sub}(faa) = \{faa, a\}$. A mode is (r, Γ) where $r \in \mathrm{Sub}(u)$ and Γ is a set of variable profiles. Next, we consider the transitions, first for constants.

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

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

The last rule allows f to be labelled with the state $\{(faa, \Gamma_1 \cup \Gamma_2)\}$ when its successors are labelled with the states $\{(a, \Gamma_1)\}$ and $\{(a, \Gamma_2)\}$. Next we examine the transitions 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. Assume Γ' is $\{A(x_1), A'(x_1), A(x_2)\}$ where $A(x_1)$ is the profile $(x_1, faa, \{(y_1, faa, \emptyset)\})$, $A'(x_1)$ is $(x_1, a, \{(y_1, a, \emptyset)\})$ and $A(x_2)$ is $(x_2, faa, \{(y_3, a, \emptyset)\})$.

$$x_1\ \emptyset\ \emptyset \Rightarrow \emptyset \qquad x_2\ \emptyset \Rightarrow \emptyset$$

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

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

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

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 $\Gamma \cup \{A(x_1)\} = \Gamma''$. Finally, there are the rules for the λ's and @.

$$\lambda\ \Lambda \Rightarrow \Lambda \qquad \lambda x_1 x_2\ \Lambda \Rightarrow \Lambda$$

$$\lambda y_1 y_2\ \Lambda \Rightarrow \Lambda \qquad \lambda y_3\ \Lambda \Rightarrow \Lambda$$

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

where (1) is the condition: $A(x_1) \in \Gamma$ 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 (Dowek 1994): the automaton accepts the interpolation tree when t is any term of the form $\lambda x_1 x_2.x_1(x_1 \ldots x_1(x_2(x_1 \ldots x_1 ab)b \ldots) \ldots)b$. □

Theorem 21. Assume E is the interpolation equation $xw_1 \ldots w_k = u$ and Σ is a fixed finite alphabet. The automaton A_E accepts the well-named Σ-tree $@tw_1 \ldots w_k$ iff t solves E.

Proof. Let E be $xw_1 \dots w_k = u$ and assume the labels of the nodes in the well-named tree $@tw_1 \dots w_k$ belong to Σ. Let t be a solution to E. Therefore, player $\forall$ loses $\mathsf{G}(t, E)$. We show how to build an accepting run of A_E on $@tw_1 \dots w_k$. The main construction is to transform each position $n[r]\theta$ into a mode (r, Γ) at n in the successful run; consequently, the set of all positions at a node is transformed into a state of the automaton. The construction starts from leaf nodes of $@tw_1 \dots w_k$. If n is labelled $a : \mathbf{0}$ then the state of the successful run at this node is either $\{(a, \emptyset)\}$ if there is a position $n[a]\theta$ or $\emptyset$ if there is no such position. These both obey the conditions on a transition in Definition 25. If n is labelled $y : \mathbf{0}$ then the state of the successful run at this node is $\{(r, \{(y, r, \emptyset)\}) \,|\, \text{there is a position } n[r]\theta\}$. Again, this obeys the transition condition. Next we consider the general case of an arbitrary node n of $@tw_1 \dots w_k$. If n is labelled $\lambda\overline{y}$ and $n[r]\theta$ is a position then as the next position in the play is $n1[r]\theta$ the automaton state at $n1$ includes a mode (r, Γ); therefore, the state at n is that at $n1$ as required by the transition rule for $\lambda\overline{y}$ of Definition 25. If n is labelled $f : (A_1, \dots, A_p, \mathbf{0})$ then because t solves E every position at n has the form $n[fr_1 \dots r_p]\theta$ and every subsequent position has the form $ni[r_i]\theta$; therefore, for each such position at n there is a mode $(fr_1 \dots r_p, \Gamma)$ where $\Gamma = \bigcup \Gamma_i$ such that (r_i, Γ_i) is a mode at ni. So, the conditions for a f transition in Definition 25 are fulfilled. If n is labelled $y : (A_1, \dots, A_p, \mathbf{0})$ and $n[r]\theta$ is a position then any child of this position is at ni for some i; conversely, any position at ni has a parent at n. Assume position $n[r]\theta$. We associate the mode (r, Γ) with it as follows. Consider the next position $n'[r]\theta'$ at n' labelled with $\lambda z_1 \dots z_p$ for some $z_1, \dots, z_p$. Assume all the children of $n'[r]\theta'$ are as follows

$$\begin{array}{ll} \{ & m_{11}[r_{11}]\theta_{11}, \dots, m_{1l_1}[r_{1l_1}]\theta_{1l_1} \\ & \vdots \\ & m_{p1}[r_{p1}]\theta_{p1}, \dots, m_{pl_p}[r_{pl_p}]\theta_{pl_p} \ \} \end{array}$$

where each m_{ij} is labelled with z_i. The next position of each $m_{ij}[r_{ij}]\theta_{ij}$ is $ni[r_{ij}]\theta'_{ij}$: associated with it is the mode $(r_{ij}, \Gamma'_{ij} \cup \Sigma_{ij})$ where if ni is labelled $\lambda s_1 \dots s_q$ then Γ'_{ij} is the set of variable profiles for each $s_{i'}$ (and Σ_{ij} does not contain any such profiles). Therefore, (r, Γ) is the mode at n such that $(y, r, \Gamma') \in \Gamma$ and $\Gamma = \bigcup \Sigma_{ij} \cup \{(y, r, \Gamma')\}$ and Γ' is

$$\begin{array}{ll} \{ & (z_1, r_{11}, \Gamma'_{11}), \dots, (z_1, r_{1l_1}, \Gamma'_{1l_1}) \\ & \vdots \\ & (z_p, r_{p1}, \Gamma'_{p1}), \dots, (z_p, r_{pl_p}, \Gamma'_{pl_p}) \ \} \end{array}$$

Now this is repeated for every position at n; it follows that the conditions for a transition at n are thereby obeyed. If n is labelled $@$ then there is the single initial position $n[u]\theta_1$ at n; its associated mode is $(u, \emptyset)$. There is also just a single position at $n1$ with corresponding mode (u, Γ); for each child of this position, by construction, there is a corresponding variable profile (x_i, r_i, Γ_i) such that there is a child of $n[u]\theta_1$ at $n(i+1)$ whose corresponding mode is (r_i, Γ_i). We leave the reader to check the details. Conversely, assume A_E accepts $@tw_1 \dots w_k$. We derive a contradiction if we also assume that t does not solve E. So, there is a play $\pi = n_1[r_1]\theta_1, \dots, n_l[r_l]\theta_l$ of $\mathsf{G}(t, E)$ that $\forall$ wins. As there is a successful run of A_E on $@tw_1 \dots w_k$ we are interested in modes that are associated with positions in π at the same nodes; for the initial position there is the associated mode $(u, \emptyset)$; for the penultimate position $n_{l-1}[r_{l-1}]\theta_{l-1}$ either n_{l-1} is labelled with $a : \mathbf{0}$ and $r_{l-1} \neq a$ or it is labelled with f and r_{l-1} does not have the form $fr'_1 \ldots r'_p$; therefore, there can not be a mode in the successful run at n_{l-1} that corresponds to this position because it would violate the transition rules of Definition 25. Therefore, consider the latest position $n_j[r_j]\theta_j$ in π for which thare is a corresponding mode (r_j, Γ_j) at n_j in the accepting run. We now examine the label at n_j; it cannot be a constant a, f, @ or a $\lambda\overline{y}$. Therefore, it must be at a variable y. There is a $(y, r_j, \Gamma'') \in \Gamma_j$ as (r_j, Γ_j) is the corresponding mode at n_j. Consider the next position $n_{j+1}[r_{j+1}]\theta_{j+1}$ at a node labelled $\lambda\overline{z}$ for some $\overline{z}$. So, (r_j, Γ'') is not a mode at r_{j+1}. Next consider the parent π_i of π_j; at node n_i labelled $\lambda\overline{y}$; there is a corresponding mode (r_i, Γ_i) such that $(y, r_j, \Gamma'') \in \Gamma_i$. If y is level 1, x_m, and in t then n_j is node $(m + 1)$ and the result now follows by the tree automaton rule for @. Otherwise, π_{j+1} is a child of π_{i-1} where n_{i-1} is labelled with varaible s and the result follows by considering the corresponding mode for π_{i-1} whcih must contain (s, r_{i-1}, Γ''') with $(y, r_j, \Gamma'') \in \Gamma'''$ and therefore (r_j, Γ''') is a mode at node n_{j+1}. □

5.3. *Dependency tree automata*

The question is how to extend Thereom 21 and overcome the restriction to a finite alphabet. First we represent terms as *binding* trees as defined in Section 3.1. Next, we introduce a special kind of tree automaton that can recognise binding trees that we call *dependency* tree automata. These are a top-down tree automaton (unlike the previous tree automata) that start at the root of the tree and work down to the leaves; the main difference is that the transition rules are therefore reversed; of the form $sq \Rightarrow q_1 \ldots q_k$: if s has arity k and state q labels a node labelled s then $q_1, \ldots, q_k$ can label its successors; compare Definition 14 and see (Comon et als 2002).

Definition 26. A dependency Σ-tree automaton $\mathsf{A} = (Q, \Sigma, q_0, \Delta)$ where Q is a finite set of states, Σ is the finite alphabet ($\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ where Σ_1 are the binders, Σ_2 the bound variables and Σ_3 the remaining symbols), $q_0 \in Q$ is the initial state and Δ is a finite set of transition rules each of which has one of the following three forms.

1 $qs \Rightarrow (q_1, \ldots, q_k)$ where $s \in \Sigma_2 \cup \Sigma_3$, $\mathrm{ar}(s) = k$, $q, q_1, \ldots, q_k \in Q$
2 $qs \Rightarrow q's'$ where $s \in \Sigma_1$, $s' \in \Sigma_3$ and $q, q' \in Q$
3 $(q', q)s \Rightarrow q_1x$ where $s \in \Sigma_1$, $x \in \Sigma_2$ and $q', q, q_1 \in Q$

Definition 27. A run of $\mathsf{A} = (Q, \Sigma, q_0, \Delta)$ on $t \in \mathsf{T}_\Sigma$ is a $(\Sigma \times Q)$-tree whose nodes are pairs (n, q) where n is a node of t and $q \in Q$ labelled (s, q) if n is labelled s in t which is defined top-down with root (ϵ, q_0) where ϵ is the root of t. Consider a node (n, q) labelled (s, q) of a partial run tree which does not have successors. If $s \in \Sigma_2 \cup \Sigma_3$ and $qs \Rightarrow (q_1, \ldots, q_k) \in \Delta$ then the successors of (n, q) are the nodes (ni, q_i), $1 \leq i \leq k$. If $s \in \Sigma_1$, $n1$ is labelled $s' \in \Sigma_3$ and $qs \Rightarrow q's' \in \Delta$ then $(n1, q')$ is the successor of (n, q). If $s \in \Sigma_1$, $n1$ is labelled $x \in \Sigma_2$, $m \downarrow n1$ in t, (m, q') occurs above or at (n, q) and $(q', q)s \Rightarrow q_1x \in \Delta$ then $(n1, q_1)$ is the successor of (n, q). A accepts the Σ-tree t iff there is a run of A on t such that if (n, q) is a leaf then n is a leaf of t. Let $\mathsf{T}_\Sigma(\mathsf{A})$ be the set of Σ-trees accepted by A.

A dependency tree automaton A has a finite set of states Q and transitions Δ (which can be nondeterministic). A run of A on a Σ-tree t adds an additional Q labelling to (a subtree of) t as we described in Section 3.2:so it is a $(\Sigma \times Q)$-tree. It starts with (ϵ, q_0) where ϵ is the root of t and q_0 is the initial state of A. Subsequent nodes are derived by percolating states down t. The state at a node that is labelled with a variable not only depends on the state of its immediate predecessor but also on the state of the node that labels its binder. This introduces non-local dependence in the automaton (hence the name). A run on t is accepting if it is complete in the sense that each node of t is labelled with an element of Q: if (n, q) is a leaf of the run tree then n is a leaf of t.

Dependency tree automata were partly inspired by nested word and tree automata (Alur and Madhusudan 2006; Alur, Chaudhuri and Madhusudan 2006) which are also an amalgam of a traditional automaton and a binary relation $\downarrow$ on nodes of the (possibly infinite) word or tree. However, in that setting $\downarrow$ represents *nesting* such as provided by bracketing and useful for modelling procedure calls and returns. Nesting involves natural restrictions on the relation $\downarrow$ such as "no-crossings": if $m_1 \downarrow m_2$ and $n_1 \downarrow n_2$ and m_1 is above n_1 then either m_2 is above n_1 or n_2 is above m_2. Such restrictions are not appropriate when modelling binding, for instance as with a formula $\forall f.\exists x.\phi(f(x))$.

Theorem 22. Assume A, A_1 and A_2 are dependency Σ-tree automata.

1 The non-emptiness problem, given A is $\mathsf{T}_\Sigma(\mathsf{A}) \neq \emptyset$?, is decidable.
2 Given A_1 and A_2, there is an A such that $\mathsf{T}_\Sigma(\mathsf{A}) = \mathsf{T}_\Sigma(\mathsf{A}_1) \cap \mathsf{T}_\Sigma(\mathsf{A}_2)$.
3 Given A_1 and A_2, there is an A such that $\mathsf{T}_\Sigma(\mathsf{A}) = \mathsf{T}_\Sigma(\mathsf{A}_1) \cup \mathsf{T}_\Sigma(\mathsf{A}_2)$.

Proof. Assume $\mathsf{A} = (Q, \Sigma, q_0, \Delta)$ is a Σ-tree automaton and $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3$ where Σ_1 are the binders, Σ_2 the variables and Σ_3 the other symbols. Let depth(t) be the depth of the Σ-tree t and $|S|$ be the size of the finite set S. We show that if $\mathsf{T}_\Sigma(\mathsf{A}) \neq \emptyset$ then A accepts a Σ-tree t such that depth$(t) \leq (|\Sigma_1||Q| + 1)(|\Sigma||Q| + 1)$. If A accepts t and depth$(t) > l(|\Sigma||Q| + 1)$ then in the accepting run of A on t there are l nodes of t, $n_1, \ldots, n_l$ with the same label in Σ and labelled with the same state $q \in Q$ such that each n_i occurs above n_j when $i < j$. Let $B(i, j)$, where $1 \leq i < j \leq l$, be the set of pairs binders $a \in \Sigma_1$ and states $q' \in Q$ such that there is a node n' between n_i and n_j (excluding n_j) labelled with a and q' in the successful run of A on t such that there is an edge $n' \downarrow n''$ where n'' is n_j or occurs below it in t. Also, let $U(i)$ be the set of pairs binders $a \in \Sigma_1$ and states $q \in Q$ such that there is a node n' above n_i in t labelled with a and q in the successful run of A on t. Clearly, if $B(i, j) \subseteq U(i)$ then there is a smaller Σ-tree t' which is accepted by A: the subtree at node n_i is replaced with the subtree at n_j and any edge $n' \downarrow n''$ where n'' is n_j or below it and n' is between n_i and n_j (excluding n_j) is replaced with an edge $n \downarrow n''$ where n is the node above n_i labelled with the same binder and state as n'. Clearly, if A accepts t then it accepts t'. By simple counting, there must be an i, j with $1 \leq i < j \leq (|\Sigma_1||Q| + 1)$ such that $B(i, j) \subseteq U(i)$. Therefore, non-emptiness is decidable. The other parts of the theorem follow from the usual product and disjoint union of automata (which here includes the binding relations). $\square$

We now extend the definition to *alternating* dependency tree automata.

Definition 28. An alternating dependency Σ-tree automaton $\mathsf{A} = (Q, \Sigma, q_0, \Delta)$ is as in Definition 26 except for the first clause for transitions which now is

1 $qs \Rightarrow (Q_1, \ldots, Q_k)$ where $s \in \Sigma_2 \cup \Sigma_3$, $\mathrm{ar}(s) = k$, $q \in Q$ and $Q_1, \ldots, Q_k \subseteq Q$.

Definition 29. A run of alternating dependency Σ-automaton $\mathsf{A} = (Q, \Sigma, q_0, \Delta)$ on $t \in \mathsf{T}_\Sigma$ is a $(\Sigma \times Q)$-tree whose nodes are pairs (n, α) where n is a node of t and $\alpha \in Q^*$ is a sequence of states, labelled (s, q) if n is labelled s in t and $\alpha = \alpha' q$ which is defined top-down with root (ϵ, q_0) where ϵ is the root of t. Consider a node (n, α) labelled (s, q) of a partial run tree which does not have successors. If $s \in \Sigma_2 \cup \Sigma_3$ and $qs \Rightarrow (Q_1, \ldots, Q_k) \in \Delta$ then the successors of (n, α) are $\{(ni, \alpha q') \,|\, 1 \leq i \leq k \text{ and } q' \in Q_i\}$. If $s \in \Sigma_1$, $n1$ is labelled $s' \in \Sigma_3$ and $qs \Rightarrow q's' \in \Delta$ then $(n1, \alpha q')$ is the successor of (n, α). If $s \in \Sigma_1$, $n1$ is labelled $x \in \Sigma_2$, $m \downarrow n1$ in t, $(m, \alpha' q')$ occurs above or at (n, α) and $(q', q)s \Rightarrow q_1 x \in \Delta$ then $(n1, \alpha q_1)$ is the successor of (n, α). A accepts the Σ-tree t iff there is a run of A on t such that if $(n, \alpha q)$ is a leaf labelled (s, q) of the run tree then either s has arity 0 or $qs \Rightarrow (\emptyset, \ldots, \emptyset) \in \Delta$. Let $\mathsf{T}_\Sigma(\mathsf{A})$ be the set of Σ-trees accepted by A.

A run of an alternating automaton on a Σ-tree t is itself a tree built out of the nodes of t and sequences of states Q^+. There can be multiple copies of nodes of t within a run because a transition applied to a node n $qs \Rightarrow (Q_1, \ldots, Q_k)$ spawns individual copies at ni for each state in Q_i. These automata are alternating as the Q_is can be viewed as conjuncts $\bigwedge_{q \in Q_i} q$ and nondeterminism provides the disjuncts.

Classically, nondeterministic and alternating tree automata accept the same families of trees and the non-emptiness problem for alternating automata is decidable in exponential time (Comon et als 2002). However, this is not the case for dependency tree automata; the non-emptiness problem for alternating dependency tree automata is undecidable as shown in (Ong and Tzevelekos 2009); the proof encodes the observational equivalence problem for finitary PCF in terms of nonemptiness of such automata; this problem is undecidable (Loader 2001A).

We now describe how alternating dependency automata characterise solutions to interpolation problems. Because of the binding relation, we can fix a finite alphabet Σ (as described in Section 3.1). Again we appeal to variable profiles.

Definition 30. Assume the associated binding tree for the interpolation equation E, $xw_1 \ldots w_k = u$, Σ is its alphabet. The dependency tree automaton is $\mathsf{A}_P = (Q, \Sigma, q_0, \Delta)$ where Q is the set of modes (r_i, Γ_i), $r_i \in \mathrm{Sub}(u)$, $q_0 = (u, \emptyset)$ and the transition relation Δ is defined on nodes of the binding Σ-tree by cases on Σ.

1 $(u, \emptyset)@ \Rightarrow (\{(u, \Gamma)\}, \Sigma_1, \ldots, \Sigma_k)$ where $\Sigma_i = \{(r_{i_j}, \Gamma_{i_j}) \,|\, (x_i, r_{i_j}, \Gamma_{i_j}) \in \Gamma\}$
2 $((r, \Gamma), (r', \Gamma'))\lambda\overline{y} \Rightarrow (r', \Sigma)z_i$ if $(z_i, r', \Sigma) \in \Gamma$
3 $(fr_1 \ldots r_k, \Gamma)\lambda\overline{y} \Rightarrow (fr_1 \ldots r_k, \emptyset)f$
4 $(a, \emptyset)\lambda\overline{y} \Rightarrow (a, \emptyset)$
5 $(r, \Gamma)z_j \Rightarrow (Q_1, \ldots, Q_k)$ if $Q_i = \{(r', \Gamma') \,|\, (y_i, r', \Gamma') \in \Gamma\}$ for each $i : 1 \leq i \leq k$ and $\mathrm{ar}(z_j) = k > 0$
6 $(fr_1 \ldots r_k, \emptyset)f \Rightarrow (\{(r_1, \emptyset)\}, \ldots, \{(r_k, \emptyset)\})$

The root of the interpolation tree labelled @ has $k+1$ successors, the first of which is t of the form $\lambda x_1 \dots x_k.t'$ and the $(i+1)$th w_i. The automaton starts with state $(u, \emptyset)$ at @ and then a family of variable profiles Σ_i each of the form $(x_i, r_{i_j}, \Gamma_{i_j})$ is chosen for each $i : 1 \le i \le k$. The state at the node labelled $\lambda x_1 \dots x_k$ is then $(u, \bigcup \Sigma_i)$ and then for each i and for each j there is the state (r_{i_j}, Γ_{i_j}) at the $(i+1)$th successor of @; the same node of the interpolation tree is repeated. Assume the current state is (r', Γ') at node n of the interpolation tree labelled $\lambda \overline{y}$. If $n1$ is labelled with variable z_i and $m \downarrow n1$ then m is labelled $\lambda z_1 \dots z_p$ for some p and the state above (r', Γ') at m has the form (r, Γ) where Γ is a set of profiles for each z_j, $1 \le j \le p$. One of the z_i profiles, (z_i, r', Σ) where the right term r' is as in the state at n is chosen and state (r', Σ') labels $n1$. If $n1$ is labelled f then for the automaton to proceed from node n to $n1$, r' must have the form $fr_1 \dots r_k$. In which case $n1$ is labelled with state $(r', \emptyset)$. Similarly, if $n1$ is labelled with the constant $a : \mathbf{0}$ then r' must be a and $\Gamma' = \emptyset$. If the state is (r, Γ) at node n of the interpolation tree and n is labelled z_j with arity $p > 0$ then Γ consists of sets of y_i profiles, $1 \le i \le p$ for some $y_1, \dots, y_p$. For each y_i profile (y_i, r', Γ') the automaton spawns a copy at ni with state (r', Γ'). Finally, if the state is $(fr_1 \dots r_k, \emptyset)$ at node n of the interpolation tree labelled with f then the automaton proceeds down each successor ni with state $(r_i, \emptyset)$.

The proof of the following characterisation result is very similar to Theorem 21, so we omit it here.

Theorem 23. Assume E is interpolation equation $xw_1 \dots w_k = u$ and Σ is a finite alphabet. The alternating dependency A_E is the dependency Σ-tree automaton (of Definition 30) accepts the Σ binding tree $@tw_1 \dots w_k$ iff t solves E.

6. Decidability

In this section we prove decidability of higher order matching; the proof essentially depends on the game theoretic characterisation of interpolation. The main theorem, Theorem 36 of Section 6.5, is a small solution property: if the interpolation problem has a (canonical) solution then it has a *small* solution that is bounded in the size of the problem.

6.1. *Uniformity properties of game playing*

We now examine further properties of game playing and, in particular, repeating patterns of sequences of positions that build on properties described in Sections 4.3 and 4.4.

We isolate two key technical properties. The first is the following.

Proposition 24. Assume $\pi_1 = n_1[r_1]\theta_1, \dots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$ and n_i is labelled with a variable. If $i < j < m$ and π_j, π_m are children of π_i then there is a $k : j < k < m$ such that $n_j \downarrow n_k$ and π_k is a child of π_j.

Proof. Assume $\pi_1 = n_1[r_1]\theta_1, \dots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$, n_i is labelled with a variable and π_j, π_m are children of π_i for $i < j < m$. Therefore, position π_{i+1}

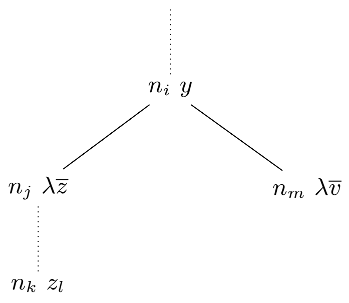


Fig. 15. Situation in Proposition 24

is at a node labelled $\lambda\overline{s}$ for some s and π_{j-1}, π_{m-1} are children of π_{i+1}. Therefore, $\theta_{j-1}(n_{i+1}) = \theta_{m-1}(n_{i+1}) = ((n_i1, \ldots, n_ip), \theta_i)$ for some p. Position π_j is $n_j[r_j]\theta_j$ where $\theta_j = \theta_i\{((n_{j-1}1, \ldots, n_{j-1}q), \theta_{j-1})/n_j\}$ for some q. The look-up table θ_i cannot contain an entry $((m'_1, \ldots, m'_r), \theta'')$ with $\theta''(n_{i+1}) = \theta_{m-1}(n_{i+1})$; so, the only access to such an entry is with a child of π_j via the entry for n_j which ensures that indeed there is a child π_k, $k < m$, of π_j. □

The property is depicted in Figure 15: assume that π_i is at node n_i labelled with y, π_j and π_m, $j < m$, are children of π_i where n_j and n_m can be the same node. There is a child of π_j which occurs before π_m. In the play of Figure 7 on the tree of Figure 6, positions $(3)[ga]\theta_4$ and $(11)[a]\theta_2$ are children of $(2)[ga]\theta_2$. The child of $(3)[ga]\theta_4$ is $(8)[a]\theta_6$. Positions $(3)[u]\theta_4$ and $(3)[ha]\theta_{17}$ of Figure 9 on the tree of Figure 8 at the same node are children of $(2)[u]\theta_2$: their previous positions are at (22) and (26), both labelled with the same variable y_1 (which have a common parent). There are two children of $(3)[u]\theta_4$, the positions $(6)[u]\theta_6$ and $(6)[u_2]\theta_{14}$.

The second property appeals to sequences of positions that "correspond" when they pass through the same nodes of the interpolation tree.

Definition 31. Assume $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$. Two sequences of positions $\pi_i, \ldots, \pi_{i+m}$ and $\pi_j, \ldots, \pi_{j+m}$ correspond if for each $k : 0 \leq k \leq m$, $n_{i+k} = n_{j+k}$.

Proposition 25. Assume $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$, $i < j < k$, n_i is a binder such that $n_i \downarrow n_j$, $n_i \downarrow n_k$, n_j, n_k are both labelled with the same variable and π_j, π_k are both children of π_i. There exists $m < k - (j + 1)$ such that π_{j+m+1} is a child of π_j and either

1 $\pi_{k+1}, \ldots, \pi_{k+m}$ and $\pi_{j+1}, \ldots, \pi_{j+m}$ correspond and π_{k+m+1} is a child of π_k, or
2 there is $p < m$ such that $\pi_{k+1}, \ldots, \pi_{k+p}$ and $\pi_{j+1}, \ldots, \pi_{j+p}$ correspond, n_{k+p} is labelled with a constant f and either $[r_{k+p}]$ is final or $n_{k+p+1} \neq n_{j+p+1}$.

Proof. Assume $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$, $n_i \downarrow n_j$ and $n_i \downarrow n_k$ and n_j, n_k are both labelled with the same variable and π_j, π_k are both children of π_i for $j < k$. Therefore, $\theta_j(n_i) = \theta_k(n_i) = \theta_i(n_i) = ((n'_1, \ldots, n'_q), \theta')$ for some $n'_1, \ldots, n'_q$

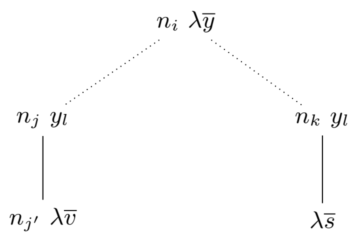


Fig. 16. Situation in Proposition 25

and θ'. So, $n_{j+1} = n_{k+1} = n'_l$ for some l and $\theta_{j+1} = \theta'\{((n_j1, \ldots, n_jr), \theta_j)/n'_l\}$ and $\theta_{k+1} = \theta'\{((n_k1, \ldots, n_kr), \theta_j)/n'_l\}$ for some r. So, positions π_{j+1}, π_{k+1} correspond as do their continuations because θ_{j+1}, θ_{k+1} have the same entries except for the binder n'_l. Now the result follows; first there is child of π_j at π_{j+m+1} for some $m < k-(j+1)$ because otherwise there could not be the position π_k. Either $\pi_{k+1}, \ldots, \pi_{k+m}$ and $\pi_{j+1}, \ldots, \pi_{j+m}$ correspond and π_{k+m+1} is a child of π_k or the continuations from π_{j+1}, π_{k+1} correspond until they reach a constant and the final state occurs or a different $\forall$ choice is made. $\square$

The second uniformity property is pictured in Figure 16 when $j' = j + m + 1$: assume that $n_i \downarrow n_j$ and $n_i \downarrow n_k$ and π_j, π_k are children of π_i, $j < k$; nodes n_j, n_k can be the same. First, and compare with Proposition 24, there must be a child π_{j+m+1} of π_j. However, more than this: either there are corresponding sequences of positions $\pi_{j+1}, \ldots, \pi_{j+m}$ and $\pi_{k+1}, \ldots, \pi_{k+m}$ and π_{k+m+1} is a child of π_k or there are corresponding prefixes of these sequences until a node labelled with a constant is reached and either there is a final state or a different $\forall$ choice. There are numerous instances of this property in the game of Figure 9 on the tree of Figure 8. A simple case is that $(2)[u]\theta_2$ and $(4)[u]\theta_4$ are children of $(1)[u]\theta_2$: position $(3)[u]\theta_4$ is a child of $(2)[u]\theta_2$ and $(21)[u]\theta_3$,$(22)[u]\theta_3$ corresponds to $(21)[u]\theta_5$,$(22)[u]\theta_5$ and so the next position $(5)[u]\theta_6$ is a child of $(4)[u]\theta_4$. A second example is when $n_j = n_k$: $(6)[u]\theta_6$ and $(6)[u_2]\theta_{14}$ are both children of $(3)[u]\theta_4$; the sequences $(23)[u]\theta_7, \ldots, (28)[u_1]\theta_{11}$ and $(23)[u_1]\theta_{22}, \ldots, (28)[ha]\theta_{20}$ correspond and their next positions $(7)[u_1]\theta_6$, $(7)[ha]\theta_{14}$ are children of the positions at (6).

To apply these properties, first we structurally identify particular nodes of interpolation trees.

Definition 32. Let E be $xw_1 \ldots w_k = u$ and assume the game $\mathsf{G}(t, E)$. A node of t, or w_i, is level 1 if it is bound by the initial $\lambda x_1 \ldots x_k$ of t, or the initial $\lambda y_1 \ldots y_p$ of w_i. A node of t or w_i is level $j + 1$ if it is bound by the successor node of a level j node.

Solution terms of 4th or 5th order interpolation equations contain at most two levels of nodes: those labelled with the x_is are level 1 and those with the z_is are level 2. In the tree of Figure 8, nodes (2), (4), (22), (24) and (26) are level 1; other nodes labelled with a variable are level 2. In the general case, a term of order $2n$ or $2n+1$ can contain nodes at each level k for $1 \leq k \leq n$.

Definition 33. Assume $n' \downarrow n$, $n' \downarrow m$ and n, m are distinct nodes of a binding tree labelled with the same variable. Node m is embedded if m occurs somewhere below n in the tree.

In Figure 8, node (4) is embedded because it is below (2) and both (1) $\downarrow$ (2) and (1) $\downarrow$ (4); nodes (10) and (26) are also embedded.

Definition 34. Assume nodes n, m of a binding tree are both labelled with a variable. Node n is an ancestor of m, or m is a descendent of n, if $n = m$ or n' is a successor of n, $n' \downarrow m'$ and m' is an ancestor of m. Nodes n, m belong to the same family if they have a common ancestor. Node n is end if it has no descendents except itself.

Node (2) of Figure 8 is an ancestor of (10). Nodes (4), (8) and (16) belong to the same family. Nodes (6), (8), (10), (24) and (26) are end nodes (as are leaves such as (16) and (28)).

We now examine some particular uniformity features of plays of a game. The first follows from Proposition 24.

Fact 26. Assume $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$. If n_i is end then there is at most one position π_j, $j > i$, that is a child of π_i.

The next property follows from Proposition 25.

Fact 27. Assume $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$, nodes n_i, n_j of t are both level 1 and labelled with the same variable, π_{i+k} is the first position that is a child of π_i and n_{i+k} is the mth successor of n_i.

1 If $r_i = r_{i+k}$ then π_{j+k} is the first position that is a child of π_j, $r_j = r_{j+k}$, sequences $\pi_{i+1}, \ldots, \pi_{i+(k-1)}$ and $\pi_{j+1}, \ldots, \pi_{j+(k-1)}$ correspond and n_{j+k} is the mth successor of n_j.

2 If $r_i \neq r_{i+k}$ and $\pi_{j+k'}$ is a child of π_j then $r_j \neq r_{j+k'}$.

In Figure 8, nodes (2) and (4) are level 1 and labelled with x_1. Consider any position in the play of Figure 9 at these nodes, such as $(2)[u]\theta_2$, $(4)[ha]\theta_{17}$ and $(4)[a]\theta_{24}$: the earliest children of these positions is at the initial successors, $(3)[u]\theta_4$, $(5)[ha]\theta_{19}$ and $(5)[a]\theta_{26}$, and the intermediate sequences of positions correspond.

The next property is a consequence of both propositions.

Fact 28. Assume $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$, π_i and π_j are both children of $\pi_{i'}$, n_i and n_j are nodes of t labelled with the same variable, n_j is below n_i, π_{i+k} is the first child of π_i and n_{i+k} is the mth successor of n_i.

1 If $r_i = r_{i+k}$ then π_{j+k} is a child of π_j, $r_j = r_{j+k}$, n_{j+k} is the mth successor of n_j and $\pi_{i+1}, \ldots, \pi_{i+(k-1)}$ and $\pi_{j+1}, \ldots, \pi_{j+(k-1)}$ correspond.

2 If $r_i \neq r_{i+k}$ and $\pi_{j+k'}$ is a child of π_j then $r_j \neq r_{j+k'}$.

3 If n_i, n_j are end nodes then n_j is below the mth successor of n_i.

In Figure 8 node (4) is embedded as it is below (2): Position $(3)[u]\theta_4$ of Figure 9 is the first child of $(2)[u]\theta_3$ and so $(5)[u]\theta_6$ is a child of $(4)[u]\theta_4$. Nodes (6) and (10) are end and (10) is embedded, as it is below the only successor of (6). Positions $(6)[u]\theta_6$ and $(10)[ha]\theta_6$ are both children of $(3)[u]\theta_4$; the children of these positions are $(7)[u_1]\theta_6$ and

$(11)[a]\theta_6$ and their intermediate sequences of positions correspond (as there is the same $\forall$ choice at node (18)).

6.2. *Two transformations*

We now define two local transformations on terms that preserve solutions of an interpolation problem. A transformation T of t has the form $t[s/n]$ where $s : \mathbf{0}$ is also a term and n is a node of t labelled with a variable or a constant: this transformation produces the term t' which is the result of replacing the subtree rooted at n in t with s; we write tTt' if t' is the result of applying T to t.

Definition 35. Assume node n of t is labelled with a variable or a constant $f : A \neq \mathbf{0}$ and whenever $m[r]\theta$ is a position in a play of $\mathsf{G}(t, E)$, $m \neq n$. The transformation T_1 is $t[b/n]$.

If no position of a play of $\mathsf{G}(t, E)$ is at the node n of t then the subtree rooted there can be replaced without affecting game playing; transformation T_1 replaces the redundant subtree (whose root is labelled with a variable or a constant which is not of ground type) with the singleton node labelled $b : \mathbf{0}$ (of Definition 10).

Fact 29. If t solves E and tT_1t' then t' solves E.

The second transformation uses the notion of an end branch which generalises Definition 34 of end node.

Definition 36. The sequence of nodes $m_1, \ldots, m_{2p}$ is end in t if m_1 is labelled with a variable, for each $i : 1 \leq i < 2p$, m_{i+1} is a successor of m_i and no node beneath m_{2p} in t is bound within this sequence.

For instance, whereas the branch $(4), (5), (6), (7), (8), (9)$ of Figure 8 is end, the sequence $(2), (3), (4), (5)$ is not because nodes (6), (8) and (10) are bound by (3) and (5). If node n is end, and n' is a successor of n then the branch n, n' is end according to this more general definition.

Next we define when an end sequence of nodes in t is redundant in the game $\mathsf{G}(t, E)$; which appeals to compatibility of positions of Definition 23.

Definition 37. Assume $m_1, \ldots, m_{2p}$ is an end sequence in t.

1 This sequence is redundant in the suffix $\pi_i = n_i[r_i]\theta_i, \ldots, \pi_l = n_l[r_l]\theta_l$ of $\pi \in \mathsf{G}(t, E)$ if

   (a) for all j, $i \leq j \leq l$, $n_j \neq m_1$, or

   (b) π_j is the first position, $j \geq i$, such that $n_j = m_1$ and there is a later position π_k with $n_k = m_{2p}$, π_j is compatible with π_k, $r_k = r_j$ and this sequence is redundant in the suffix $\pi_{k+1}, \ldots, \pi_l$.

2 This sequence is redundant in the game $\mathsf{G}(t, E)$ if it is redundant in every play $\pi \in \mathsf{G}(t, E)$.

The end sequence $m_1, \ldots, m_{2p}$ is redundant in a suffix of a play in $\mathsf{G}(t, E)$ if m_1 does not occur in this suffix or there is a first position at m_1 and a later position at m_{2p} such that the first position occurs in the sequence of branch positions of the second, these positions share the same state and the end sequence is also redundant in the suffix after the second position. It is redundant in $\mathsf{G}(t, E)$ if it is redundant in every play. This inductive definition is well defined (as the length of the suffix reduces in the definition).

Definition 38. Assume the end sequence $m_1, \ldots, m_{2p}$ is redundant in $\mathsf{G}(t, E)$ and t' is the subtree rooted at $m_{2p}1$ (directly below m_{2p}). The transformation T_2 is $t[t'/m_1]$.

This transformation disposes of the branch $m_1, \ldots, m_{2p}$ in t (and all branches below m_i that do not pass through m_{2p}) by replacing the subtree at m_1 with t' that occurs directly below m_{2p}: this preserves being a closed term (of the right type) because all free variable occurrences in t' are bound above m_1 in t.

Proposition 30. If t solves E and tT_2t' then t' solves E.

Proof. Assume that t solves E, the sequence $m_1, \ldots, m_{2p}$ is redundant in $\mathsf{G}(t, E)$, t'' is the subtree rooted at $m_{2p}1$ and $t' = t[t''/m_1]$. We show that t' solves E. First, t' is a properly formed closed term with the same type as t; by definition if node m occurs below m_{2p} in t and $n \downarrow m$ then either n occurs above m_1 or below m_{2p} because $m_1, \ldots, m_{2p}$ is an end sequence in t. We now show that $\forall$ loses the game $\mathsf{G}(t', E)$. We assume that the nodes in t' retain the same names as in t. Consider any play $\pi \in \mathsf{G}(t, E)$. If no position in π occurs at m_1 then this end sequence is redundant in π as required. Otherwise, assume π_{i_1} is the first position at m_1. By Definition 37, therefore, there is a later position π_{j_1} at m_{2p} such that $\theta_{i_1} \leq \theta_{j_1}$ using Proposition 19, and $\pi_{i_1}, \ldots, \pi_{j_1}$ does not contribute as their states are the same. Therefore, by repeating this, we can decompose π as follows: $\pi = \sigma_1\pi_{i_1}, \ldots, \pi_{j_1}\sigma_2\pi_{i_2}, \ldots, \pi_{j_2}, \ldots, \sigma_n\pi_{i_n}, \ldots, \pi_{j_n}\sigma_{n+1}$ where each π_{i_k} is the first position after $\pi_{j_{k-1}}$ at m_1 and π_{j_k} is at m_{2p} using Definition 37 and no position in σ_{n+1} is at m_1. We claim that $\sigma = \sigma_1 \ldots \sigma_{n+1}$, modulo minor changes to the look-up tables, is a play of $\mathsf{G}(t', E)$. Using Proposition 24, if a position in σ_j is below m_1 then it must also be below m_{2p}. Conversely, assume that σ is a play in $\mathsf{G}(t', E)$. Then there is a play $\pi \in \mathsf{G}(t, E)$ such that $\sigma = \sigma_1 \ldots \sigma_{n+1}$ and $\pi = \sigma_1\pi_{i_1}, \ldots, \pi_{j_1}\sigma_2\pi_{i_2}, \ldots, \pi_{j_2}, \ldots, \sigma_n\pi_{i_n}, \ldots, \pi_{j_n}\sigma_{n+1}$, modulo minor changes to look-up tables, such that each π_{i_k} is at m_1 and π_{j_k} is at m_{2p} and $\theta_{i_k} \leq \theta_{j_k}$ and $\pi_{i_k}, \ldots, \pi_{j_k}$ does not contribute. □

Consider the tree in Figure 6 and its single play in Figure 7. The transformation T_2 applies to the end sequence $(6), (7)$ because there is just one position at (6) and then later its child at (7) (with the same state); so, node (8) becomes the successor of (5). Similarly, the end sequence $(12), (19)$ is also redundant; so T_2 allows the singleton subtree (20) to replace the subtree at (12). The result is a smaller solution term. T_2 does not apply to the sequence $(2), (3)$ because (3) binds (8).

Consider Figure 21 with the single play of Figure 20. The sequence $(8), (11)$ in the solution term is end because no node below (11) is bound by it. This sequence is also redundant; there is the first position $(8)[ga]\theta_6$ at (8) and its child $(11)[ga]\theta_8$ and then

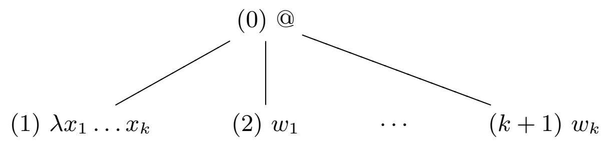


Fig. 17. Subtree for the first stage of partition

later $(8)[a]\theta_{13}$ with its child $(11)[a]\theta_{15}$. Therefore, replacing node (8) with the tree of the subterm zs_1s_2 is a smaller term that solves the equation of Example 16.

The transformations T_1 and T_2, by themselves, are not sufficient for proving decidability of matching (by showing a small solution property as in Theorem 36). However, using Fact 28, they do provide immediate bounds on the number of embedded end nodes within a smallest solution term.

Proposition 31. Assume t is a smallest solution to E whose goal is u. If $m_1, \dots, m_p$ are embedded end nodes of t then $2p \leq |u|$.

Proof. Assume $m_1, \dots, m_p$ are embedded end nodes. The result uses Fact 28 as follows. Consider each m_k; if m_k is embedded because it is below the lth successor of m'_k (which makes it embedded) then m_k, m_kl is end; it is either redundant for every play in $\mathsf{G}(t, E)$ or there is a sequence of positions involving m_k that contributes and an earlier sequence of positions involving m'_k that also contributes. □

This proposition shows in the 5th order case that there cannot be an arbitrary amount of "stacking" of the same variable z_n in a smallest solution term t; see the discussion in Example 11. However, this does not preclude the possibility that a solution term contains arbitrarily many level 1 nodes and more generally, at higher orders, arbitrarily many level k nodes that are not end nodes.

6.3. *Partitioning play*

In this section we describe an important step of the decidability proof, a partition of each play $\pi \in \mathsf{G}(t, E)$ when t is a solution term for E.

For the first stage of the partition of $\pi \in \mathsf{G}(t, E)$, we examine the initial positions on the subtree in Figure 17; instead of the full tree t, there is just its root node (1) labelled $\lambda x_1 \dots x_k$. Therefore, this stage of the partition consists of the first two positions $\pi_1 = (0)[u]\theta_1$ and $\pi_2 = (1)[u]\theta_2$ where θ_2 is $\{((2, \dots, k+1), \theta_1)/1\}$.

If π is not completed after stage N of the partition then there is a subtree of t, a node n' of this subtree labelled $\lambda\overline{z}'$ where stage N finishes, as depicted in Figure 18. The next position, $n[r]\theta$, in π is at the node $n = n'1$ labelled with v; this is the first position of stage $N+1$ of the partition; the node n and its successors $n1, \dots, np$, if there are any, are included as part of the subtree of t at stage $N+1$ (if they were not already nodes at stage N); see Figure 18. What happens next, according to the moves in Figure 5, depends on the label v at n.

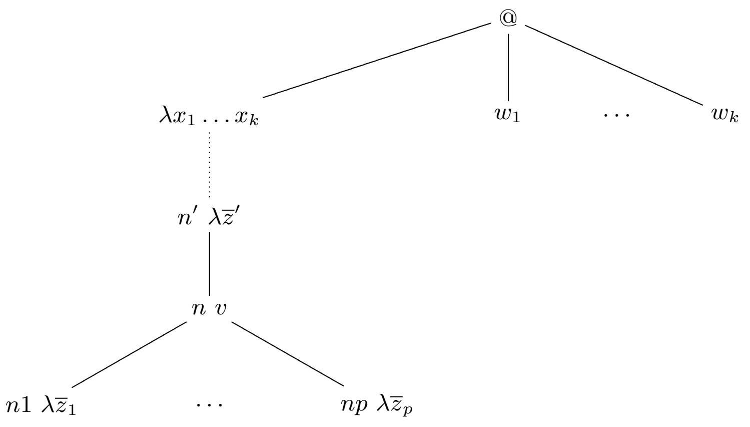

@
$\lambda x_1 \dots x_k$
w_1
$\dots$
w_k
$n'\ \lambda\overline{z}'$
$n\ v$
$n1\ \lambda\overline{z}_1$
$\dots$
$np\ \lambda\overline{z}_p$

Fig. 18. Subtree for the $N+1$th stage of partition

— If v is a constant $a : \mathbf{0}$ then $r = a$ and the next position is the final position in π; stage $N+1$ consists of the two positions $n[a]\theta$, $n[\exists]\theta$.

— If v is a constant f with arity > 0 then $r = fr_1 \dots r_q$ and $\forall$ chooses a direction j; stage $N+1$ consists of the two positions $n[r]\theta$, $nj[r_j]\theta$.

— If v is a variable z_l then it is bound by a $\lambda\overline{z}$ at a node somewhere between the root of t and n (including the possibility that $\lambda\overline{z}$ is $\lambda x_1 \dots x_k$ and z_l is x_i for some i); see Figure 19. Therefore, the next position in the partition is $m_1l[r]\theta'$ for some θ' (m_1l is $i+1$ when z_l is x_i) and the remaining positions of stage $N+1$ are defined iteratively by cases on the label at the node m of w'_i; initially, m is m_1l1, directly beneath m_1l.

  – If m is labelled $a : \mathbf{0}$ then the next two positions of π at m complete stage $N+1$ (and π).

  – If m is labelled f with arity > 0 then the current state has the form $[fr_1 \dots r_q]$, $\forall$ chooses a direction j and the two positions of π at m and mj (with state $[r_j]$) are included as part of stage $N+1$; the iteration continues with node $m = mj1$, the node directly beneath mj.

  – If m is labelled with a variable s_q and π_j is the next position at m then m is bound by a $\lambda\overline{s}$ on the branch between $\lambda\overline{y}$ and m in w_i. Assume π_r is the parent of π_j. We examine whether there is at least one previous position (before π_j) that is also a child of π_r and at a node also labelled s_q. If not then the positions π_j, π_{j+1} complete stage $N+1$: node n_{j+1} is in the subtree of t at stage $N+1$. Otherwise, there are previous positions, π_{i_k} at nodes m_k labelled s_q that are children of π_r for $k > 0$. Now we invoke Proposition 25: for each k, there is a position π_{j_k}, $j_k < i_{k+1}$, that is a child of π_{i_k} at node m_kl_k for some l_k. There are two cases. First, for some k, there is a corresponding sequence of positions from π_{j+1} to $\pi_{j-1+(j_k-i_k)}$ and $\pi_{j+(j_k-i_k)}$ is a child of π_j at node ml_k; so, stage $N+1$ includes the positions $\pi_j, \pi_{j+1}, \dots, \pi_{j+(j_k-i_k)}$ and the iteration continues with $m = ml_k1$, the node directly beneath ml_k. The second case is that for each k, there is a $p_k < j_k - i_k$ such that $\pi_{j+1}, \dots, \pi_{j+p_k}$ and $\pi_{i_k+1}, \dots, \pi_{i_k+p_k}$ correspond and $n_{j+p_k+1} \neq n_{i_k+p_k+1}$ and $n_{i_k+p_k}$ is labelled with a constant. Consider the

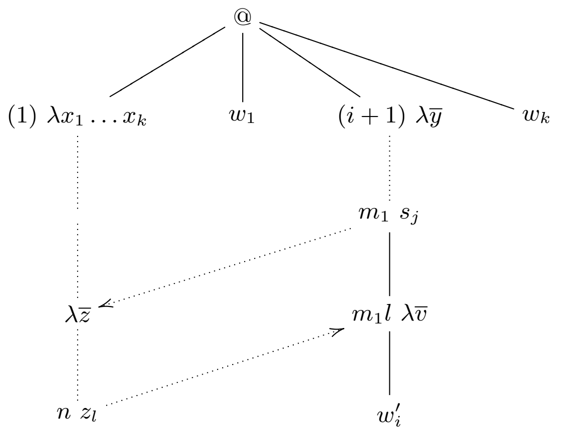

@
(1) $\lambda x_1 \ldots x_k$
w_1
$(i+1)$ $\lambda\overline{y}$
w_k
m_1 s_j
$\lambda\overline{z}$
$m_1 l$ $\lambda\overline{v}$
n z_l
w'_i

Fig. 19. When play is at z_l and jumps to $\lambda\overline{v}$

largest such p_k; stage $N+1$ finishes at the first position π_r, $r > j + p_k$, such that node n_r is a node in the subtree of t at Stage $N+1$ which is labelled with $\lambda\overline{z}$ for some $\overline{z}$.

The initial position π_i at stage $N+1$ of the partition is at a node n of the subtree of t that is labelled with a constant or a variable. The partition and the play both finish with π_{i+1} if it is a constant $a : \mathbf{0}$. If it is a constant f with arity greater than 0, stage $N+1$ ends at the child position π_{i+1}, at a successor node of n depending on ∀'s choice. If it is a variable then play jumps into a w_i, descends a branch and either finishes at a node labelled with a constant $a : \mathbf{0}$ or reaches a node labelled with a variable s_q in w_i; assume its parent position is π_r. If there are no previous positions at nodes also labelled s_q with the same parent π_r, the partition will then end at the next position which is in the subtree of t at stage $N+1$: it is at a node labelled with a $\lambda\overline{z}$ (for some $\overline{z}$) which is a successor of a node in the same family as n. Otherwise there are previous positions at nodes labelled s_q with the same parent π_r. Proposition 25 is invoked as the next position after any such child is at node n' in the subtree of t. Stage $N+1$ continues by examining corresponding sequences of positions from n': if a different ∀ choice is encountered then stage $N+1$ finishes at the first node in the subtree of t labelled with a $\lambda\overline{z}$ for some $\overline{z}$; otherwise, there is a later position which is a child of the position at s_q and the definition of stage $N+1$ is iterated as play further descends the branch of w_i.

Example 16. The equation, $xw_1w_2 = ga$ is 5th-order with $w_1 = \lambda y_1 y_2.y_2 y_1 y_1$, $w_2 = \lambda y_3.y_3(y_3 a \lambda w_1 w_2.w_1)(\lambda v_1 v_2.g v_2)$, $x : ((\mathbf{0},(\mathbf{0},\mathbf{0},\mathbf{0})),((\mathbf{0},(\mathbf{0},\mathbf{0},\mathbf{0}),\mathbf{0}),\mathbf{0}),\mathbf{0})$ and $g : (\mathbf{0},\mathbf{0})$. A solution term t is presented as part of the interpolation tree in Figure 21. The single play on this tree and the stages of its partition, S_i, are presented in Figure 20. S_1 consists of the first two positions. For S_2 play starts at position $(2)[ga]\theta_2$, jumps to (23) and descends to (24); as this position is a child of the position at (23) there cannot be previous positions that are also children of it; so, S_2 finishes at the next position at (3). In general, any stage that starts at a level 1 node ends at a successor node except

S_1	$(0)[ga]\theta_1 = \emptyset$	S_7	$(6)[a]\theta_4$
	$(1)[ga]\theta_2 = \{((17,23),\theta_1)/1\}$		$(25)[a]\theta_3$
S_2	$(2)[ga]\theta_2$		$(26)[a]\theta_3$
	$(23)[ga]\theta_3 = \{((3),\theta_2)/23\}$		$(3)[a]\theta_{10} = \theta_2\{((27,29),\theta_3)/3\}$
	$(24)[ga]\theta_3$		$(4)[a]\theta_{10}$
	$(3)[ga]\theta_4 = \theta_2\{((25,31),\theta_3)/3\}$		$(17)[a]\theta_{11} = \{((5,7),\theta_{10})/17\}$
S_3	$(4)[ga]\theta_4$		$(18)[a]\theta_{11}$
	$(17)[ga]\theta_5 = \{((5,7),\theta_4)/17\}$		$(7)[a]\theta_{13} = \theta_{10}\{((19,21),\theta_{11})/7\}$
	$(18)[ga]\theta_5$		$(8)[a]\theta_{13}$
	$(7)[ga]\theta_6 = \theta_4\{((19,21),\theta_5)/7\}$		$(17)[a]\theta_{14} = \{((9,11),\theta_{13})/17\}$
S_4	$(8)[ga]\theta_6$		$(18)[a]\theta_{14}$
	$(17)[ga]\theta_7 = \{((9,11),\theta_6)/17\}$		$(11)[a]\theta_{15} = \theta_{13}\{((19,21),\theta_{14})/11\}$
	$(18)[ga]\theta_7$		$(12)[a]\theta_{15}$
	$(11)[ga]\theta_8 = \theta_6\{((19,21),\theta_7)/11\}$		$(29)[a]\theta_{16} = \theta_3\{((13,15),\theta_{15})/29\}$
S_5	$(12)[ga]\theta_8$		$(30)[a]\theta_{16}$
	$(31)[ga]\theta_9 = \theta_3\{((13,15),\theta_8)/31\}$		$(13)[a]\theta_{15}$
	$(32)[ga]\theta_9$	S_8	$(14)[a]\theta_{15}$
	$(33)[a]\theta_9$		$(19)[a]\theta_{11}$
	$(34)[a]\theta_9$		$(20)[a]\theta_{11}$
	$(15)[a]\theta_8$		$(5)[a]\theta_{10}$
S_6	$(16)[a]\theta_8$	S_9	$(6)[a]\theta_{10}$
	$(21)[a]\theta_5$		$(27)[a]\theta_3$
	$(22)[a]\theta_5$		$(28)[a]\theta_3$
	$(5)[a]\theta_4$		$(28)[\exists]\theta_3$

Fig. 20. Stages of the partition of the single play of Example 16

when it is the final stage of the partition. S_3 and S_4 involve sequences of positions that start at a level 1 node labelled x_1, jump to (17) and return to a successor node in t via (18). After S_4, the subtree of t consists of nodes (1) – (5), (7), (8), (9) and (11). S_5 also includes the level 2 node (12) and its successors (13) and (15). From (12), because $\theta_8(3) = ((25, 31,), \theta_3)$, play jumps to (31), proceeds to (32), (33), (34) and returns to t at (15). For S_6, just node (16) is added; so play jumps from (16) to (21), (22) and then to (5) in t. Node (6) is added for S_7; play jumps to (25) and then proceeds to (26) labelled y_3 which is a child of $(23)[ga]\theta_3$ and there is an earlier position $(24)[ga]\theta_3$ also at a node labelled y_3 which is also a child of $(23)[ga]\theta_3$; the position $(31)[ga]\theta_9$ is a child of $(24)[ga]\theta_3$. The sequences $(3)[ga]\theta_4, \ldots, (12)[ga]\theta_8$ and $(3)[a]\theta_{10}, \ldots, (12)[a]\theta_{15}$ correspond; and $(29)[a]\theta_{16}$ is a child of $(26)[a]\theta_3$; so, play at S_7 continues with node (30) before ending at (13). S_8 includes node (14); play jumps to (19) and then proceeds to position $(20)[a]\theta_{11}$; although earlier there is the position $(22)[a]\theta_5$, these two positions (both at a node labelled y) can not share a parent because $y : \mathbf{0}$ (see Proposition 25); so S_8, as does S_6, finishes at node (5). For the final stage, play jumps from (6) to (27) and ends at (28). □

Definition 39. Assume $\pi_i = n_i[r_i]\theta_i, \ldots, \pi_j = n_j[r_j]\theta_j$ is stage N of the partition of

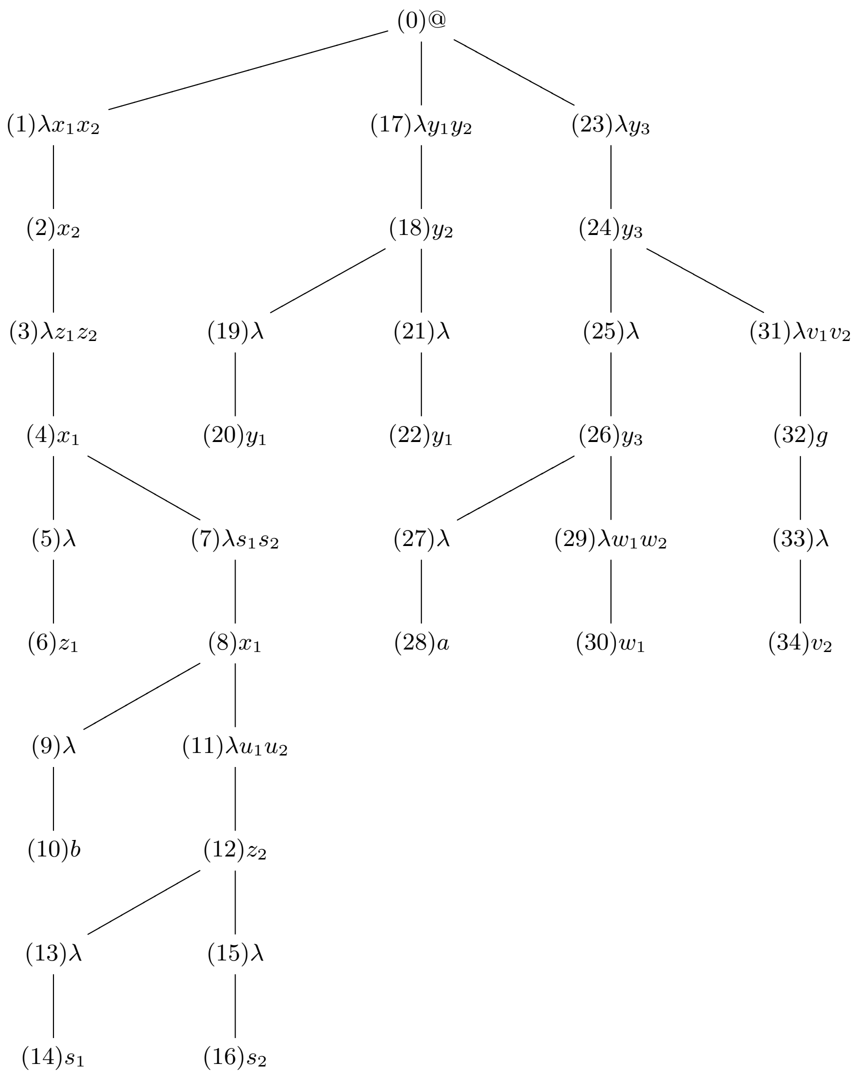


Fig. 21. The interpolation tree of Example 16

$\pi \in \mathsf{G}(t, E)$. Stage N is final if $r_j = \exists$. If stage N is not final then it contributes (to the solution) when $r_i \neq r_j$. If stage N is final then it contributes when $r_i \neq r_{j-1}$.

A stage contributes to the solution if it contains a position that is at a node labelled with a constant f. In any play, the number of stages that contribute to a solution is at most $\text{depth}(u) - 1$, where u is the goal term. In the game in Figure 20, as $u = ga$ there is a single stage, S_5, that contributes; its initial state is $[ga]$ and its last state is $[a]$; stage S_9 is final.

If a stage of a partition starts at a node labelled with a variable then it jumps into a w_i and descends a branch of it. The following definition identifies when a stage is standard which uses Definition 23 of branch positions.

Definition 40. Assume $\pi_i = n_i[r_i]\theta_i, \ldots, \pi_j = n_j[r_j]\theta_j$ is stage N of the partition of $\pi \in \mathsf{G}(t, E)$. Stage N is standard if there is $\pi_{j_1}, \ldots, \pi_{j_p}$ which is the sequence of branch positions of π_{j-1} to π_{i+1} and $n_{j_1}, \ldots, n_{j_p}$ is a branch in w_q for some q.

Every stage of the partition in Figure 20 is standard; for instance, for S_7 there is the sequence $(25)[a]\theta_3$, $(26)[a]\theta_3$, $(29)[a]\theta_{16}$ and $(30)[a]\theta_{16}$.

Proposition 32. Assume $\pi_i = n_i[r_i]\theta_i, \ldots, \pi_j = n_j[r_j]\theta_j$ is stage N of the partition of $\pi \in \mathsf{G}(t, E)$.

1 If n_i is labelled $a : \mathbf{0}$ then stage N is final.
2 If stage N is not final and n_i is a level 1 node or labelled with a constant then π_j is a child of π_i.
3 If stage N does not contribute and n_i is an embedded node then π_j is a child of π_i.
4 If stage N is standard and is not final then n_j is the successor of a node in the same family as n_i.
5 If stage N does not contribute and n_i is labelled with a variable then stage N is standard.

Proof. Let $\pi_i = n_i[r_i]\theta_i, \ldots, \pi_j = n_j[r_j]\theta_j$ be stage N of the partition of $\pi \in \mathsf{G}(t, E)$. 1. Clearly, if n_i is labelled $a : \mathbf{0}$ then $r_i = a$ and $r_{i+1} = \exists$ as t solves E. 2. Assume stage N is not final. If n_i is labelled with a constant then $n_j = n_{i+1}$ and π_j is a child of π_i. If n_i is a level 1 node labelled x_q then n_{i+1} is the root of w_q; play proceeds down a branch of w_q until n_{j-1} where $n_{i+1} \downarrow n_{j-1}$; clearly, no earlier position could be a child of π_{i+1} and so, π_j is a child of π_i. 3. Assume that n_i is an embedded node. Therefore, there is a node n above n_i labelled with the same variable bound with the same binder at node m. Consider the branch positions for π_i of Definition 23; it includes the positions π_{i_m} at node m, π_{i_n} at node n and π_i where π_{i_m} is the parent of both π_{i_n} and π_i. Now we can use Fact 28; the continuations from π_{i+1} and π_{i_n+1} correspond until their first children at π_{i+1+p} and π_{i_n+1+p} for some p (as stage N does not contribute); as π_{i+1+p} is the first child of π_{i+1} the next position $\pi_{i+2+p} = \pi_j$ completes stage N and is a child of π_i. 4. Assume that stage N is standard and not final. We show that n_j is a successor of a node in the same family as n_i. First, n_i is labelled with a variable. So play jumps into a w_q and so n_{i+1} is labelled with a $\lambda\overline{v}$ for some v; node n_{i+2} is labelled with a variable s_q whose binder $\lambda\overline{s}$ is on the branch between the root of w_q and n_{i+2} (and could be the root or node n_{q+1}). Consider the sequence of branch positions for π_{i+2} which includes π_{i+1}, the parent π_r of π_{i+2} and the position $\pi_{q'}$ at the root of w_i; clearly, nodes n_i and n_{r-1} belong to the same family as they have the common ancestor $n_{q'} - 1$. Furthermore, node n_{i+3} is a successor of n_{r-1}. If it is the final position of stage N then the argument is clear. Otherwise by stage construction as it is standard and is not final play proceeds to a child of π_{i+3} and returns to a successor of n_{i+2} and now the argument is repeated for subsequent variable occurrences along the branch of w_q. 5. is clear. □

S_2, S_3 and S_4 of the partition in Figure 20 start at a level 1 node and end at a successor with a child position. S_4 also starts at an embedded node. Except for the final stage, every stage of the partition in this example finishes at a successor of a node in the same family as the start node: for instance, in the case of S_7, it starts at node (6) labelled z_1 and ends at a successor of (12) which is labelled z_2; node (2) is the common ancestor of (6) and (12).

We next examine the cases where a stage does contribute to the solution.

Proposition 33. Assume $\pi_i = n_i[r_i]\theta_i, \ldots, \pi_j = n_j[r_j]\theta_j$ is stage N of the partition of $\pi \in \mathsf{G}(t, E)$ which contributes to the solution. Then there is a node n_p, $i \leq p < j$, labelled with a constant f with arity > 0 and one of the following holds.

1 $p = i$, $n_j = n_{i+1}$ and π_j is a child of π_i,
2 n_p is below n_{i+1} in w_q for some q,
3 for some position $\pi_k = n_k[r_k]\theta_k$ before π_i, $n_k = n_p$.

Proof. Let $\pi_i = n_i[r_i]\theta_i, \ldots, \pi_j = n_j[r_j]\theta_j$ be stage N of the partition of $\pi \in \mathsf{G}(t, E)$ and assume it contributes to the solution. The first possibility is that n_i is labelled with a constant f and, so, $r_i = fs_1 \ldots s_l$ and $r_{i+1} \neq r_i$. Otherwise, n_i is labelled with a variable; therefore, n_{i+1} is in w_q for some q. A second possibility is that play proceeds down a branch of w_q and reaches a node labelled with a constant f. The only other possibility is that within stage N, there is a position at a node below n_{i+1} labelled with a variable and there was an earlier position at a node labelled with the same variable, and both these positions have the same parent; then there is a sequence of corresponding positions which includes a node n_p labelled with a constant f; thus, there are at least two different positions at n_p. □

S_5 of Figure 20 illustrates the second case in this proposition. When a stage of the partition is not standard then it is because of the third case in this proposition as a different ∀ choice is made at a node than was made earlier at that node.

6.4. *Contractions and prefixes*

The decidability proof is completed using a notion of unfolding which is reminiscent of unravelling a graph into a tree. The idea is to replace a stage of the partition by including prefixes which correspond to previous stages. This more complex transformation works on individual stages of a partition. The following definition uses Definition 23 of a sequence of branch positions.

Definition 41. Assume $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$, n_i is labelled with a constant or variable and π_j is a child of π_i. We now define the contraction of π_i to π_j inductively (on the distance $j - i$) and by cases on the label at n_i (and whether it occurs in t or w_q for some q).

1 n_i is labelled with a constant. The contraction is π_i, π_{i+1} (as $j = i + 1$).

2 n_i is is labelled with a variable and is a node of w_q for some q. The contraction is $\pi_i, \pi_{i+1}, \pi_{j-1}, \pi_j$.

3 n_i is labelled with a variable and is a node of t. If $\pi_{i_1}, \ldots, \pi_{i_{2p}}$ is the sequence of branch positions of π_{j-1} to π_{i+1} then the contraction is $\pi_i, \pi_{i+1}, \sigma_1, \ldots, \sigma_{p-1}, \pi_{j-1}, \pi_j$ where σ_s is the contraction of $\pi_{i_{2s}}$ to $\pi_{i_{2s+1}}$.

If n_i is labelled with a variable in w_q for some q then the contraction consists of the four positions $\pi_i, \pi_{i+1}, \pi_{j-1}, \pi_j$ where n_{i+1} is a node of t labelled with a $\lambda\overline{z}$ and π_{j-1} is a child of it; the positions (if they exist) $\pi_{i+2}, \ldots, \pi_{j-2}$ are omitted in the contraction. In contrast, if n_i is labelled with a variable in t then n_{i+1} is a node of w_q for some q labelled with a $\lambda\overline{s}$ and π_{j-1} is a child of it; the contraction not only involves the four positions $\pi_i, \pi_{i+1}, \pi_{j-1}, \pi_j$ but also the contractions of the positions in the branch between $\pi_{i+2}, \ldots, \pi_{j-2}$ if there are such positions. The asymmetry in the definition reflects the difference between the w_i terms that are a fixed part of an interpolation equation and the solution terms t that are open to transformations. Position $(3)[ga]\theta_4$ of the play in Figure 7 on the tree in Figure 6 is a child of $(2)[ga]\theta_2$: the contraction is all positions in this interval $(2)[ga]\theta_2$, $(21)[ga]\theta_3$, $(22)[ga]\theta_3$, $(3)[ga]\theta_4$. Also, $(11)[a]\theta_2$ is a child of $(2)[ga]\theta_2$; now the contraction is $(2)[ga]\theta_2$, $(21)[ga]\theta_3$, σ, $(24)[a]\theta_3$, $(11)[a]\theta_4$ where σ is the contraction of $(22)[ga]\theta_3$ to $(23)[a]\theta_3$. This subsequence is $(22)[ga]\theta_3$, $(3)[ga]\theta_4$, $(8)[a]\theta_6$, $(23)[a]\theta_3$; there is the jump from the position at (3) to its child at (8); those intermediate positions that are at nodes in between are ignored.

Next, we define a notion of prefix of a position.

Definition 42. Assume $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$. We define a (complete) prefix of the position π_j when n_j is in t and labelled with a $\lambda\overline{z}$ for some $\overline{z}$ or when π_j is the last position of π.

1 $n_j = n_2$ is the root of t labelled $\lambda x_1 \ldots x_k$; the (complete) prefix is ε.

2 n_j is in t and is either a successor of a node labelled with a constant or the last position of π; the (complete) prefix is π_{j-1}, π_j.

3 Otherwise node n_{j-1} is in w_q for some q. Let $\pi_{i_1}, \ldots, \pi_{i_{2p+1}}$ be the sequence of branch positions for π_{j-1}. For any $k : 1 \leq k \leq p$, a prefix of π_j is $\pi_{i_{2k}-1}, \pi_{i_{2k}}, \sigma_k, \ldots, \sigma_{p-1}, \pi_{j-1}, \pi_j$ where σ_s is the contraction of $\pi_{i_{2s+1}}$ to $\pi_{i_{2s+2}}$; the complete prefix is when $k = 1$.

Consider the complete prefix for $(9)[ha]\theta_6$ of Figure 9 on the tree in Figure 8. The sequence of branch positions for the previous position $(28)[ha]\theta_{21}$ is in Figure 22. The contractions $\sigma_1, \sigma_2, \sigma_3$ of the positions between $(22), (23)$, $(24), (25)$ and between $(26), (27)$ are

$$\begin{array}{ll} \sigma_1 & (22)[u]\theta_5, (5)[u]\theta_6, [8][u]\theta_6, (23)[u]\theta_{12} \\ \sigma_2 & (24)[u_1]\theta_{12}, (13)[u_1]\theta_{13}, (16)[u_2]\theta_{13}, (25)[u_2]\theta_{12} \\ \sigma_3 & (26)[u_2]\theta_{12}, (5)[u_2]\theta_{12}, (8)[ha]\theta_{14}, (27)[ha]\theta_{21} \end{array}$$

(0)@	$[u]\theta_1$
(21)$\lambda y_1 y_2$	$[u]\theta_5$
(22)y_1	$[u]\theta_5$
(23)λy_3	$[u_1]\theta_{12}$
(24)y_3	$[u_1]\theta_{12}$
(25)λ	$[u_2]\theta_{12}$
(26)y_1	$[u_2]\theta_{12}$
(27)λy_4	$[ha]\theta_{21}$
(28)y_3	$[ha]\theta_{21}$

Fig. 22. The sequence of branch positions for $(28)[ha]\theta_{21}$

So, the complete prefix is $(4)[u]\theta_4, (21)[u]\theta_5, \sigma_1, \sigma_2, \sigma_3, (28)[ha]\theta_{21}, (9)[ha]\theta_6$. A critical point is that the nodes of t labelled with variables that occur in this prefix all belong to the same family.

Definitions 41 and 42 underpin the notion of contraction and (complete) prefix of a stage of a partition.

Definition 43. Assume $\pi_1 = n_1[r_1]\theta_1, \ldots, \pi_l = n_l[r_l]\theta_l$ is a play of $\mathsf{G}(t, E)$ and $\pi_i, \ldots, \pi_j$ is stage N of its partition.

1 If n_i is labelled with a variable then a (complete) prefix of stage N is a (complete) prefix of the parent of π_i.
2 If n_i is labelled with a constant then a (complete) prefix of stage N is ε.
3 The contraction of stage N is defined when n_i is labelled with a constant including @ or when stage N is standard.
   (a) n_i is labelled with a constant. Then the contraction is all the positions $\pi_i, \ldots, \pi_j$.
   (b) Otherwise if $\pi_{i_1}, \ldots, \pi_{i_{2p}}$ is the sequence of branch positions of π_{j-1} to π_{i+1} then the contraction is $\pi_i, \pi_{i+1}, \sigma_1, \ldots, \sigma_{p-1}, \pi_{j-1}, \pi_j$ where σ_s is the contraction of $\pi_{i_{2s}}$ to $\pi_{i_{2s+1}}$.

A prefix of a stage of a partition is either empty when the initial position is at a node labelled with a constant or the prefix of the parent. There is also the notion of a prefix of the last position of a stage. The contraction of a stage is only defined when the node at the initial position is a constant or when the stage is standard (see Definition 40).

Example 17. We examine the complete prefixes and contractions for the stages of the partition in Figure 20 on the tree in Figure 21. The contractions for $S_1 - S_4$ consist of all their moves. The prefixes for $S_2 - S_4$ are each ε because π_2 is the parent of their initial positions. The parent of the initial position of S_5 is $(3)[ga]\theta_4$; therefore, its complete prefix is S_2; its contraction is all its positions (even though it contributes). S_6 starts with $(16)[a]\theta_8$ whose parent is $(7)[ga]\theta_6$ so its prefix is $S3$ and its contraction is all its positions. The complete prefix of S_7 is also S_2 because its initial position is $(6)[a]\theta_4$ whose parent is $(3)[ga]\theta_4$; its contraction is $(6)[a]\theta_4$, $(25)[a]\theta_3$, $(26)[a]\theta_3$, $(3)[a]\theta_{10}$, $(12)[a]\theta_{15}$, $(29)[a]\theta_{16}$, $(30)[a]\theta_{16}$, $(13)[a]\theta_{15}$. S_8 starts with $(14)[a]\theta_{15}$ whose parent is $(7)[a]\theta_{12}$; so its complete prefix is $(4)[a]\theta_{10}$, $(17)[a]\theta_{10}$, $(18)[a]\theta_{10}$, $(7)[a]\theta_{13}$ which corresponds to S_3;

its contraction is all its positions. S_9 starts with $(6)[a]\theta_{10}$ whose parent is $(3)[a]\theta_{10}$; so its prefix is S_2 followed by $(6)[a]\theta_4$, $(25)[a]\theta_3$, $(26)[a]\theta_3$, $(3)[a]\theta_{10}$; and its contraction is all its positions. □

Proposition 34. Assume $\pi_i = n_i[r_i]\theta_i, \ldots, \pi_j = n_j[r_j]\theta_j$ is stage N of the partition of $\pi \in \mathsf{G}(t, E)$. If n_i is labelled with a constant or stage N is standard then the complete prefix of π_j corresponds to the complete prefix of stage N followed by the contraction at stage N.

Proof. Assume $\pi_i = n_i[r_i]\theta_i, \ldots, \pi_j = n_j[r_j]\theta_j$ is stage N of the partition of $\pi \in \mathsf{G}(t, E)$. If n_i is labelled with a constant then $j = i + 1$ and the complete prefix at π_j is π_i, π_{i+1} which is the contraction of stage N as required because the prefix of stage N is ε. Otherwise n_i is labelled with a variable and we assume that stage N is standard. Assume that π_r is the parent of π_i: now, the result follows because π_{r-1} and π_{i+1} must belong to the sequence of branch positions for π_j. □

For instance in Example 17 in the case of S_6 the complete prefix of its last position $(5)[a]\theta_4$ is S_3 followed by its contraction where S_3 is the prefix of its initial position.

6.5. *Unfolding*

In this section we complete the proof of decidability of solvability of interpolation equations which starts with the play partitioning of Section 6.3. We do this by defining a transformation that applies to the individual stages of a partition that we call "unfolding" which is reminiscent of unravelling a model as in modal logic.

As we are going to generalise the notion of a stage of a partition we introduce the following definitions (which uses Definitions 23 and 34).

Definition 44. Assume $\pi_i = n_i[r_i]\theta_i, \ldots, \pi_j = n_j[r_j]\theta_j$ is stage M of the partition of $\pi \in \mathsf{G}(t, E)$.

1 Stage M is local if π_i belongs to the sequence of branch positions for π_j.
2 Stage M is safe for π if
  (a) for all stages $N \geq M$ of the partition of π if N starts from a descendent of a node in $\{n_i, \ldots, n_j\}$ then N does not contribute, and
  (b) for all stages $N' > N \geq M$ of the partition of π if N, N' both start from a descendent of a node in $\{n_i, \ldots, n_j\}$ and end at the same node then there is not a K such that stages $N + K$ and $N' + K$ contribute and for each $I : 1 \leq I < K$, stage $N + I$ corresponds to $N' + I$.

To be local implies that the last position of the stage is at a node below its initial position; by Proposition 32 if stage N of the initial partition of π is not final and starts from a level 1 node or one labelled with a constant or an embedded node and N does not contribute then N is local. A stage of a partition may contribute directly as S_5 does in Figure 20 or indirectly when it finishes at the same node as an earlier stage and then there are corresponding sequences of positions that lead to stages that contribute. A stage is unsafe if it involves a node such that there is a descendent of it and a stage that starts from this descendent which contributes (directly or indirectly).

Definition 45. Assume $\pi_i = n_i[r_i]\theta_i, \ldots, \pi_j = n_j[r_j]\theta_j$ is stage M of the partition of $\pi \in \mathsf{G}(t, E)$. The subtree at stage M is the node n_i together with all its successors $n_i1, \ldots, n_ik$ for $k \geq 0$.

The subtree for S_3 of Figure 20 consists of the nodes (4), (5) and (7) of Figure 21 and for S_7 it is the singleton node (6).

Definition 46. Assume M, N are stages in the partition of $\pi \in \mathsf{G}(t, E)$ and M is safe. We say that M precedes N if the root of the subtree of N is bound within the subtree of M.

S_3 precedes S_8 in Figure 20; however, S_2 does not precede S_5 because S_2 is unsafe.

This leads us to the principal idea of unfolding; assuming that M precedes N and M is local, we expand the starting node and its successors at stage N of the partition with a "prefix" piece of term that is associated with stage M. This is then repeated for further stages.

Definition 47. Assume $N_1, \ldots, N_k$ are stages of the partition of $\pi \in \mathsf{G}(t, E)$ and for each $j : 1 < j \leq k$ there is an $i < j$ such that N_i precedes N_j and the complete prefix at N_j corresponds to a prefix of the complete prefix of the last position of N_{j-1} and stage N_k finishes at a node of a subtree of $N_1, \ldots, N_k$ that is a different node from where each N_i, $1 \leq i < k$, finishes. Then for each $j : 1 < j \leq k$ the unfolding of N_1 into $N_2, \ldots, N_j$ is the the tree $t_j = t_{j-1}[t'/n]$ where n is the node that stage N_{j-1} finishes at and t' is the subtree at N_j. The sequence of positions for the unfolding of N_1 into $N_2, \ldots, N_k$, the new stage N_k, starts from the root of t_k and finishes at a leaf of t_k.

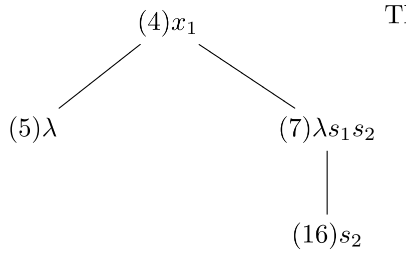
Assume N_1 is unfolded into $N_2, \ldots, N_k$. It follows that the binder of the initial node of N_j, $1 < j \leq k$, occurs above the final position of N_{j-1} (because of the condition that the complete prefix of N_j corresponds to a prefix of the complete prefix of the last position of N_{j-1}). This unfolding is a tree which involves adding the prefix subterms of stages $N_1, \ldots, N_{k-1}$ to N_k. The new stages N_j, $1 \leq j \leq k$, are each local: each such stage corresponds to the concatenation of the contractions of stages $N_1, \ldots, N_j$. There is not a requirement in this definition of unfolding that the tree of N_j occurs below the tree of N_{j-1}; they can belong to different branches but must have a common ancestor.

Proposition 35. Assume $N_1, \ldots, N_k$ are stages of the partition of $\pi \in \mathsf{G}(t, E)$ and N_1 is unfolded int $N_2, \ldots, N_k$.

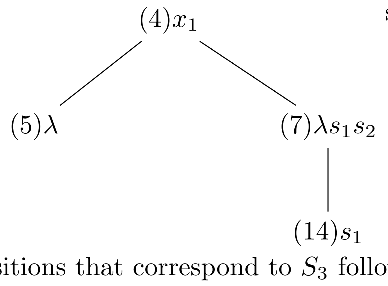
1 Each N_j is safe for π.
2 For each $j \geq 1$, the final position of N_j is at a node in a subtree at N_1 or N_2 or $\ldots$ or N_j.
3 The sequence of positions for the unfolding of N_1 into $N_2, \ldots, N_k$ corresponds to the concatenation of contractions of $N_1, \ldots, N_k$.

Proof. 1 is clear by definition. 2 follows by definition for N_k and from the two requirements that the complete prefix at stage N_j corresponds to a prefix of the complete prefix of the last position of stage N_{j-1} (so, the binder for N_j is at or above this position) and that for some $i < j$, N_i precedes N_j. 3 By a simple induction on j , it follows that the sequence of moves on t_j corrresponds to the concatenation of the contractions of $N_1, \dots, N_j$; by definition stage N_k finishes at a different node than the previous stages, which means that the sequence of positions on t_k finishes at one of its leaves. □

Consider the partition in Figure 20 on the tree t in Figure 21 without stage S_4 and where (12) is directly below (7) because the end sequence $(8), (11)$ is redundant as described in Section 6.2. As S_5 contributes, S_2 fails to be safe. S_3 precedes S_6, the complete prefix at S_6 is the complete prefix of the last position of S_3 and S_6 ends at a node of the tree of S_3; therefore, the unfolding of S_3 into S_6 is the following subtree.



The new stage S_6 is local, takes place within this subtree (and the subtree rooted at (17)) and consists of a sequence of positions that correspond first to S_3 and then to the contraction of S_6 finishing at its leaf node (5). (The nodes at this new stage would then be renamed; but we leave their old names in to show where they come from.) Stage S_7 is unchanged except for where it starts from; it starts at node (6) below (5) of the new subtree at S_6 and still ends at node (13); S_2 is not unfolded into S_7 because it is unsafe. Consider next S_8; S_3 precedes S_8, the complete prefix at S_8 corresponds to the complete prefix of the final position of S_3 and S_8's final position is at a node in the subtree of S_3. So, S_3 can be unfolded into S_8 which means that node (14) is replaced with the subtree:



so, the new S_8 is local and consists of a sequence of positions that correspond to S_3 followed by the contraction of S_8; it finishes at its new node (5). S_9 is unchanged except where it starts from; it starts at node (6) directly beneath (5) at S_8. Although the initial tree t has now grown because of the unfoldings, bindings are more local; so, the first occurrence of node (7) no longer binds (14) and (16). Now we can apply transformation T_2 of Section 6.2 to the expanded tree; first to the end sequence $(4), (7)$ (for (4) below (3)) because this (7) no longer binds; second to $(4), (5)$ of the subtree at S_6 that replaced node (16); third

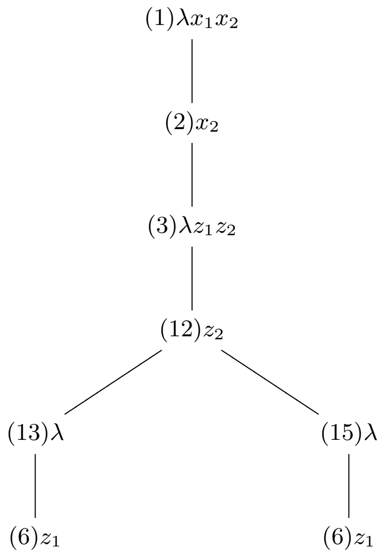


Fig. 23. Reduced solution term for Example 16

to $(4), (5)$ of the subtree at S_8 that replaces node (14). Therefore, the resulting term, depicted in Figure 23, is a "small" solution to the interpolation equation.

Unfolding is relative to a particular play and to its partition; therefore, it is not a transformation in the sense of T_1 and T_2 of Section 6.2.

The central theorem is a small solution property as follows (see Definitions 2 and 8 for arity, depth means the length of a longest branch in the tree representation of a term and $|t|$ is the size, the number of nodes, in the tree representation of t). If the interpolation problem E has a (canonical) solution then it has a *small* solution that is bounded in the size of the problem: the parameter n is the highest level of a node in a potential solution term, Definition 32. Therefore, there is a tree automaton such that A_E is non-empty iff E has a (canonical) solution.

Theorem 36. If E, $xw_1 \ldots w_k = u$, has order $2n$ or $2n+1$ and arity m then it has a (canonical) solution iff it has a (canonical) solution whose depth is at most $4(n^2 m^{2n-2} \times |u|)$.

Proof. Assume E, $xw_1 \ldots w_k = u$, has order $2n$ or $2n+1$ and arity m. Let t be a smallest (canonical) solution. We now examine the game $\mathsf{G}(t, E)$. The number of plays is branch(u) which is bounded by $|u|$. Term t can contain nodes whose level is at most n. In any full branch of t, the number of occurrences of constants is bounded in terms of depth(u); otherwise transformation T_1 of Section 6.2 would apply and so reduce the size of t. Also T_2 does not apply, as it would also reduce the size of t. Consider the partition of each play $\pi \in \mathsf{G}(t, E)$. They all start with the same initial stage and start from the same node at stage 2. We build an auxiliary structure, the partition tree; the nodes of this tree are themselves trees (the subtrees at each stage). The tree for stage N of a play π is the node of the initial position at this stage plus, if they exist, all its successors, Definition 45; if N is not final for π then there is also a directed edge labelled π from stage $N+1$ of π in this tree to a node of a previous stage of π where stage N finishes. The term t can be recovered from this tree by viewing directed edges as successor relations. A stage N is a separator if there are two different plays that share the same stages before N and start from the same node at stage N but finish at different nodes (or for one of the plays stage N is final). If N is local in π and not final then stage $N+1$ has a π edge into N. If M precedes N then there is a path of π edges from N into M. We do the unfolding on the partition tree; N_1 is unfolded into $N_2, \ldots, N_k$ only when the sequence of positions for this unfolding corresponds to the concatenation of the contractions of $N_1, \ldots, N_k$, see Proposition 35; the subtree N_k is replaced by this unfolding (the tree t_k of Definition 47). The new stage N_k is local; the π edge from N_{k+1} into some N_i will then change to into N_i's node in N_k; this change may cause further changes to π edges; if there is a path of π edges from a later stage M to N_k and the π edge from stage $M+1$ is into some N_i then it is changed into N_i's node in N_k. Clearly, if stage N of π contributes then it still contributes after all unfoldings; and no new unfolded stage can contribute by construction. If stage N of π is not changed by unfolding then it retains the same contraction as before the unfolding.

If the number of level $j-1$ nodes is k in a branch of t and the number of level j nodes is $mk+kmq$ in that branch then at least kmq of these nodes are embedded and at least q are labelled with the same variable and bound by the same binder: if $q \geq mr$ then at least r stages which start from such nodes can be unfolded into a later stage provided they are safe. The number of stages that can directly or indirectly contribute is at most $|u|$ and therefore the number of later stages that start from a node that is a descendent of a starting node of such a contributing stage and that is not embedded is nm^{n-1}. If the only reason that N_1 cannot be unfolded into $N_2, \ldots, N_k$ is that N_k finishes at the same node as some N_i, $i < k$, then by the uniformity properties it follows that there must be later stages $N_{k+1}, \ldots, N_{k+m}$ such that N_1 can be unfolded into $N_2, \ldots, N_k, \ldots, N_{k+m}$ and that the subtree $t_{k+m} = t_k$ of Definition 47. Once a maximum amount of unfolding has taken place, we apply transformations T_1 and T_2 of Section 6.2 to reduce the size of the unfolded term. The maximum depth of an unfolding which does not contain redundant embedded subtrees is is $2nm^{n-1}$ (when a level 1 node is unfolded into m level 2 nodes and together they are unfolded into m^2 level 3 nodes and so on). The question is how many overall stages will T_2 not be applicable to? These include the stages that contribute and those later stages that start from descendent nodes that are not embedded. There is also the issue that in the tree representation of the stages if the same node has more than one edge entering it from two different later stages of different plays then they need to be the same subterm; to guarantee this we may need extra stages that are also play separators; the number of such separators is bounded by $(nm^{n-1} \times |u|)$ which is at most the number of stages that either contribute or start from a descendent that is not embedded. This therefore gives a depth bound of $4(n^2m^{2n-2} \times |u|)$. $\square$

The bound in Theorem 36 is very crude. For instance, for Example 6.2 whose order is 5, arity is 2 and where the size of the goal term is also 2, the bound on the depth of a smallest solution term is 128!

7. Conclusion

We have described a proof of decidability of higher-order matching that is simpler than that presented in (Stirling 2009A) mainly because it uses a more symmetric game-theoretic characterisation of solutions of interpolation equations; the notions of uniformity properties of game playing, partitioning plays and unfolding are common to both proofs. We also described how tree automata provide a different characterisation of solutions to interpolation equations. It is an open question whether such a characterisation can also lead to decidability. There is a third characterisation that uses type checking; the types are intersection types generated by subterms of u, based on (Kobayashi and Ong 2009); in this way one can also reduce the interpolation problem, whether it has a (canonical) solution, to the (undecidable) type inhabitation problem (Urzyczyn 1999), whether there is a term that has a given intersection type.

In the presentation of matching here we made two simplifications. First, we assume that there is only a single base type. Allowing multiple base types is straightforward. Their presence merely adds extra constraints on well-formedness of terms. For instance, if we start with an interpolation equation $xw_1 \dots w_k = u$ with $x : (A_1, \dots, A_k, \alpha)$ where α is a base type then a potential solution term in normal form is $\lambda x_1 \dots x_k.s$ where the head of s must be a variable x_i or a constant with type α or $(B_1, \dots, B_p, \alpha)$; in the former case this means that w_i is also constrained. The second simplification is that we assume that the goal term u of an interpolation equation does not contain bound variables. Assume u is well named and contains the bound variables $\{z_1, \dots, z_n\}$. To deal with this, we assume a new family of constants $\{c_1, \dots, c_n\}$, the "forbidden" constants, that cannot occur in potential solution terms, where each c_i has the same type as z_i; these constants will then replace bound variables in subterms of states and can also be associated with variables in look-up tables; see (Stirling 2007; Stirling 2009A) for the details.

It still remains to be seen whether there is a simpler proof of decidability; for instance, although we tried (Stirling 2005), one possibility is a proof using further simple transformations on terms such as T_1 and T_2 of Section 6.2.

There is an intimate relationship between observational equivalence of terms and families of interpolation equations (Padovani 1995; Padovani 2000). We have started to examine whether there is a simple characterisation of observational equivalence using comparison games on terms (similar in spirit to bisimulation games).

References

Alur, R. and Madhusudan P. Adding nested structure to words *Lecture Notes in Computer Science*, 4036, 1-13, (2006).

Alur, R., Chaudhuri S. and Madhusudan, P. Languages of nested trees, *Lecture Notes in Computer Science*, 4144, 329-342, (2006).

Barendregt, H. *The Lambda Calculus Its Syntax and Semantics*, Studies in Logic and the Foundations of Mathematics, Vol 103, North-Holland, (1984).

Barendregt, H. Lambda calculi with types. In *Handbook of Logic in Computer Science, Vol 2*, ed. Abramsky, S., Gabbay, D. and Maibaum, T., Oxford University Press, 118-309, (1992).

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).

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

Dougherty, D. and Wierzbicki, T. A decidable variant of higher order matching. *Lecture Notes in Computer Science*, 2378, 340-351, (2002).

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

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).

Goldfarb, W. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13, 225-230, (1982).

Hindley, J. and Seldin, J. *Introduction to Combinators and λ-Calculus*, London Mathematical Society Student Texts 1, Cambridge University Press, (1986).

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

Joly, T. The finitely generated types of the lambda calculus. *Lecture Notes in Computer Science*, 2044, 240-252, (2001).

Joly, T. On λ-definability I: the fixed model problem and generalizations of the matching problem. *Fundamenta Informaticae*, 65, 135-151, (2005).

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

Kobayashi, N. and Ong, C.-H. L. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. *Procs. LICS 2009*, 179-188, (2009).

Loader, R. Unary PCF is decidable. *Theoretical Computer Science*, 206, 317-329, (1998).

Loader, R. Undecidability of λ-definability

Loader, R. Finitary PCF is not decidable. *Theoretical Computer Science*, 266, 342-362, (2001).

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

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, 55 pages preprint, 2006.)

Ong and Tzevelekos. Functional Reachability. *Procs LICS 2009*, 286-295, (2009).

Padovani, V. On equivalence classes of interpolation equations. *Lecture Notes in Computer Science*, 902, 335-349, (1995).

Padovani, V. Decidability of all minimal models. *Lecture Notes in Computer Science*, 1158, 201-215, (1996).

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

Schmidt-Schauβ, M. Decidability of arity-bounded higher-order matching. *Lecture Notes in Artificial Intelligence*, 2741, 488-502, (2003)

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

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

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

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

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

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

Stirling, C. Higher-order matching, games and automata. *Procs LICS 2007*, 326-335, (2007).

Stirling, C. Dependency tree automata. *Lecture Notes in Computer Science*, 5504, 92-106, (2009).

Stirling, C. Decidability of higher-order matching *Logical Methods in Computer Science*, 5(3:2), 1-52, (2009).

Wierzbicki, T. Complexity of higher-order matching. *Lecture Notes in Computer Science*, 1632, 82-96, (1999).

Støvring, K. Higher-order beta matching with solutions in long beta-eta normal form. *Nordic Journal of Computing*, 13, 117-126, (2006).

Urzyczyn, P. The emptiness problem for intersection types. *Journal of Symbolic logic*, 64(3), 1195-1215, (1999).