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 August 2012

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 the free variables in t be instantiated to produce a term that matches 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; Vorobyov 1997; 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-Schauß 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 a 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 terms w_j 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 an example 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. 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 as we discuss further in Section 6.5.

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 $A \rightarrow B \rightarrow C$ is $A \rightarrow (B \rightarrow C)$, 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 $A_1, \dots, A_n$. 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 $A_1, \dots, A_n$.

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 $x^A, y^B, \ldots$ and *typed* constants $a^{\mathbf{0}}, f^A, \ldots$ (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 $\lambda x.t : A \to B \in T$,
4. if $t : A \to B \in T$ and $u : A \in T$ then $(tu) : B \in T$.

We let $s, t, \ldots$ range over terms in T. Function abstraction is given by lambda abstraction: $\lambda x.t$ has type $A \to B$, if $x : A$ and $t : B$. Function application is written using juxtaposition; $t : A \to B$ 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 $tu_1 \ldots u_k$ is $((\ldots(tu_1)\ldots)u_k)$.

Usual definitions of when a variable occurrence is free or bound and when a term is closed are assumed: for instance, both occurrences of x in $xy(\lambda u.xzu)$ in the term $\lambda x.\lambda y.xy(\lambda u.xzu)$ are bound (by λx) whereas 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 $f : \mathbf{0} \to (\mathbf{0} \to \mathbf{0})$, so $(fa) : \mathbf{0} \to \mathbf{0}$ 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 $\lambda z_1 \ldots z_m$ for $\lambda z_1 \ldots \lambda z_m$; 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	$(\lambda x.t)u \to_\beta t\{u/x\}$	
η-reduction	$\lambda x.tx \to_\eta t$	if x is not free in t

Here $\{u_1/x_1\}$ and, more generally, $\{u_1/x_1, \ldots, u_n/x_n\}$ is (simultaneous) substitution where each x_i is 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 $\lambda z.faz \rightarrow_\eta fa$. 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, $\rightarrow_{\beta\eta}$ is $\rightarrow_\beta \cup \rightarrow_\eta$ and $\rightarrow^*_\delta$ is the reflexive and transitive closure of $\rightarrow_\delta$.

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

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

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 $t \rightarrow^*_\delta s$ and $t' \rightarrow^*_\delta s'$ 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 $t \rightarrow_\beta t'$.
2. Term t in β-normal form is *η-long* if there is not a t' such that $t' \rightarrow_\eta t$.

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

Fact 2. If $t : A \rightarrow B$ and $u : A$ are both in lnf, and $tu \rightarrow^*_\beta t'$ 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

— $ut_1 \ldots t_m$, $m \geq 0$ 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. $\square$

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 $x : (((\mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$, each $z_i : (\mathbf{0}, \mathbf{0})$ and $y : \mathbf{0}$. $\square$

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 equation* has the form $v = u$ where $v, u : \mathbf{0}$ are in lnf and u is closed. The *order* of the equation 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 equation is the largest arity of the types A_i such that $x_i : A_i$.

The *matching problem* is: given a matching equation $v = u$, does it have 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 equation is similar $v = u$ 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 $v\{t_1/x_1, \ldots, t_n/x_n\} =_{\beta\eta} u\{t_1/x_1, \ldots, t_n/x_n\}$. A β-matching equation is $v = u$ 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 $v\{t_1/x_1, \ldots, t_n/x_n\} =_{\beta} u$. The unification or β-matching problem is: given a unification or β-matching equation, does it have a solution? Unification is undecidable, even at order 2 (Goldfarb 1982) and β-matching 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 matching. First is the general case of an equation $v = u$ when $v, u : A$ for arbitrary A and u is closed. If $A = (A_1, \ldots, A_k, \mathbf{0})$ then the term u has the form $\lambda z_1 \ldots z_k.u'$; therefore, there is the matching equation $v' = u''$ when v', u'' are the lnfs of $vc_1 \ldots c_k$ and $u'\{c_1/z_1, \ldots, c_k/z_k\}$, and each c_i is a new constant of correct type that cannot occur in a solution term. Statman presents "the range question" (Statman 1982): given closed $v : (A_1, \ldots, A_n, B)$ and $u : B$ are there terms $t_1 : A_1, \ldots, t_n : A_n$ such that $vt_1 \ldots t_n =_{\beta\eta} u$? This is equivalent to the matching problem $v' = u$ where v' is the lnf of $vx_1 \ldots x_n$ where each x_i has type A_i. In (Padovani 2000) a matching problem is a family of matching equations $v_1 = u_1, \ldots, v_m = u_m$ to be solved uniformly, as all occurrences of a free x_i in each equation must be instantiated with the same t_i: this problem reduces to the single matching equation $fv_1 \ldots v_m = fu_1 \ldots u_m$ 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 $x : (A_1, \ldots, A_k, \mathbf{0})$. An *interpolation equation* is $xw_1 \ldots w_k = u$. The *type* and *order* of the equation is that of x. A *solution* is a closed term $t : (A_1, \ldots, A_k, \mathbf{0})$ in lnf such that $tw_1 \ldots w_k =_{\beta\eta} u$. The term u of the equation $xw_1 \ldots w_k = u$ is called its *goal term.*

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

Proof. If t solves $xw_1 \ldots w_k = u$ then $tw_1 \ldots w_k =_{\beta\eta} u$ and the result therefore follows from Fact 2 because the terms $t, w_1, \ldots, w_k, u$ 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 $xw_1 \ldots w_k \neq u$ and is solved by t of correct type and in lnf if $tw_1 \ldots w_n \neq_{\beta\eta} u$. 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 equation with free variables $x_1 : A_1, \ldots, x_n : A_n$ then its *associated interpolation equation* is $x(\lambda x_1 \ldots x_n.v) = u$ where $x : ((A_1, \ldots, A_n, \mathbf{0}), \mathbf{0})$. This appears to raise order by 2 as with 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) $\lambda z.zt_1 \ldots t_n$ 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 equation has a solution iff its associated interpolation equation has a canonical solution.

Proof. Assume $v = u$ is a matching equation, $x_1 : A_1, \ldots, x_n : A_n$ are the free variables that occur in v and $x(\lambda x_1 \ldots x_n.v) = u$ is its associated interpolation equation. If the matching equation 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 equation $v = u$. $\square$

Similarly, a β-matching equation $v = u$ also has an associated β-interpolation equation $x(\lambda x_1 \ldots x_n.v) = u$; and the equation 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 equation $x_1(\lambda z.x_1(\lambda z'.za)) = a$ 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 equation 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 $t = \lambda x.x(\lambda y.y(\lambda y_1 \ldots y(\lambda y_k.s)...))$ where s is the constant a or one of the variables y_j, $1 \leq j \leq k$. To see this assume $w_1 = \lambda y.y(\lambda y_1 \ldots y(\lambda y_k.s)...)$: 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 $y(\lambda y_j.s''')$. $\square$

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 equation, 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 $t(\lambda y_1y_2.y_2)a \rightarrow_\beta (\lambda y_1y_2.y_2)ba \rightarrow^*_\beta a$. $\square$

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.

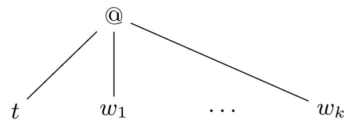

Fig. 1. An interpolation tree

In general the goal u of an interpolation equation may contain bound variables: for instance, $x(\lambda z.z) = h(\lambda x_1 x_2 x_3.x_1 x_3)a$ 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 $\lambda y.y(y(h(\lambda x z_1 z_2.x(y z_2))(ya)))$. For convenience, as it simplifies the presentation considerably, we restrict ourselves to the case where the goal u does not contain bound variables; the extension of what is to follow to include bound variables is discussed in Section 6.5. As u does not contain bound variables, Proposition 4 can be slightly strengthened: if t solves $xw_1 \dots w_k = u$ then $tw_1 \dots w_k \to_\beta^* u$.

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

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

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 $@tw_1 \dots w_k \to_\beta^* tw_1 \dots w_k$.

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.)

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 $m \geq 0$, 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 odd depth is labelled with a $\lambda\overline{y}$ and a node that is at even 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 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 odd depth are labelled with a lambda. $\square$

We assume standard terminology of finite trees. We let n, m range over nodes of a tree: in the case of a term tree nodes are labelled with variables, constants or lambdas; so, we say that a node is a variable node if it is labelled with a variable and similarly for a constant and a lambda node (which may also be a binding node). More generally, if Σ is an alphabet then a Σ-tree is a finite tree whose nodes are labelled with elements of Σ (see Definition 13). Nodes have successors: often we use the notation that ni is the ith successor of n. We say that a node n occurs *above* m if $n = m$ or there is a path from n down to m in the tree; m occurs *below* n if there is a path from m up to n in the tree.

The following is a formal definition of a binding tree.

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 $t_i w_1 \dots w_k \to_\beta^* u$; 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 al 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 al 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 $n1, \dots, nk$ 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 $sq_1 \dots q_k \Rightarrow q$ where $k \geq 0$, $s \in \Sigma$, $\mathrm{ar}(s) = k$ and $q_1, \dots, q_k, q \in Q$.

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 $s \in \Sigma$, $\mathrm{ar}(s) = k \geq 0$, $sq_1 \dots q_k \Rightarrow q \in \Delta$ and $q_1, \dots, q_k$ label $n1, \dots, nk$ 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 a Σ-tree t is a labelling of t 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 $n1, \dots, nk$ are labelled with the states $q_1, \dots, q_k$ and $sq_1 \dots q_k \Rightarrow q \in \Delta$ 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 al 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

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

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 belong to $\{x_1, \dots, x_k\}$. 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: $tw_1w_2 \to_\beta^* w_1(w_2(w_1(w_1ab)b))b \to_\beta^* w_1(w_2a)b \to_\beta w_1(faa)b \to_\beta faa$. $\square$

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

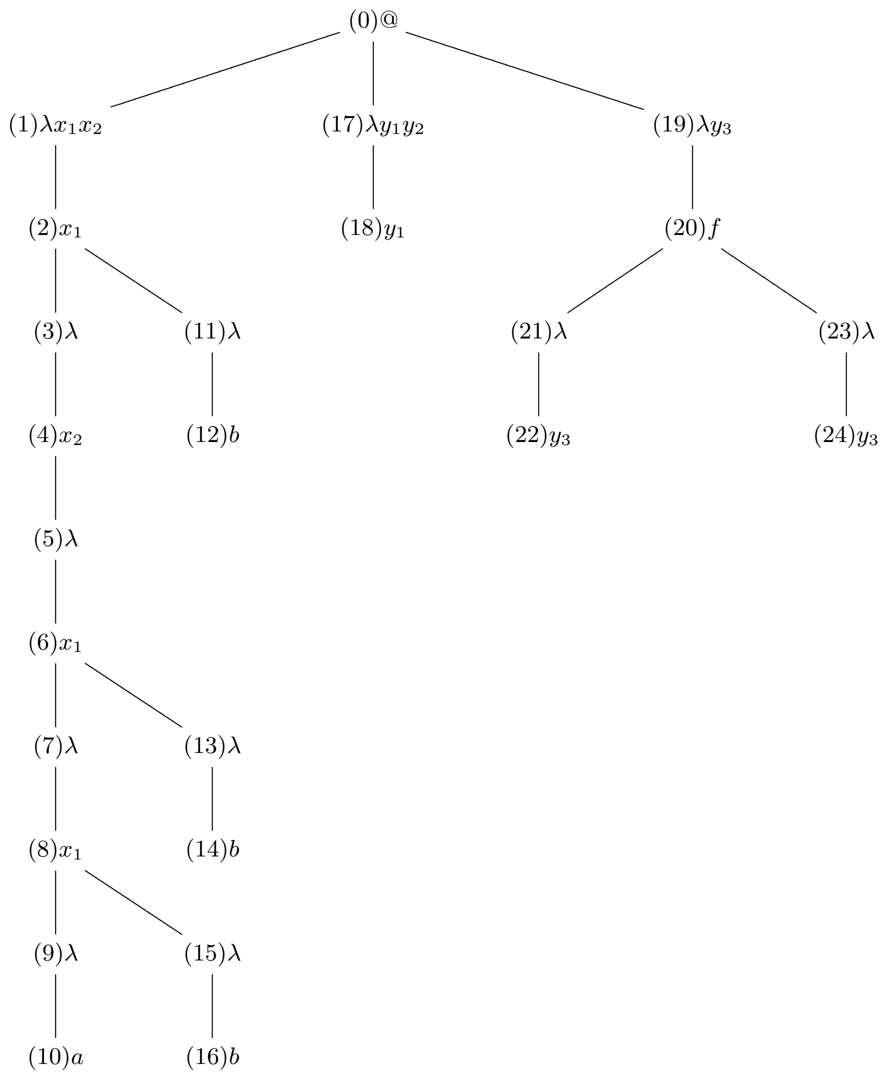

Fig. 2. A 3rd-order example

Jurski 1997), the normal form of $t'\{w_1/x_1, \ldots, w_k/x_k\}$ when t' is the subterm rooted at n; this normal form must be a subterm of u, an element of Sub(u). For instance, nodes (12), (14) and (16) of Figure 2 are inaccessible. Node (6) is accessible; its subterm is $x_1(x_1ab)b$ whose evaluation is the normal form of $x_1(x_1ab)b\{\lambda y_1y_2.y_1/x_1, \lambda y_3.fy_3y_3/x_2\}$ 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 $\lambda x_1 \ldots x_k$; such a node n is inaccessible if its 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 Sub(u) where u is the goal term and one other state for inaccessible nodes plus their versions for lambda nodes.

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

The states $[-]$ and $[\lambda.-]$, 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 $[\lambda x_1 \ldots x_k.u]$ is the final state of the automaton for the root node.

Next we define 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 $[\lambda x_1 \ldots x_k.u]$. Transitions are of the form $sq_1 \ldots q_k \Rightarrow q$ where $s \in \Sigma$ has arity k and $q_1, \ldots, q_k, q$ are states. First when s is a constant or a λ there are the following transitions.

$$\begin{array}{ll}
f[\lambda.u_1] \ldots [\lambda.u_m] \Rightarrow [fu_1 \ldots u_m] & \quad m \geq 0 \\
f[\lambda.-] \ldots [\lambda.-] \Rightarrow [-] & \\
\lambda[u'] \Rightarrow [\lambda.u'] & \\
\lambda[-] \Rightarrow [\lambda.-] & \\
\lambda x_1 \ldots x_k[u] \Rightarrow [\lambda x_1 \ldots x_k.u] &
\end{array}$$

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 $x_i \in \{x_1, \ldots, x_k\}$. If n is accessible then its evaluation employs w_i for x_i; therefore, we are interested in the normal form of $w_ie_1 \ldots e_m$ when for each j state $[\lambda.e_j]$ labels nj; $e_j \in \mathrm{Sub}(u) \cup \{-\}$; so, if e_j is $-$ then it does not contribute to the normal form.

$$\begin{array}{ll}
x_i[\lambda.e_1] \ldots [\lambda.e_m] \Rightarrow [u'] & \quad m \geq 0 \text{ and } w_ie_1 \ldots e_m \to_\beta^* u' \in \mathrm{Sub}(u) \\
x_i[\lambda.-] \ldots [\lambda.-] \Rightarrow [-] &
\end{array}$$

Example 9. The automaton for the equation in Example 8 has, as we have seen, alphabet $\Sigma = \{a, b, f, x_1, x_2, \lambda, \lambda x_1x_2\}$. Q is $\{[faa], [\lambda.faa], [a], [\lambda.a], [-], [\lambda.-], [\lambda x_1x_2.faa]\}$ and its last element is the final state. Transitions are as follows where e ranges over $\{a, faa, -\}$.

$$\begin{array}{ll}
a \Rightarrow [a] & a \Rightarrow [-] \quad b \Rightarrow [-] \\
f[\lambda.a][\lambda.a] \Rightarrow [faa] & f[\lambda.-][\lambda.-] \Rightarrow [-] \\
x_1[\lambda.e][\lambda.-] \Rightarrow [e] & \\
x_2[\lambda.a] \Rightarrow [faa] & x_2[\lambda.-] \Rightarrow [-] \\
\lambda[e] \Rightarrow [\lambda.e] & \\
\lambda x_1x_2[faa] \Rightarrow [\lambda x_1x_2.faa] &
\end{array}$$

In the rule for x_1, the second argument of $w_1e_1e_2$ does not contribute to normal form because w_1 is $\lambda y_1y_2.y_1$. 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. □

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.

$$\begin{aligned}
s &::= z_j \mid fs_1 \dots s_m \mid x_iv_1 \dots v_m \\
v &::= s \mid \lambda z_1 \dots z_m.s
\end{aligned}$$

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 each z_j is of ground type. Therefore, with this grammar we can write 4th-order terms which use arbitrarily many variables such as

$$\lambda x_1x_2.x_1(\lambda z_1.x_1(\lambda z_2.x_1 \dots x_1(\lambda z_n.x_2(x_2 \dots (x_2(x_2z_1z_2)z_3) \dots)z_n) \dots))$$

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 z_j of ground type: therefore, its evaluation 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 variables z_j. 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 variables z_j 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.

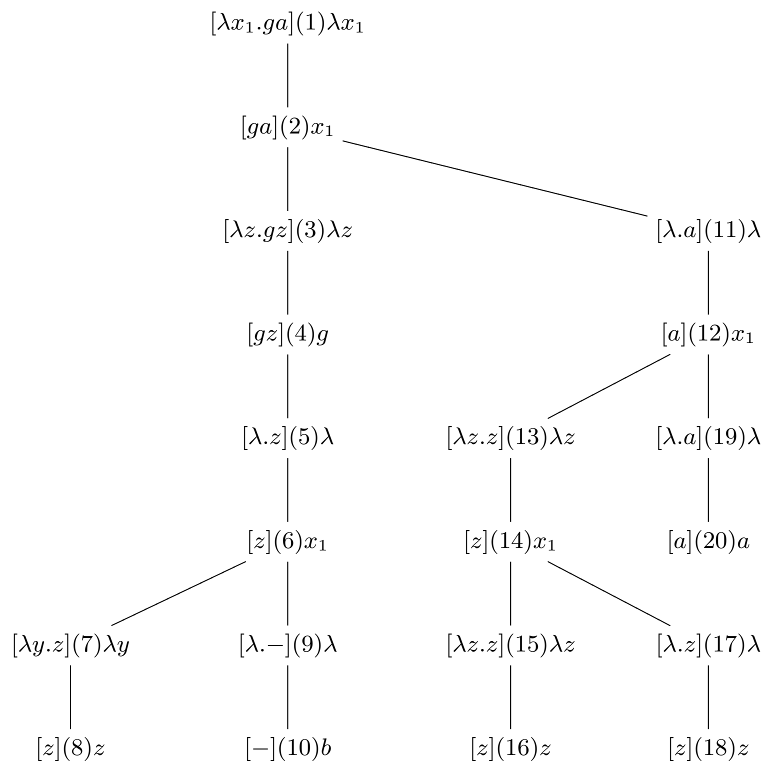

Fig. 3. A successful run of the automaton of Example 10

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

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

Below are the transitions where $e' \in \{-, z, a, gz, ga\}$.

$a \Rightarrow [a]$	$a \Rightarrow [-] \qquad b \Rightarrow [-]$
$z \Rightarrow [z]$	$z \Rightarrow [-]$
$g[\lambda.e] \Rightarrow [ge]$ for $e \in \{a, z\}$	$\lambda[e'] \Rightarrow [\lambda.e']$
$\lambda y[e] \Rightarrow [\lambda y.e]$ for $e \in \{a, z, ga, gz\}$	$\lambda z[e] \Rightarrow [\lambda z.e]$ for $e \in \{a, z, ga, gz\}$
$\lambda x_1[ga] \Rightarrow [\lambda x_1.ga]$	$x_1[\lambda y.e][\lambda.e'] \Rightarrow [e]$ for $e \in \{a, z, ga, gz\}$
$x_1[\lambda z.e][\lambda.e'] \Rightarrow [e]$ for $e \in \{a, ga\}$	$x_1[\lambda z.z][\lambda.e] \Rightarrow [e]$ for $e \in \{a, z, ga, gz\}$
$x_1[\lambda z.gz][\lambda.e] \Rightarrow [ge]$ for $e \in \{z, a\}$	

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 $\lambda x_1.x_1(\lambda z.g(x_1(\lambda y.z)b))(x_1(\lambda z.x_1(\lambda z.z)z)a)$. 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 $[\lambda z.z]$ 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. □

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 variables z_i 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 variables z_j we may still end up with arbitrary many evaluations: for instance, assume that they are all z_n and consider the evaluation at the position of the first z_n which is $z_n(z_n(\ldots(z_n a)\ldots))$.

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 $\lambda xy.x(\lambda z_1.x(\lambda z_2 \ldots (\lambda z_n.z_n(z_{n-1}(\ldots z_1(y))\ldots))\ldots))$ 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

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

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.

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 $a : \mathbf{0}$ the play ends. At other constants f, player Proponent, $\exists$, matches 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).

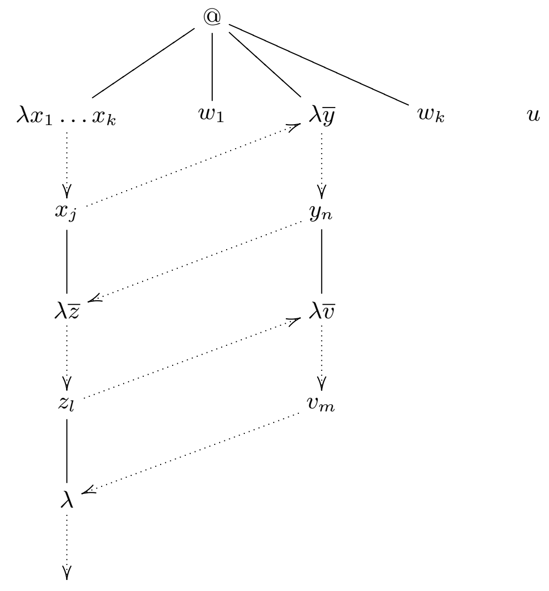

Fig. 4. Game-theoretic view

4.1. *Interpolation games*

Assume E is the equation $xw_1 \ldots w_k = u$ 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 $@tw_1 \ldots w_k$ of Figure 1. In the following we use the interpolation tree notation that was introduced in Section 3.1.

Definition 16. Let N be the set of nodes of the interpolation tree $@\,tw_1 \ldots w_k$ and $N_1 \subset N$ be the binding nodes (labelled with binders in Σ_1). S is the set $\{[\,r\,] \,|\, r \in \mathrm{Sub}(u) \cup \{\forall, \exists\}\}$ of *game states.* $[\,\forall\,]$ and $[\,\exists\,]$ are the *final* game states.

Definition 17. For each $i \geq 1$, the *set of look-up tables* Θ_i is iteratively defined:

1. $\Theta_1 = \{\theta_1\}$ where $\theta_1 = \emptyset$,
2. Θ_{i+1} is the set of partial maps $N_1 \to (\bigcup_{\lambda y_1 \ldots y_p \in \Sigma_1} N^p \times \bigcup_{j \leq i} \Theta_j)$.

Definition 18. A *play* of $\mathsf{G}(t, E)$ is a finite sequence $n_1 q_1 \theta_1, \ldots, n_l q_l \theta_l$ 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 $[\,r\,]$, $r \in \mathrm{Sub}(u)$ 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,

Current position is $n[r]\theta$. The next position is by cases on the label at node n:
— @ then $n1[r]\theta'$ where $\theta' = \theta\{((n2, \ldots, n(k+1)), \theta)/n1\}$
— $\lambda\overline{y}$ then $n1[r]\theta$
— $a : \mathbf{0}$ if $r = a$ then $n[\exists]\theta$ else $n[\forall]\theta$
— $f : (B_1, \ldots, B_p, \mathbf{0})$ if $r = fr_1 \ldots r_p$ then $\forall$ chooses $j \in \{1, \ldots, p\}$ and $nj\,[r_j]\theta$ else $n[\forall]\theta$
— $y_j : \mathbf{0}$ if $m \downarrow n$ and $\theta(m) = ((m_1, \ldots, m_l), \theta')$ then $m_j\,[r]\theta'$
— $y_j : (B_1, \ldots, B_p, \mathbf{0})$ if $m \downarrow n$ and $\theta(m) = ((m_1, \ldots, m_l), \theta')$ then $m_j\,[r]\theta''$ where $\theta'' = \theta'\{((n1, \ldots, np), \theta)/m_j\}$

Fig. 5. Game moves

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 $\lambda y_1 \ldots y_p$ and $\theta_k(n)$ is defined then it has the form $((n_1, \ldots, n_p), \theta_j)$ 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.

Definition 19. 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 (and where ni is the ith successor node of n).

At the initial node labelled @, play proceeds to its first successor labelled with $\lambda x_1 \ldots x_k$ and the look-up table is updated as this binding node is associated with the other successors of the root; so any free variable x_j in the subtree below $\lambda x_1 \ldots x_k$ 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: $\gamma\{((m1, \ldots, ml), \gamma')/n\}$ is the partial function similar to γ except that $\gamma(n) = ((m1, \ldots, ml), \gamma')$ where n is labelled $\lambda z_1 \ldots z_l$ for some z. If play is at a lambda node 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 with arity more than 0, $\forall$ immediately wins if the state is not of the form $[fr_1 \ldots r_m]$. 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 $\theta(m) = ((m_1, \ldots, m_l), \theta')$ when $m \downarrow n$ then play jumps to the lambda node m_j which has type $\mathbf{0}$ (so, is labelled with a dummy lambda) and θ' is then the look-up table; θ' consists of entries for the binders that occur above m_j in the interpolation tree (for interpreting the free variables in the subtree at m_j). If n is labelled $y_j : (B_1, \ldots, B_p, \mathbf{0})$ and $\theta(m) = ((m_1, \ldots, m_l), \theta')$ when $m \downarrow n$ then play jumps to the lambda node m_j which is labelled $\lambda z_1 \ldots z_p$ for some z (because m_j has

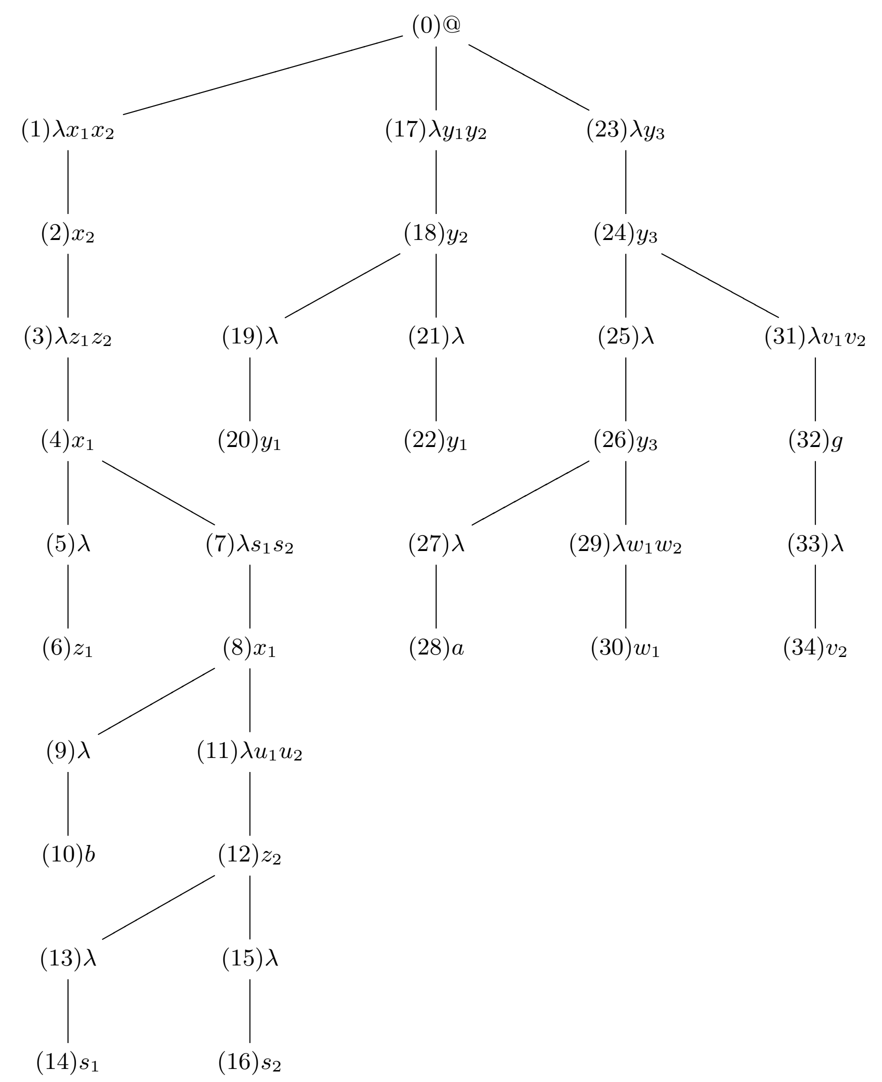

Fig. 6. The interpolation tree of Example 12

the same type as y_j): the new look-up table is θ' plus the entry $((n1, \ldots, np), \theta)$ for the binder m_j; each z_i bound by m_j is associated with the ith successor of n and θ for each $i : 1 \leq i \leq p$.

Definition 20. 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).

Example 12. 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 $g : (\mathbf{0}, \mathbf{0})$.

1.	$(0)[ga]\theta_1 = \emptyset$	25.	$(6)[a]\theta_4$
2.	$(1)[ga]\theta_2 = \{((17,23),\theta_1)/1\}$	26.	$(25)[a]\theta_3$
3.	$(2)[ga]\theta_2$	27.	$(26)[a]\theta_3$
4.	$(23)[ga]\theta_3 = \{((3),\theta_2)/23\}$	28.	$(3)[a]\theta_{10} = \theta_2\{((27,29),\theta_3)/3\}$
5.	$(24)[ga]\theta_3$	29.	$(4)[a]\theta_{10}$
6.	$(3)[ga]\theta_4 = \theta_2\{((25,31),\theta_3)/3\}$	30.	$(17)[a]\theta_{11} = \{((5,7),\theta_{10})/17\}$
7.	$(4)[ga]\theta_4$	31.	$(18)[a]\theta_{11}$
8.	$(17)[ga]\theta_5 = \{((5,7),\theta_4)/17\}$	32.	$(7)[a]\theta_{13} = \theta_{10}\{((19,21),\theta_{11})/7\}$
9.	$(18)[ga]\theta_5$	33.	$(8)[a]\theta_{13}$
10.	$(7)[ga]\theta_6 = \theta_4\{((19,21),\theta_5)/7\}$	34.	$(17)[a]\theta_{14} = \{((9,11),\theta_{13})/17\}$
11.	$(8)[ga]\theta_6$	35.	$(18)[a]\theta_{14}$
12.	$(17)[ga]\theta_7 = \{((9,11),\theta_6)/17\}$	36.	$(11)[a]\theta_{15} = \theta_{13}\{((19,21),\theta_{14})/11\}$
13.	$(18)[ga]\theta_7$	37.	$(12)[a]\theta_{15}$
14.	$(11)[ga]\theta_8 = \theta_6\{((19,21),\theta_7)/11\}$	38.	$(29)[a]\theta_{16} = \theta_3\{((13,15),\theta_{15})/29\}$
15.	$(12)[ga]\theta_8$	39.	$(30)[a]\theta_{16}$
16.	$(31)[ga]\theta_9 = \theta_3\{((13,15),\theta_8)/31\}$	40.	$(13)[a]\theta_{15}$
17.	$(32)[ga]\theta_9$	41.	$(14)[a]\theta_{15}$
18.	$(33)[a]\theta_9$	42.	$(19)[a]\theta_{11}$
19.	$(34)[a]\theta_9$	43.	$(20)[a]\theta_{11}$
20.	$(15)[a]\theta_8$	44.	$(5)[a]\theta_{10}$
21.	$(16)[a]\theta_8$	45.	$(6)[a]\theta_{10}$
22.	$(21)[a]\theta_5$	46.	$(27)[a]\theta_3$
23.	$(22)[a]\theta_5$	47.	$(28)[a]\theta_3$
24.	$(5)[a]\theta_4$	48.	$(28)[\exists]\theta_3$

Fig. 7. The single play of Example 12

The interpolation tree with a solution term is presented in Figure 6. We now examine the game for this tree which consists of the single play in Figure 7. The positions 1 to 48 are presented in two columns; 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 the binding node (1); any node n such that $(1) \downarrow n$ is, therefore, either associated with (the subtree at) (17) or (23) and the empty look-up table θ_1; this means that the look-up table θ_2 at this position has the single entry $\theta_2(1) = ((17,23),\theta_1)$: in examples, we reduce the number of brackets by abbreviating nodes such as (1) and (21) to 1 and 21 in look-up tables. Position 3 is therefore at node (2) labelled x_2 which is bound by (1); therefore, play jumps to node (23) with look-up table θ_1 which is updated because (23) is a binding node; therefore, the look-up table θ_3 at position 4 has the single entry $\theta_3(23) = ((3),\theta_2)$ which represents that any node labelled y_3 bound by (23) is associated with (3), the successor of (2), and look-up table θ_2. Position 5 is at node (24) which is bound by (23); so, play jumps to (3) and θ_2 is updated with the entry $((25,31),\theta_3)/3$; the successors of (24) are associated with the nodes bound by (3) and the look-up table θ_3. Position 7 is at (4); so play now jumps to (17), descends to (18) and then jumps to node (7) and then to (8). Therefore, play jumps to (17), descends to (18) and jumps to (11), the second successor of (8).

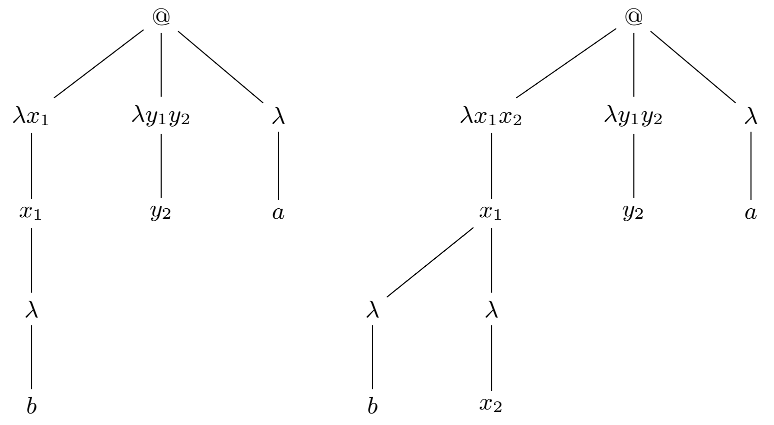

Fig. 8. Why long normal forms matter

Position (15) is, therefore, at node (12) which is bound by (3): so play jumps to (31) and descends to (32) labelled with the constant g; the current state is $[ga]$ and so play proceeds to node (33) with new state $[a]$. After position 19 it jumps to (15), the second successor of (12), and then to (16) which is bound by (7). Play therefore jumps to (21), the second successor of (18), and so position 23 is at node (22) which is bound by (17). Play thereby jumps to (5), the first successor of (4), and then to (6) before jumping to (25) at position 26. The reader is invited to check the rest of the play and why positions 44 and 45 at (5) and (6) are followed by a jump to (27) (and not (25)); so, play ends at (28) with refuter losing. $\square$

4.2. *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 presented in Figure 8. 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 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.

First, some notational abbreviations that we will employ. We assume that if π is a play in $\mathsf{G}(t,E)$ then it has the form $n_1[r_1]\theta_1, \ldots, n_l[r_l]\theta_l$ and π_i is the ith position $n_i[r_i]\theta_i$ for $i : 1 \leq i \leq l$. Given $\pi \in \mathsf{G}(t,E)$, we allow ourselves to write n_i or $[r_i]$ or θ_i to mean the node or state or look-up table of position π_i.

We begin with some some simple observations; the number of branches in a term u is the number of leaves in its tree representation.

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 the number of branches in u.

If t does not solve E then the number of plays in $\mathsf{G}(t,E)$ can be less than the number of branches u.

Fact 11. Assume $\pi \in \mathsf{G}(t,E)$.

1. If $i < l$ and $i = 2k$ for $k > 0$ then n_i is a lambda node.
2. If $i = 2k+1$ for $k > 0$ then n_i is a variable or a constant node.
3. n_l is a constant node 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 ga has only one branch there is a single play; each even position that is not final is at a lambda node such as positions 6 at node (3) and 18 at node (33); odd positions are at variable or constant nodes with the initial position at the root labelled @. A term of a later state is a subterm of that of an earlier state: after position 17, the term is a and until then it is ga. Play must end at a constant node.

The same look-up table may occur multiple times in a play. To be precise, 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 $\pi \in \mathsf{G}(t,E)$, $1 \leq j \leq l$ and $\theta_j(m) = ((m_1, \ldots, m_p), \theta')$.

1. $\theta_j(n)$ is defined iff n is a binding node above n_j.
2. $\theta'(n)$ is defined iff n is a binding node strictly above each m_k, $1 \leq k \leq p$.
3. For some y, m is labelled $\lambda y_1 \ldots y_p$ and each y_k has the same type as m_k, $1 \leq k \leq p$.
4. There is an $i < j$ such that $\theta' = \theta_i$ and either $i = 1$ and each $m_k = n_1(k+1)$ or n_i is a variable node and each $m_k = n_i k$, $1 \leq k \leq p$.

Proof. We prove the four properties together by simultaneous induction on the position j. For the base case $j = 1$, they all hold because θ_1 is empty, there are no nodes above the root labelled @ and the root node is not a binding node.

For the general case assume that the four properties hold for all positions before $n_s[r_s]\theta_s$. We show that they hold at $n_s[r_s]\theta_s$ by cases on the label at node $n = n_{s-1}$.

@: this means that $s = 2$ and $n_s = n1$. The only entry of θ_2 is that for $n1$ which is labelled $\lambda x_1 \ldots x_k$, which shows 1 (as $n1$ is above $n1$). For 2, we note that $\theta_2(n1) =$

$((n2,\dots,n(k+1)),\theta_1)$, θ_1 has no entries and there are no binding nodes strictly above $n2,\dots,n(k+1)$. 3 is clear as each x_k has the same type as $n(k+1)$. 4 is also clear.

$\lambda\overline{y}$: so $n_s = n1$ and $\theta_s = \theta_{s-1}$. We know that $n1$ is not a binding node and so all four hold by the induction hypothesis.

a: so $n_s = n$ and $\theta_s = \theta_{s-1}$. Therefore, the results hold by the induction hypothesis.

f: so $n_s = n$ or $n_s = nq$ for some q and in both cases $\theta_s = \theta_{s-1}$. Node nq is not a binding node (by assumption) and so the results follow by the induction hypothesis.

y_q: assume $m \downarrow n$ and $\theta_{s-1}(m) = ((m_1,\dots,m_p),\theta')$ for some $p \geq q$. Therefore, $n_s = m_q$ and $\theta_s = \theta'$ when $y_q : \mathbf{0}$ and so by the induction hypothesis m_q is then labelled with a dummy lambda (as it has type $\mathbf{0}$ too). Otherwise, $\theta_s = \theta'\{((n1,\dots,nr),\theta_{s-1})/m_q\}$ and $y_q : (B_1,\dots,B_r,\mathbf{0})$ and by the induction hypothesis m_q has the same type (so, is labelled $\lambda z_1\dots z_r$ for some z). As $n_s = m_q$ we note that the binding nodes above m_q are those strictly above $m_1,\dots,m_p$ (which by the induction hypothesis on 4 have the form $n'1,\dots n'p$) and so are strictly above m_q and the node m_q when y_q is not of base type. Thus, 1 holds for θ_s using the induction hypothesis for 2 (for θ') and the entry for m_q in θ_s when y_q has higher type. For the remaining cases assume $\theta_s(m') = ((m'_1,\dots,m'_{p'}),\theta'')$. So, either this is also an entry for θ' and therefore 2, 3 and 4 follow by the induction hypothesis on θ' (which is θ_i for $i < s$) or $m' = m_q$, $\theta'' = \theta_{s-1}$ and $m'_1,\dots,m'_{p'}$ are $n1,\dots,nr$; and so 2, 3 and 4 hold.

□

The binding nodes (1), (3) and (7) occur above (7) in Figure 6. At position 10 in the game of Figure 7 at (7), $\theta_6 = \{((17,23),\theta_1)/1,((25,31),\theta_3)/3,((19,21),\theta_5)/7\}$ has precisely three entries: moreover, the only entry for θ_5 is node (17) which is strictly above nodes (19) and (21) (which are successors of (16)). A simple corollary of this proposition part 3 is a type preservation feature that we have already described: if a position is at a variable of type A then the next position (involving a "jump") is at a lambda node with the same type A. Together with part 1 and the definition of a next position, play in a game cannot become stuck: if a current position is at a lambda node then the next position is at its successor node; if it is at a variable node then there is an entry for its binder in the current look-up table.

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 as the proof of the following theorem, that the game *characterises* interpolation, shows.

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

Proof. Assume E is $xw_1\dots w_k = u$ where $x : A$ and $t : A$ is a closed term in lnf. We define leftmost reduction sequences using the following two rules:

1. $(\lambda y_1\dots y_p.t')t_1\dots t_p \to t'\{t_1/y_1,\dots,t_p/y_p\}$,
2. $ft_1\dots t_p \to t_j$ for $j : 1 \leq j \leq p$.

The second rule allows for reduction under constants. A term t' is *terminal* if neither rule applies to it. Assume $v : \mathbf{0}$ is closed in normal form and does not contain binders.

A reduction sequence of length 0 of the form t_1 is *compatible* with v iff if t_1 is terminal then $t_1 = v$. When the sequence $t_1 \to \ldots \to t_k$ of length more than 0 is *compatible* with v is defined by cases on the initial reduction step $t_1 \to t_2$,

— instance of rule 1: iff $t_2 \to \ldots \to t_k$ is compatible with v.
— $t_1 = fs_1 \ldots s_p$ and $t_2 = s_j$: iff $v = fv_1 \ldots v_p$ and $t_2 \to \ldots \to t_k$ is compatible with v_j.

We slightly enlarge reduction sequences by allowing a third reduction rule for padding: $t \to t$. A *padding* of $t_1 \to \ldots \to t_m$ includes 0 or more applications of the padding rule as well as each $t_i \to t_{i+1}$. Let σ range over reduction sequences.

Consider the game $\mathsf{G}(t, E)$. The tree representation of t and each w_i introduces extra dummy lambda nodes which we assume are not present in the reduction sequences. Given a node of the interpolation tree n and a look-up table θ we define the *associated* term as $\theta[n]$ as follows by cases on the label at n:

@: $\theta[n1]\,\theta[n2] \ldots \theta[n(k+1)]$,
a: a,
f: $f\,\theta[n1] \ldots \theta[np]$, f has arity p
λ: $\theta[n1]$,
$\lambda\overline{z}$: $\lambda\overline{z}.\theta[n1]$, $\overline{z}$ not empty
y_j: of type $\mathbf{0}$; if $m \downarrow n$ and $\theta(m) = ((m_1, \ldots, m_p), \theta')$ then $\theta'[m_j]$ else y_j,
y_j: of type $(B_1, \ldots, B_q, \mathbf{0})$; if $m \downarrow n$ and $\theta(m) = ((m_1, \ldots, m_p), \theta')$ then $\theta'[m_j]\,\theta[n1] \ldots \theta[nq]$ else $y_j\,\theta[n1] \ldots \theta[nq]$.

We shall show that there is a correspondence between terms in reduction sequences and associated terms in game plays.

Assume a reduction sequence σ that is compatible with u and whose initial term is $t_1 = tw_1 \ldots w_k$. We show that there is a play $\pi \in \mathsf{G}(t, E)$ and a padding of σ of the form $t_1 \to \ldots \to t_m$ such that $t_j = \theta_{2j-1}[n_{2j-1}]$ for all $j : 1 \leq j \leq m$. The proof is by induction on i equal to the length of σ.

The base case is $i = 1$. The first position of any play $\pi \in \mathsf{G}(t, E)$ is $n[u]\theta_1$ where n is the root of the interpolation tree. So, $\theta_1[n] = \theta_1[n1]\,\theta_1[n2] \ldots \theta_1[n(k+1)]$ as n is labelled @. As θ_1 is empty, and the definition of $\theta[n]$ removes dummy lambdas $\theta_1[n1] = t$ and $\theta_1[n(i+1)] = w_i$. So, $\theta_1[n_1] = t_1$.

Assume the result holds for $i \leq q$: there is a $\pi \in \mathsf{G}(t, E)$ and a padding $t_1 \to \ldots \to t_s$ such that $t_i = \theta_{2i-1}[n_{2i-1}]$. This means that if $t_{i-1} = ft'_1 \ldots t'_p$ and $t_i = t'_j$ then $\theta_{2i-3}[n_{2i-3}] = t_{i-1}$ and $\theta_{2i-1}[n_{2i-1}] = t_i$; so, π follows the same branch of u as the reduction sequence. We consider $i = q+1$ and assume a reduction step $t_q \to t_{q+1}$ so that $t_1 \to \ldots \to t_{q+1}$ is compatible with u. We know that $t_q = t_s = \theta_{2s-1}[n_{2s-1}]$. We proceed by cases on the label at n_{2s-1} (which cannot be a lambda node by Fact 10).

@: therefore, $s = 1$. Assume $t = \lambda x_1 \ldots x_k.t'$ So, $t_1 \to t'\{w_1/x_1, \ldots, w_k/x_k\} = t_2$. Consider π_3 (for any play): n_3 is the root of t' and $\theta_3 = \theta_2 = \{((n2, \ldots, n(k+1)), \theta_1)/n_2\}$ where n is the root of the interpolation tree. Node n_2 binds each node labelled with x_i. By the induction hypothesis we know that $\theta_1[n(i+1)] = w_i$. Therefore, $\theta_3[n_3] = t_2$.
a: cannot arise as then $t_s = a$.
f: so $t_q = t_s = ft'_1 \ldots t'_p$ and $t_{q+1} = t_{s+1} = t'_j$. Consequently, $\theta_{2s-1}[n_{s-1}] = ft'_1 \ldots t'_p = f\theta_{2s-1}[n1] \ldots \theta_{2s-1}[np]$. Because $t_1 \to \ldots \to t_{q+1}$ is compatible with u and $\pi_1, \ldots, \pi_{2s-1}$

has followed the same branch of u as $t_1 \to \ldots \to t_q$, π_{2s} will not be the final position of π. We choose as the play π a continuation of $\pi_1, \ldots, \pi_{2s-1}$ such that n_{2s} is the jth successor of n_{2s-1}. So, n_{2s+1} is the successor of n_{2s} and $\theta_{2s+1} = \theta_{2s-1}$. Therefore, $\theta_{2s+1}[n_{2s+1}] = t'_j$.

y_j: by Proposition 12 we know that there is an entry in θ_{2s-1} for the binding node m such that $m \downarrow n_{2s-1}$. Let $\theta_{2s-1}(m) = ((m_1, \ldots, m_p), \theta')$. First, assume $y_j : (B_1, \ldots, B_r, \mathbf{0})$. So, $t_q = t_s = \theta_{2s-1}[n_{2s-1}] = \theta'[m_j]\theta_{2s-1}[n1] \ldots \theta_{2s-1}[nr]$ where ni is the ith successor of n_{2s-1}. This has the form $(\lambda y_1 \ldots y_r.t')w'_1 \ldots w'_r$. So, $t_{s+1} = t_{q+1} = t'\{w'_1/y_1, \ldots, w'_r/y_r\}$. Now $n_{2s} = m_j$ and $\theta_{2s} = \theta'\{((n1, \ldots, nr), \theta_{2s-1})/m_j\}$. By the induction hypothesis $w_i = \theta_{2s-1}[n_i]$ and $\theta'[m_j] = \lambda y_1 \ldots y_r.t'$; therefore, $\theta_{2s+1} = t_{s+1}$, as required. Next, assume $y_j : \mathbf{0}$. It is only this case that introduces padding. So, $t_s = \theta_{2s-1}[n_{2s-1}]$ which is $\theta'[m_j]$. Moreover, $\theta'[m_j] = \theta'[m']$ when m' is the successor of m_j since it is labelled with a dummy lambda. However, $n_{2s} = m_j$ and $\theta_{2s} = \theta'$; so, $n_{2s+1} = m'$ and $\theta_{2s+1} = \theta'$. Therefore, $\theta_{2s+1}[n_{2s+1}] = t_s$ so we add the padding move $t_s \to t_{s+1} = t_s$. Notice, by Proposition 12 that $\theta' = \theta_i$ for $i < 2s - 1$. Now, we repeat this argument for node n_{2s+1} by examining its label. It is not possible to enter into an infinite sequence of paddings because each time we invoke a look-up table from an earlier position.

An almost identical argument shows that given any *prefix* π of a play in $\mathsf{G}(t, E)$ there is a reduction sequence σ and a padding of σ of the form $t_1 \to \ldots \to t_m$ such that $\theta_{2j-1}[n_{2j-1}] = t_j$ for $j : 1 \le j \le m$. The proof proceeds by induction on the length of π. One imediate corollary of this argument is that because of strong normalisation there cannot be an infinite length play in $\mathsf{G}(t, E)$.

Now we can prove the main theorem. A *maximal* reduction sequence is one whose last term is terminal. If t solves E then by strong normalisation every maximal reduction sequence is finite and compatible with u. So there are corresponding plays $\pi \in \mathsf{G}(t, E)$ which $\forall$ loses. Moreover, there cannot be any other plays (as there would be corresponding reduction sequences). So $\forall$ loses the game $\mathsf{G}(t, E)$. Conversely, if $\forall$ loses $\mathsf{G}(t, E)$ then for each play π there is a corresponding reduction sequence compatible with u and all branches of u are covered, so t solves E. $\square$

One corollary of the proof of this result is that there cannot be an infinite length play in $\mathsf{G}(t, E)$. Another is the following result that the same look-up table cannot occur in different positions at the same node.

Proposition 14. Assume $\pi \in \mathsf{G}(t, E)$ and $i \neq j$. If $n_i = n_j$ then $\theta_i \neq \theta_j$.

Proof. This follows from the proof of Theorem 13. Given a prefix of π of length at least $j + 1$ there is a reduction sequence σ and a padding of σ of the form $t_1 \to \ldots \to t_m$ such that $t_k = \theta_{2k-1}[n_{2k-1}]$ for all $k : 1 \le k \le m$. Assume $n_i = n_j$ and $\theta_i = \theta_j$; if i is even then $n_{i+1} = n_{j+1}$ and $\theta_{i+1} = \theta_{j+1}$; so, assume $i = 2l - 1$ and $j = 2q - 1$ are odd. Now since $\theta_i[n_i] = \theta_j[n_j]$ we have that $t_l \to \ldots \to t_q = t_l$. Now using the observation from the proof in Theorem 13 that they cannot all be padding steps this would contradict strong normalisation. $\square$

$$\begin{tikzcd}
\lambda y_1 \dots y_p & n_i[r_i]\theta_i \arrow[d, dotted, no head] & \theta_i(n_i) = ((m_1,\dots,m_p),\theta) \\
y_k & n_j[r_j]\theta_j & \theta_j(n_i) = ((m_1,\dots,m_p),\theta)
\end{tikzcd}$$

Fig. 9. Position π_i is the parent of the position π_j

4.3. *Parent and child positions*

When a position is at a variable node, we identify the earlier position at its binding node which determines where play will jump to next.

Definition 21. Assume that $\pi \in \mathsf{G}(t, E)$ and n_j is a variable node. The position π_i is the *parent* of the later position π_j if $n_i \downarrow n_j$ and $\theta_i(n_i) = \theta_j(n_i)$. We also say that π_j is a *child* of π_i.

Besides the condition that n_i binds n_j, there is also the requirement that the look-up tables agree on their entry for the binder at n_i. Node n_i is labelled with $\lambda y_1 \dots y_p$ for some $p > 0$ and n_j is labelled with y_k for some k, a situation depicted in Figure 9.

In Figure 7 node (3) binds (6) label. In the play in Figure 6 position 6 at (3) is the parent of position 25 at (6). However, position 6 is not the parent of 45 at (6) because $\theta_{45}(3) = ((27, 29), \theta_3)$ and $\theta_6(3) = ((25, 31), \theta_3)$; the parent of 45 is position 28.

Proposition 15. Assume $\pi \in \mathrm{G}(t, E)$ and n_j is at a variable node. There is a unique $i < j$ such that π_i is the parent of π_j.

Proof. Assume $\pi \in \mathrm{G}(t, E)$, $n_i \downarrow n_j$, $\theta_j(n_i) = ((m_1, \dots, m_p), \theta)$ and n_j is a variable node. The first possibility is that there is not an earlier position π_k at node n_i with $\theta_k(n_i) = \theta_j(n_i)$. However, the only time an entry for a binding node m enters a look-up table is when play is at m; see the two cases @ and $y_j : (B_1, \dots, B_p, \mathbf{0})$ of Figure 5. Therefore there must be at least one such position. Next assume that there is more than one, π_k and $\pi_{k'}$. So, $\theta_k(n_i) = \theta_{k'}(n_i) = \theta_j(n_i)$. However, π_{k-1} and $\pi_{k'-1}$ must be at the same node (labelled with a variable or @) since $m_1, \dots, m_p$ are its successors with the same look-up table θ; this contradicts Proposition 14. □

We extend the definition of parent position to positions at lambda nodes.

Definition 22. Assume $\pi \in \mathsf{G}(t, E)$ and n_j is a lambda node. We define π_i the *parent* of π_j by cases on the label at the node n directly above n_j in the interpolation tree:
@: $i = 1$;
f: $i = j - 1$;
z: i is such that π_{i+1} is the parent of π_{j-1} (according to Definition 21).

For these three cases if π_i is the parent of π_j then n_j is a successor of n_i in the interpolation tree. For example, the third case is illustrated in Figure 10. Position π_i is at a node labelled with a variable z and π_j is at its kth successor; position π_{j-1} is at a variable y_k and it is a child of position π_{i+1}.

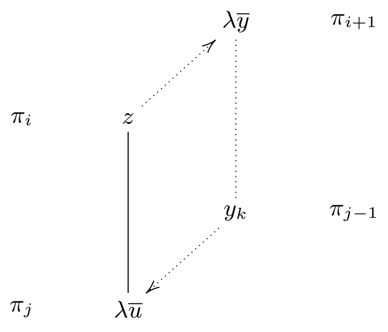

Fig. 10. Position π_j is a child of the position π_i

We illustrate these cases with the play of Figure 7 on the tree of Figure 6. Position 1 at the root labelled @ is the parent of position 8. Position 17 at the constant node is the parent of 18. Position 27 at the variable node labelled y is the parent of 46: position 45 at the variable node 6 is a child of position 28 at the lambda node 3.

Fact 16. Assume $\pi \in \mathsf{G}(t, E)$ and n_j is a lambda node. There is a unique $i < j$ such that π_i is the parent of π_j.

A very useful abstraction is to chain together sequences of parent child positions.

Definition 23. Assume $\pi \in \mathsf{G}(t, E)$ and π_j is a lambda node. The sequence $\pi_{j_1}, \ldots, \pi_{j_{2p}}$ is a *chain* (of parent child positions) for π_j if

1. $\pi_j = \pi_{j_{2p}}$,
2. for $1 \leq m \leq p$, $\pi_{j_{2m-1}}$ is the parent of $\pi_{j_{2m}}$,
3. for $1 \leq m < p$, $\pi_{j_{2m+1}}$ is the next position after $\pi_{j_{2m}}$.

If $\pi_{j_1}, \ldots, \pi_{j_{2p}}$ is a chain for π_j then $n_{j_1}, \ldots, n_{j_{2p}}$ is a path (a contiguous sequence of nodes) in the interpolation tree. For instance, in the case of the play in Figure 6 there is the following chain for position 38 which starts at the root of the tree in Figure 6; we include the position, the node of the interpolation tree and its label.

$$1\ (0)@, 4\ (23)\lambda y_3, 5\ (24)y_3, 26\ (25)\lambda, 27\ (26)y_3, 38\ (29)\lambda w_1 w_2$$

Assume a position π_j at a lambda node n_j and a node m that occurs above n_j (so, there is a path from m to n_j) that is labelled with a constant, variable or @; using Fact 16, there is a unique chain $\pi_{j_1}, \ldots, \pi_{j_{2p}}$ whose first position is at m and whose last position is π_j.

Definition 24. 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 17. Assume $\pi \in \mathsf{G}(t, E)$ and n_j is a lambda node. If π_i is the parent of π_j then $\theta_i \leq \theta_j$.

Proof. Assume π_i is the parent of π_j when n_j is a lambda node. We consider the label of n_i (which is directly above n_j). If it is @ then $i = 1$ and $\theta_1 \leq \theta_k$ for all k because $\theta_1 = \emptyset$. If it is a constant f then $i = j-1$ and $\theta_j = \theta_i$. The final case is a variable z of higher type, see Figure 10. So, $\theta_{i+1} = \theta'\{((m_1, \ldots, m_p), \theta_i)/n_{i+1}\}$ for some θ' and where $m_1, \ldots, m_p$ are the successors of n_i. Assume n_{i+1} is labelled $\lambda y_1 \ldots y_p$, n_{j-1} is labelled y_k and π_{i+1} is the parent of π_{j-1}: so $\theta_{j-1}(n_{i+1}) = \theta_{i+1}(n_{i+1})$. If $y_k : \mathbf{0}$ then $\theta_j = \theta_i$ and the result holds. Otherwise $y_k : (B_1, \ldots, B_q, \mathbf{0})$ for some q and so $\theta_j = \theta_i\{((m'_1, \ldots, m'_q), \theta_{j-1})/n_j\}$ where $m'_1, \ldots, m'_q$ are the successors of n_{j-1}. By Proposition 12 part 1, θ_i does not have an entry for the node n_j; therefore, $\theta_i \leq \theta_j$. □

Proposition 17 also holds for the case that π_i is a parent of π_j when n_j is a variable, which is a corollary of the next result.

Proposition 18. Assume $\pi \in \mathsf{G}(t, E)$ and $\pi_{j_1}, \ldots, \pi_{j_{2p}}$ is a chain for π_j.

1. $\theta_{j_1} \leq \ldots \leq \theta_{j_{2p}}$.
2. If $n_{j_m} \downarrow n_{j_n}$ then π_{j_m} is the parent of π_{j_n}.

Proof. Assume $\pi_{j_1}, \ldots, \pi_{j_{2p}}$ is a chain for π_j. For 1, by Proposition 17 we know $\theta_{2m-1} \leq \theta_{2m}$, $1 \leq m \leq p$. Moreover, since $\pi_{j_{2k+1}}$ is the next position after $\pi_{j_{2k}}$ and $n_{j_{2k}}$ is a lambda node, $1 \leq k < p$, it follows that $\theta_{j_{2k+1}} = \theta_{j_{2k}}$ according to Figure 5. For 2 we use 1 and Proposition 15: assume $n_{j_m} \downarrow n_{j_n}$, so the parent of π_{j_n} is at node n_{j_m}; however, $\theta_{j_m} \leq \theta_{j_n}$ so $\theta_{j_m}(n_{j_m}) = \theta_{j_n}(n_{j_m})$. □

The following is another useful consequence of the notion of chain.

Proposition 19. Assume $\pi \in \mathsf{G}(t, E)$, $\theta_i(m), \theta_j(m)$ are both defined and $\theta_i(m) = \theta_j(m)$. If m' is a binding node above m in the interpolation tree then $\theta_i(m') = \theta_j(m')$.

Proof. Assume $\pi \in \mathsf{G}(t, E)$, $\theta_i(m)$, $\theta_j(m)$ are both defined and $\theta_i(m) = \theta_j(m) = ((m'_1, \ldots, m'_k), \theta)$. By Proposition 12 node m is a binding node above n_i and n_j. Without loss of generality assume n_i and n_j are both lambda nodes: if not then use their previous position at a lambda node with the same look-up table. Assume m' is a binding node above m and m' is a successor of n. Therefore, there is the chain $\pi_{i_1}, \ldots, \pi_{i_{2p}}$ for π_i starting at node n and there is the chain $\pi_{j_1}, \ldots, \pi_{j_{2q}}$ for π_j also starting at n. Consider the prefixes $\pi_{i_1}, \ldots, \pi_{i_{2r}}$ $\pi_{j_1}, \ldots, \pi_{j_{2r}}$ that end at node m. Because $\theta_{i_{2r}} \leq \theta_i$, $\theta_{j_{2r}} \leq \theta_j$ and $\theta_i(m) = \theta_j(m)$ it follows that $\theta_{i_{2r}}(m) = \theta_{j_{2r}}(m) = ((m'_1, \ldots, m'_k), \theta)$. Therefore, $\pi_{i_{2r}} = \pi_{j_{2r}}$; otherwise $\pi_{i_{2r}-1}$ and $\pi_{j_{2r}-1}$ are at the same node with the same look-up table contrary to Proposition 14. So, these prefixes are the same sequence; so, $\theta_i(m') = \theta_j(m')$ since $\theta_{i_2} \leq \theta_i$ and $\theta_{i_2} \leq \theta_j$. □

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 use 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 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 Example 12. 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 25. 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 $\Gamma(y) = \{(y, r, \Gamma) \,|\, r \in \mathrm{Sub}(u), \Gamma \subseteq \bigcup_{1 \le i \le p} \bigcup_{z : B_i \in V} \Gamma(z)\}$. 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, as we shall see, to be an abstraction from subsequences of positions. Consider the play in Figure 7 on the interpolation tree of Figure 6. A $z_1 : \mathbf{0}$ profile has the form $(z_1, r, \emptyset)$ where $r \in \mathrm{Sub}(u)$, which captures a game position at node (6) labelled z_1; there are two such positions 25 and 45 with the same associated profile $(z_1, a, \emptyset)$. The key point is that there is the upper bound of the size of $\mathrm{Sub}(u)$ on the number of such profiles (which, in principle, is much smaller than an upper bound on the number of times a play may be at a node labelled z_1). In this sense a variable profile is an *abstraction.* A $y_2 : (\mathbf{0}, \mathbf{0}, \mathbf{0})$ profile captures positions at node (18) and has the form (y_2, r, Γ') where Γ' is a set of profiles for variables $z : \mathbf{0}$. Assume a position $n[r]\theta$ at a node labelled y_2 and a next position $n'[r]\theta'$. The associated profile is (y_2, 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 case of position 9 the corresponding profile is $(y_2, ga, \emptyset)$ whereas for position 31 it is $(y_2, a, \{(s_1, a, \emptyset)\})$ (as position 41 is a child of 32).

For higher-order variables such as $x_2 : ((\mathbf{0}, (\mathbf{0}, \mathbf{0}, \mathbf{0}), \mathbf{0}), \mathbf{0})$ of Figure 6 the pattern of a profile is the same. For instance, position 3 of Figure 6 has the associated profile in

$$\begin{array}{ll}(x_2, ga, \{ & (y_3, ga, \{(z_2, ga, \{(v_2, a, \emptyset)\}), (z_1, a, \emptyset)\}) \\ & (y_3, a, \{(z_2, a, \{(w_1, a, \emptyset)\}), (z_1, a, \emptyset)\})\}\end{array}$$

Fig. 11. A x_2 profile for Example 12

Figure 11. There are two children of position 4, namely positions 5 and 27 at nodes (24) and (26) labelled with y_3: in turn, positions 6 and 28 both have two children.

A mode is a pair (r, Γ) where $r \in \mathrm{Sub}(u)$ and Γ is a set of variable profiles. Because $\mathrm{Sub}(u)$ 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 cannot share variables; for any tree which is not well-named, 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 26. Assume an interpolation equation E, $xw_1 \dots w_k = u$, a fixed finite alphabet Σ and an interpolation Σ-tree $@tw_1 \dots w_k$ 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 $\{(r_1, \Gamma_1), \dots, (r_m, \Gamma_m)\}$, $m \geq 0$, $r_i \in \mathrm{Sub}(u)$ 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 $\Lambda \subseteq \{(r, \{(y, r, \emptyset)\}) \mid r \in \mathrm{Sub}(u)\}$.

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

(a) 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

(b) 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 $\Lambda_1, \dots, \Lambda_p, \Lambda$ such that

(a) 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

(b) 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.

$@ : ((A_1, \ldots, A_k, \mathbf{0}), A_1, \ldots, A_k, \mathbf{0})$; then $@\Lambda_1 \ldots \Lambda_{k+1} \Rightarrow \{(u, \emptyset)\}$ for each $\Lambda_1, \ldots, \Lambda_{k+1}$ such that $\Lambda_1 = \{(u, \Gamma)\}$ for some Γ and

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

(b) 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 play at a node are associated with modes at that node; and vice versa. For example, if (r, Γ) is at the constant node n labelled f then r is $fr_1 \ldots r_p$, for some $r_1, \ldots, r_p$, and for each i, there is a mode (r_i, Γ_i) at the ith successor ni of n so that $\Gamma = \bigcup \Gamma_i$. If (z, r', Γ') is a variable profile in Γ and (r, Γ) is a mode at node n then z occurs free in the subtree rooted at n or in the subtree rooted at its successor if n is labelled $\lambda\overline{z}$. If (r, Γ) is at a variable node n labelled $y : (A_1, \ldots, A_p, \mathbf{0})$ and $n[r]\theta$ is its associated position in a play, then 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 in the play: this position is at a lambda node labelled $\lambda z_1 \ldots z_p$ for some $z_1, \ldots, z_p$. In the case of the transition for @, there is a single mode $(u, \emptyset)$ at the root of the interpolation tree and there is a single mode (u, Γ) at its first successor (the root of t labelled $\lambda x_1 \ldots x_k$); each variable profile in Γ is of the form (x_i, r_i, Γ_i) representing a later position in a play $m[r_i]\theta'$ at a node m labelled x_i and so (r_i, Γ_i) is therefore a mode at the $(i+1)$th successor of the root (the root of w_i) as play jumps from m to this node.

Example 13. 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 $\{(a, \Gamma_2)\}$. The transitions for variables y_1 and y_3 of type $\mathbf{0}$ are as follows.

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

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

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 $(x_2, faa, \{(y_3, a, \emptyset)\})$.

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

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

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

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

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 @.

$$\lambda\ \Lambda\Rightarrow\Lambda \qquad \lambda x_1x_2\ \Lambda\Rightarrow\Lambda$$

$$\lambda y_1y_2\ \Lambda\Rightarrow\Lambda \qquad \lambda y_3\ \Lambda\Rightarrow\Lambda$$

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

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 $\lambda x_1x_2.x_1(x_1\ldots x_1(x_2(x_1\ldots x_1ab)b\ldots)\ldots)b$. □

Theorem 20. Assume E is the interpolation equation $xw_1\ldots w_k=u$ and Σ is a fixed finite alphabet. If $@tw_1\ldots w_k$ is a well-named Σ-tree then the automaton A_E accepts $@tw_1\ldots w_k$ iff t solves E.

Proof. Let E be $xw_1\ldots w_k=u$ and assume $@tw_1\ldots w_k$ is a well-named Σ-tree. 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 $@tw_1\ldots w_k$. 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 $@tw_1\ldots w_k$. 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 26. If n is labelled $y:\mathbf{0}$ then the state of the successful run at this node is $\{(r,\{(y,r,\emptyset)\})\,|\,\text{there is a position } n[r]\theta \text{ of a play in } \mathsf{G}(t,E)\}$. Again, this obeys the transition condition. Next we consider the general case of an arbitrary node n of $@tw_1\ldots w_k$. 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 26. If n is labelled $f:(A_1,\ldots,A_p,\mathbf{0})$ then because t solves E every position at n has the form $n[fr_1\ldots r_p]\theta$ and every subsequent position has the form $ni[r_i]\theta$ for $1\le i\le p$; therefore, for each such position at n there is a mode $(fr_1\ldots r_p,\Gamma)$ 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 26 are fulfilled. If n is labelled $y:(A_1,\ldots,A_p,\mathbf{0})$ 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 $\lambda z_1\ldots z_p$ for some $z_1,\ldots,z_p$. Assume all the children of $n'[r]\theta'$ are as follows

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

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 $\lambda s_1 \ldots s_q$ 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

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

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.

Assume A_E accepts $@tw_1 \ldots w_k$ but t does not solve E. We derive a contradiction. There is, therefore, a play $\pi = \pi_1, \ldots, \pi_l$ of $\mathsf{G}(t, E)$ that $\forall$ wins. As there is a successful run of A_E on $@tw_1 \ldots w_k$ 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 π_{l-1} either n_{l-1} is labelled with $a : \mathbf{0}$ and $r_{l-1} \neq a$ or n_{l-1} is labelled with f and r_{l-1} does not have the form $fr'_1 \ldots r'_p$; therefore, there cannot be a mode in the successful run of A_E at n_{l-1} that corresponds to this position because it would violate the transition rules of Definition 26. Therefore, consider the latest position $n_j[r_j]\theta_j$ in π for which there is a corresponding mode (r_j, Γ_j) at n_j in the accepting run of A_E. We now examine the label at n_j; it cannot be a constant a, f, @ or a $\lambda\overline{y}$. Therefore, it must be a variable y. There is at least one variable profile $(y, r_j, \Gamma'') \in \Gamma_j$ as (r_j, Γ_j) is a corresponding mode at n_j, by the transition rule for a variable y in Definition 26. Consider the next position π_{j+1} at n_{j+1} labelled $\lambda\overline{z}$ for some $\overline{z}$. So, for each profile $(y, r_j, \Gamma') \in \Gamma_j$, (r_j, Γ'') is not a mode at n_{j+1}. Assume π_i is the parent of π_j, so n_i is labelled $\lambda\overline{y}$. There is a corresponding mode (r_i, Γ_i) to this position such that $(y, r_j, \Gamma'') \in \Gamma_i$: node n_i occurs above n_j; as the interpolation tree is well-named, the only way the variable profile (y, r_j, Γ'') would not occur in Γ_i is if the binder for y occurs below n_i, as can be seen by examining the transition rules for A_E. If y is level 1 in t, so, $y = x_m$ for some m, then n_i is the root of t; so, n_{j+1} is the $(m+1)$th successor of the root and (r_j, Γ'') corresponds to position π_{j+1} is a mode at n_{j+1} by the transition rule for @. Otherwise, π_{j+1} is a child of π_{i-1} and n_{i-1} is labelled with a variable s and the result follows by considering the corresponding mode for π_{i-1} which 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 20 and overcome the restriction to the finite alphabet Σ. 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 $sq \Rightarrow q_1 \dots q_k$: if s has arity k and state q labels a node labelled s then $q_1, \dots, q_k$ can label its successors; compare Definition 14 and see (Comon et al 2002).

Definition 27. 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. $qs \Rightarrow (q_1, \dots, q_k)$ where $s \in \Sigma_2 \cup \Sigma_3$, $\mathrm{ar}(s) = k$, $q, q_1, \dots, q_k \in Q$;
2. $qs \Rightarrow q's'$ where $s \in \Sigma_1$, $s' \in \Sigma_3$ and $q, q' \in Q$;
3. $(q', q)s \Rightarrow q_1 x$ where $s \in \Sigma_1$, $x \in \Sigma_2$ and $q', q, q_1 \in Q$.

Definition 28. 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 $qs \Rightarrow (q_1, \dots, q_k) \in \Delta$ 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 $(q', q)s \Rightarrow q_1 x \in \Delta$ 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 21. Assume A, A_1 and A_2 are dependency Σ-tree automata and n is the size of A

1. The non-emptiness problem, given A is $\mathsf{T}_\Sigma(\mathsf{A}) \neq \emptyset$?, is decidable in order n^4 time.
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 in at most order n^4. The other parts of the theorem follow from the usual product and disjoint union of automata (extended here with the binding relations). □

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

Definition 29. An *alternating dependency Σ-tree automaton* $\mathsf{A} = (Q, \Sigma, q_0, \Delta)$ is as in Definition 27 except for the first clause for transitions which is now

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

Definition 30. A *run* of an 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 each set Q_i can be viewed as the conjunct $\bigwedge_{q \in Q_i} q$ 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 al 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 31. Assume the associated binding tree for the interpolation equation E, $xw_1 \ldots w_k = u$, Σ is its alphabet. The *alternating 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 $\Sigma_i = \{(r_{i_j}, \Gamma_{i_j}) \mid (x_i, r_{i_j}, \Gamma_{i_j}) \in \Gamma\}$
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. $(a, \emptyset)\lambda\overline{y} \Rightarrow (a, \emptyset)a$
5. $(r, \Gamma)z_j \Rightarrow (Q_1, \ldots, Q_k)$ if $Q_i = \{(r', \Gamma') \mid (y_i, r', \Gamma') \in \Gamma\}$ 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 $\lambda x_1 \ldots x_k.t'$ 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 $i : 1 \leq i \leq k$. The state at the node labelled $\lambda x_1 \ldots x_k$ 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 $\lambda z_1 \ldots z_p$ 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, $1 \leq j \leq p$. 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 $fr_1 \ldots r_k$. 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, $1 \leq i \leq p$ for some $y_1, \ldots, y_p$. 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 $(fr_1 \ldots r_k, \emptyset)$ 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 20, so we omit it here.

Theorem 22. *Assume E is the interpolation equation $xw_1 \ldots w_k = u$ and Σ is a finite alphabet. If $@tw_1 \ldots w_k$ is a binding Σ-tree then the alternating dependency Σ-tree automaton A_E (of Definition 31) accepts $@tw_1 \ldots w_k$ 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 37 of Section 6, is a small solution property: if an 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 4.2 and 4.3.

First, we isolate a key technical property of multiple child positions; see Definitions 21 and 22 for when a position is the parent of another position.

Proposition 23. *Assume $\pi \in \mathsf{G}(t, E)$ and $j < m$. If π_i is the parent of π_j and π_m then there is a $k : j < k < m$ such that π_j is the parent of π_k.*

Proof. We start with the case where n_i is a lambda node that binds both n_j and n_m and π_i is the parent of both π_j and π_m. So, by Definition 21, $\pi_i(n_i) = ((m_1, \ldots, m_p), \theta_l)$ $= \pi_j(n_i) = \pi_m(n_i)$ for some $m_1, \ldots, m_p$ and for some $l < i$ by Proposition 12. Position π_{j+1} is at m_q for some q and $\theta_{j+1} = \theta_l\{((n1, \ldots, nr), \theta_j)/m_q\}$ where $n1, \ldots, nr$ are the successors of n_j. The look-up table θ_l does not have an entry for n_i (because the move from n_j to n_{j+1} is a jump from t into some w_s or vice-versa, and so n_i cannot occur above n_{j+1}). If play does reach a variable node bound with a binder at n_b whose entry is in θ_l, say $\theta_l(n_b) = (((m'_1, \ldots, m'_a), \theta_c)$ then $c < l$. θ_c cannot have an entry for n_i of the form $((n1, \ldots, nr), \theta_j)$; for suppose it does then there there is a position π_d at n_i before position π_i such that π_{d-1} and π_{i-1} are at the same node with the same look-up table which contradicts Proposition 14. Therefore, the only opportunity to reach position π_m with its look-up entry for n_i is after a position bound by m_q; that is, a position that is a child of π_j.

We next consider the cases where π_i is the parent of π_j and π_m according to Definition 22, when n_j and n_m are lambda nodes. We now proceed by cases on the label at n_i. @ : then n_j and n_m are root nodes of t or w_q. Without loss of generality assume n_j is the root of t. There is a first position π_k after π_j (and before π_m) such that π_{k+1} is the result of a jump from t into some w_p; otherwise, π_m could not be at the root of t or w_q. Therefore, π_j is the parent of π_k. f : this case cannot arise as π_i has at most one child. y : therefore, π_{i+1} is the parent of π_{j-1} and of π_{m-1}. Therefore, by the first part of the proof, π_{j-1} is the parent of some π_{k+1} (before π_{m-1}); however, this means that π_j is the parent of π_k as required. □

In the play of Figure 7 on the tree of Figure 6, position 2 is the parent of positions 3 and 11; in this case position 3 is the parent of 6. Also, position 5 is the parent of 16 and 26; position 19 is a child of 16.

We now introduce interval notation on plays: $\pi[i, j]$ is the *sequence of positions* $\pi_i, \ldots, \pi_j$. Intervals correspond when they pass through the same sequence of nodes.

Definition 32. Assume $\pi \in \mathsf{G}(t, E)$. Two intervals $\pi[i, i+m]$ and $\pi[j, j+m]$ *correspond* if $n_{i+k} = n_{j+k}$ for each $k : 0 \leq k \leq m$.

We now present the key uniformity property when there are corresponding intervals with *almost* the same look-up tables.

Proposition 24. Assume $\pi \in \mathsf{G}(t, E)$, π_i is the parent of π_j, π_m, $j < m$, and n_j, n_m are both labelled with the same variable. There is a k such that $\pi[j+1, j+k]$ and $\pi[m+1, m+k]$ correspond and

1. (a) π_j is the parent of π_{j+k+1} and π_m is the parent of π_{m+k+1}, and (b) n_{j+k+1} is the p-successor of n_j iff n_{m+k+1} is the p-successor of n_m, or
2. n_{j+k} is labelled with a constant and π_{m+k} is the final position of π or $n_{j+k+1} \neq n_{m+k+1}$.

Proof. Assume π_i is the parent of π_j, π_m and n_j, n_m are labelled with the same variable. Therefore, $\theta_i(n_i) = \theta_j(n_i) = \theta_m(n_i) = ((m_1, \ldots, m_p), \theta_l)$ for some $m_1, \ldots, m_p$ and $l < i$. As n_j and n_m are labelled with the same variable y_q, $n_{j+1} = m_q = n_{m+1}$, $\theta_{j+1} = \theta_l\{((n_j1, \ldots, n_jr), \theta_j)/m_q\}$ and $\theta_{m+1} = \theta_l\{((n_m1, \ldots, n_mr), \theta_m)/m_q\}$ for some r. So, positions π_{j+1}, π_{m+1} correspond as do their continuations because θ_{j+1}, θ_{m+1} have the same entries except for m_q. There are two possibilities. First that play reaches π_{j+k} and $m_q \downarrow n_{j+k}$ (and $n_{m+k} = n_{j+k}$); so, π_{j+k+1} is a child of π_j and π_{m+k+1} is a child of π_m; moreover, n_{j+k+1} is the q-successor of n_j iff n_{m+k+1} is the q-successor of n_m. Second is that play reaches π_{j+k} and n_{j+k} is labelled with a constant f; also $n_{m+k} = n_{j+k}$ and either π_{m+k} is the final position of π or n_{j+k+1} is a different successor than n_{m+k+1} of n_{j+k}. □

The conditions described in this Proposition are now used in the following definition.

Definition 33. Assume $\pi \in \mathsf{G}(t, E)$ and n_j, n_m are both labelled with the same variable for $j < m$. We say that $\pi[j+1, j+k]$ and $\pi[m+1, m+k]$ are *complementary intervals* if 1 or 2 of Proposition 24 holds.

This definition does not require positions π_j and π_m to have a common parent.

In the play of Figure 12 on the tree of Figure 6, $\pi[6,15]$ and $\pi[28,37]$ are complementary; first, these intervals correspond; nodes (24) and (26) at positions 5 and 27 are both labelled with y_3; position 5 is a parent of 16 and 27 is a parent of 38; finally, the node at position 16 is the second successor of the node at position 5, and also the node at position 38 is the second successor of the node at position 27.

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

Definition 34. Assume an interpolation tree $@tw_1 \dots w_k$. A variable node in this tree is *level* 1 if it is bound by the root of t or w_q for some q. A variable node is *level* $j+1$ if it is bound by a successor node of a level j node.

4th or 5th order interpolation trees contain at most two levels of node. In Figure 6 nodes such as (2) or (24) are level 1 whereas (12) and (34) are level 2. In the general case, a $2n$ or $2n+1$ interpolation tree can contain level k nodes for $k : 1 \leq k \leq n$.

Definition 35. Assume $n' \downarrow n$, $n' \downarrow m$ and n, m are distinct nodes of an interpolation tree labelled with the same variable. Node m is *embedded* if m occurs below n in the tree.

In Figure 6, node (8) is embedded because it is below (4) and both (1) $\downarrow$ (4) and (1) $\downarrow$ (8); node (26) is also embedded since (23) $\downarrow$ (24) and (23) $\downarrow$ (26).

Definition 36. Assume n, m are variable nodes of an interpolation tree.

1. n is a *descendant under binding* of m (m is an *ancestor under binding* of n) if $m = n$ or m' is a successor of m, $m' \downarrow n'$ and n is a descendant under binding of n'.
2. n and m *belong to the same family* if they have a common ancestor under binding.
3. n is *end* if it has no descendants under binding except itself.

Node (2) of Figure 6 is an ancestor under binding of (6) and (12). Nodes (4) and (16) belong to the same family (but (8) and (12) do not). Node (8) is an example of an end node (because there are no nodes bound by (11)).

We now examine some further uniformity features in plays of a game. The first follows from Proposition 23.

Fact 25. Assume $\pi \in \mathsf{G}(t, E)$. 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 24.

Fact 26. Assume $\pi \in \mathsf{G}(t, E)$, and n_j, n_m are level 1 nodes in t labelled with the same variable. Then there is a k such that $\pi[j+1, j+k]$ and $\pi[m+1, m+k]$ are complementary.

In the play π of Figure 7 on the tree in Figure 6, π_7 and π_{11} are at level 1 nodes labelled with x_1. Here $\pi[8,9]$ and $\pi[12,13]$ are complementary (with π_{10} a child of π_7 and π_{14} a child of π_{11}).

The next property concerns embedded nodes and is a consequence of Propositions 23 and 24.

Fact 27. Assume $\pi \in \mathsf{G}(t, E)$, n, m are labelled with the same variable, m is below the p-successor of n in the interpolation tree and $n_j = m$. Then there is an i and k such that

1. $n_i = n$, $\pi[i+1, i+k]$ and $\pi[j+1, j+k]$ are complementary, and
2. if n, m are end and $[r_{i+1}] = [r_{i+k}]$ then n_{j+k+1} is the p-successor of n_j.

Node (26) in the tree in Figure 6 is an embedded node because of (24) and position 27 in the play π of Figure 7 is at (26). There is an earlier position 5 at 24 and $\pi[6, 15]$ and $\pi[28, 37]$ are complementary.

Next, we examine a feature of chains; recall that a chain $\pi_{j_1}, \ldots, \pi_{j_{2p}}$ is a sequence of parent child positions, Definition 23. We are mostly interested in chains of an interpolation tree $@tw_1 \ldots w_k$ when $n_{j_1}, \ldots, n_{j_{2p}}$ is a path in some w_q.

Proposition 28. Assume $\pi \in \mathsf{G}(t, E)$, π_j is a position in w_q for some q and $\pi_{j_1}, \ldots, \pi_{j_{2p}}$ is a chain for π_j. If $\pi_{j_{2k}-1}$ and $\pi_{j_{2m}-1}$ are positions in t, $1 \leq k \leq m \leq p$, then $n_{j_{2k}-1}$ and $n_{j_{2m}-1}$ belong to the same family.

Proof. Let $\pi_{j_1}, \ldots, \pi_{j_{2p}}$ be a chain for π_j where $\pi \in \mathsf{G}(t, E)$ and n_j is in w_q. Assume that $\pi_{j_{2k}-1}$ and $\pi_{j_{2m}-1}$ are positions in t, $1 \leq k \leq m \leq p$. We show that $n_{j_{2k}-1}$ and $n_{j_{2m}-1}$ belong to the same family. First, both are variable nodes since $n_{j_{2k}}$ and $n_{j_{2m}}$ are in w_q. Let $\pi_{i_1}, \ldots, \pi_{i_{2r}}, \pi_{j_1}, \ldots, \pi_{j_{2p}}$ be the chain for π_j where n_{i_1} the successor of the root node of w_q for some $r \geq 0$. So, the position $\pi_{i_1 - 1}$ is at the root of w_q and the previous position π_{i_1-2} is at a level 1 node n of t; it is straightforward to see that n is an ancestor under binding of both $n_{j_{2k}-1}$ and $n_{j_{2m}-1}$. □

We also want to consider chains that start from a particular node in w_q. To help with this we define a special kind of chain.

Definition 37. Assume $\pi \in \mathsf{G}(t, E)$. The chain $\pi_{j_1}, \ldots, \pi_{j_{2p}}$ for π_j has the *recurrence property* if for each $i : 1 \leq i \leq p$, if $n_{j_{2i-1}}$ is a variable node then there is a $j < j_{2i-1}$ and a k such that $\pi[j+1, j+k]$ and $\pi[m+1, m+k]$ are complementary where $m = j_{2i-1}$ and $j_{2i} = m + k + 1$.

Proposition 29. Assume $\pi \in \mathsf{G}(t, E)$ and n_i is a variable or constant node in w_q for some q. Then there is a largest $p \geq 0$ such that $\pi_{j_1}, \ldots, \pi_{j_{2p}}$ is a chain with the recurrence property where $i = j_1$.

Proof. Let $\pi \in \mathsf{G}(t, E)$ and n_i be a variable or constant node in w_q for some q. Assume we have so far constructed a chain $\pi_{j_1}, \ldots, \pi_{j_{2r}}$ with the recurrence property where $r \geq 0$ and $i = j_1$. We let $\pi_{i'} = \pi_i$ if $r = 0$ and otherwise $\pi_{i'} = \pi_{j_{2r}+1}$ at the successor node of $n_{j_{2r}}$. Consider the first position after $\pi_{i'}$, if there is one, which is a child of $\pi_{i'}$. If there is no such position then $p = r$. Otherwise $\pi_{j'}$ is this position. If $n_{i'}$ is a constant node then $\pi_{j'} = \pi_{i'+1}$; therefore we can (uniquely) extend the chain with $\pi_{i'}, \pi_{j'}$ and the argument is then repeated for $\pi_{i'} = \pi_{j'+1}$. If $n_{i'}$ is a variable node then we consider any previous position π_l at a node labelled with the same variable and examine whether $\pi[l+1, l+k]$ and $\pi[i'+1, i'+k]$ when $k+1 = j' - i'$ are complementary; if there is such an l then we can (uniquely) extend the chain with $\pi_{i'}, \pi_{j'}$ and the argument is then repeated for

$\pi_{i'} = \pi_{j'+1}$; if there is no such π_l then $p = r$. As $n_{j_1}, \ldots, n_{j_{2r}}$ is a path in w_q, the size of p in this contruction is bounded by the depth of w_q. □

6.2. *Two transformations*

We now define two local transformations on terms that preserve being a solution to an interpolation equation. 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 assume that if $t : A$ is closed then t' has type A and is also closed; we write tTt' if t' is the result of applying T to t.

For the first transformation, we remind the reader of Definition 10 of the constant $b : \mathbf{0}$ (which does not occur in the goal term of the interpolation equation).

Definition 38. Assume the game $\mathsf{G}(t, E)$ and n is a variable node or a constant node labelled with some $f : A \neq \mathbf{0}$ of t. If for every $\pi \in \mathsf{G}(t, E)$ and i, $n_i \neq n$ then *transformation* T_1 is $t[b/n]$.

If there is not a position of any play $\pi \in \mathsf{G}(t, E)$ at the variable or constant node n then it is inaccessible in the sense described in Section 3.2; the subtree rooted at n does not contribute to the normal form and therefore can be replaced. Application of this transformation either reduces the number of nodes or the number of variable nodes of a term.

Fact 30. If t solves E and tT_1t' then t' solves E.

The second transformation uses the notion of an end path which generalises Definition 36 of end node.

Definition 39. The path $m_1, \ldots, m_{2p}$ is *end* in t if

1. m_1 is a variable node,
2. for each $i : 1 < i \leq p$, m_{2i-1} is a variable node whose binder occurs in the path,
3. every variable node below m_{2p} in t is bound by a node that either occurs above m_1 or below m_{2p} in t.

For instance, the path (8), (11) in Figure 6 is end because node (11) does not bind nodes below it in t (and, so, all the variable nodes (12), (14) and (16) below (11) are bound above (8) or below (11) in t).

We define when an end path of nodes in t is redundant in the game $\mathsf{G}(t, E)$; which appeals to a notion of contribution towards a solution (of an interpolation equation).

Definition 40. Assume $\pi \in \mathsf{G}(t, E)$ and n_j is a lambda node. The interval $\pi[i, j]$ *contributes* if $[r_i] \neq [r_j]$.

An interval contributes if it contains a position at a constant node.

Definition 41. Assume $m_1, \ldots, m_{2p}$ is an end path in t.

1. This path is *redundant* in the interval $\pi[i,k]$ of $\pi \in \mathsf{G}(t,E)$ when: if π_{j_1}, $j_1 < k$, is the first position after π_i with $n_{j_1} = m_1$ then
   (a) there is a later position π_j, $j \leq k$, and a chain $\pi_{j_1}, \ldots, \pi_{j_{2p}}$ for π_j where $n_j = m_{2p}$,
   (b) $\pi[j_1, j]$ does not contribute, and
   (c) this path is redundant in the interval $\pi[j,k]$.
2. This path is *redundant* in the game $\mathsf{G}(t,E)$ if it is redundant in every play $\pi[1,l]$ (where π_l is the final position of π).

The end path $m_1, \ldots, m_{2p}$ is redundant in the interval $\pi[i,k]$ of $\pi \in \mathsf{G}(t,E)$ if no position in this interval is at m_1, or there is a first position π_{j_1} at m_1 and a later position π_j at m_{2p} such that there is a chain $\pi_{j_1}, \ldots, \pi_{j_{2p}}$ for π_j, $\pi[j_1, j]$ does not contribute (so, the states $[r_{j_1}]$ and $[r_j]$ are the same) and the end path is redundant in the smaller interval $\pi[j,k]$. It is redundant in $\mathsf{G}(t,E)$ if for each $\pi \in \mathsf{G}(t,E)$ it is redundant for the complete play interval $\pi[1,l]$.

Definition 42. Assume the end path $m_1, \ldots, m_{2p}$ is redundant in $\mathsf{G}(t,E)$. If t' is the subterm whose root is the successor node of m_{2p} then *transformation* T_2 is $t[t'/m_1]$.

This transformation disposes of the path $m_1, \ldots, m_{2p}$ in t (and all paths below m_i that do not pass through m_{2p}) by replacing the subtree at m_1 with t' that is rooted at the successor of 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. Application of this transformation reduces the size of a term.

Proposition 31. If t solves E and tT_2t' then t' solves E.

Proof. Assume that t solves E, the end path $m_1, \ldots, m_{2p}$ is redundant in $\mathsf{G}(t,E)$, t' is the subtree rooted at the successor of m_{2p} and $t_1 = t[t'/m_1]$. We show that t_1 solves E. First, t_1 is a closed term with the same type as t: by the definition of end path, if node m occurs below m_{2p} in t and $n \downarrow m$ then either n occurs above m_1 or below m_{2p}. We next show that $\forall$ loses the game $\mathsf{G}(t_1,E)$. We assume that the nodes in t_1 retain the same names as in t. Consider any play $\pi \in \mathsf{G}(t,E)$. Assume π_{i_1} is the first position in π at m_1, if there is one. By Definition 41, there is a later position π_{k_1} at m_{2p} and a chain $\pi_{j_1}, \ldots, \pi_{j_{2p}}$ for π_{k_1} where $j_1 = i_1$. Therefore, by repeated application of Definition 41, π can be decomposed into subintervals $\sigma_1, \pi[i_1,k_1], \sigma_2, \ldots, \pi[i_q,k_q], \sigma_{q+1}$ for $q \geq 0$, where each σ_j is an interval, π_{i_r} is the first position in π after $\pi_{k_{r-1}}$ at m_1 and π_{k_r} is at m_{2p} (and there is a chain for π_{k_r} that starts at π_{i_r}). No interval $\pi[i_r,k_r]$ contributes and no position in any interval σ_j is at m_1.

Next, we show that if there is a position in some σ_l that is below m_1 then it is below m_{2p}. Assume not. Let the earliest such position be π_j which is at a lambda node. There is an earlier position π_i that is the parent of π_j. Position π_i belongs to an interval $\pi[i_s,k_s]$ and n_i is a variable node: if π_i does not belong to such an interval then $n_i - 1$ is below m_1 and π_{i-1} is in σ_l for some l which is a contradiction as $i-1 < j$; if n_i is a constant node then $j = i+1$ and so this contradicts that π_j is in σ_l. Therefore, there is the chain $\pi_{j_1}, \ldots, \pi_{j_{2p}}$ where $i_s = j_1$ and $k_s = j_{2p}$. So, π_i belongs to one of the intervals $\pi[j_{2m-1}, j_{2m}]$. Consider position π_{i+1} which is in some w_q such that $\theta_{i+1} = \theta_l\{((n'_1, \ldots, n'_r), \theta_i)/n_{i+1}\}$ where $n_j = n'_s$ for some s. It cannot be the case that $j_{2m-1} < i < j_{2m}$. because either $\theta_{j_{2m}} = \theta_{j_{2m-1}}$ or $\theta_{j_{2m}} = \theta_{j_{2m-1}}\{((m'_1, \ldots, m'_s), \theta_k)/n_{j_{2m}}\}$ where $k < j_{2m-1}$ by Propositions 12 and 18; so, there is no way to access the binder for n_{i+1} after position j_{2m} which contradicts that π_j occurs later. Therefore, $i = j_{2m-1}$ and so $\pi_{j_{2m}}$ and π_j are children of π_i. Consider the chain $\pi_{j_{2m-1}}, \ldots, \pi_{j_{2p}}$ where $\pi_{j_{2p}} = \pi_{k_s}$. By Proposition 18, $\theta_{j_{2m-1}} \leq \ldots \leq \theta_{j_{2p}}$. Now, we employ a similar argument to above to show that the only access to the binder n_{i+1} is through a descendant under binding of a node at a position in this chain which is impossible because it is an end path. Formally, one can proceed by induction on the number of positions in the interval $\pi[j', j]$ when $n_{j'}$ is a variable or constant node of t and $j > j' \geq k_s + 1$. Without loss of generality we just examine the case $j' = k_s + 1$. If $n_{j'}$ is a constant node the result follows from the interval $\pi[j' + 2, j]$. If n'_j is a variable node then n'_{j+1} is a lambda node of some w_q and $\theta_{j'+1} = \theta_r\{((m'_1, \ldots, m'_s), \theta_{j'})/n'_{j+1}\}$ where $r < j_1$ as $n_{j_1}, \ldots, n_{j_{2p}}$ is an end path and $m'_1, \ldots, m'_s$ are the successors of n'_j. Now the only access to the entry for n_{i+1} is through a child of $\pi_{j'+1}$, say π_r, and so the result follows from the interval $\pi[r + 2, j]$.

Assume m_1 is the successor of m and n is the successor of m_{2p}; so, in t_1 n is the successor of m. It follows that $\pi' = \sigma_1 \ldots \sigma_{q+1}$ with the following minor change to look-up tables is a play of $\mathsf{G}(t_1, E)$; for any position π'_i of π' where $n'_i = m$ and so n'_{i+1} is n the look-up table θ'_{i+1} is changed by removing all entries for binders that occur in the end path and this is maintained for subsequent positions.

Conversely, assume that $\pi' \in \mathsf{G}(t_1, E)$. Then by almost identical reasoning to that above there is a play $\pi \in \mathsf{G}(t, E)$ such that π' can be decomposed into intervals $\sigma_1, \ldots, \sigma_{q+1}$ and $\pi = \sigma_1, \pi[i_1, k_1], \sigma_2, \ldots, \pi[i_q, k_q], \sigma_{q+1}$ where, for each $\pi[i_s, k_s]$ there is a chain $\pi_{j_1}, \ldots, \pi_{j_{2p}}$, $i_s = j_1$, $k_s = j_{2p}$, and for each l, $n_{j_l} = m_{j_l}$; and the look-up tables in π are amended to include the binders in the end path; moreover, no $\pi[i_s, j_s]$ contributes. □

Consider Figure 6 with the single play of Figure 7. The sequence of nodes (8), (11) is an end path. It is also redundant in its only play; 11 is the first position at (8) and then its child 14 is at (11); later, position 33 is at (8) and its child position 36 is at (11); neither $\pi[11, 14]$ nor $\pi[33, 36]$ contributes. Therefore, replacing node (8) with the tree of the subterm zs_1s_2 rooted at (12) is a smaller term that solves the equation of Example 12.

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 37). However, using Fact 27, they do provide immediate bounds on how many times an embedded node with the same label can occur in a path of a smallest solution term.

Proposition 32. Assume t is a smallest solution to E whose goal is u, $m_1, \ldots, m_p$ are embedded end nodes of t labelled with the same variable, there is an n such that $n \downarrow m_i$ and m_{i+1} occurs below m_i for each i. Then $p \leq \text{depth}(u)$.

Proof. The proof uses Fact 27. Consider m_p; if it is below the lth successor of m_{p-1} then $m_p m$ where m is the lth successor of m_p is an end path. If this path is redundant for every play in $\mathsf{G}(t, E)$ then T_2 applies to t which contradicts that it is a smallest solution term to E. Otherwise, there is an interval $\pi[i_p, j_p]$ such that $m_p = i_p - 1$ and $\pi[i_p, j_p + 1]$ contributes, and $p - 1$ earlier complementary intervals $\pi[i_k, j_k]$ with $m_k = i_k - 1$; if $p > \text{depth}(u)$ then this contradicts that t solves E. $\square$

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*

Transformations T_1 and T_2 apply to potential solution terms of interpolation equations. We shall introduce a third transformation that works on a *different* tree representation of an interpolation equation that is built out of partitioning of plays and which uses Definition 37.

Definition 43. Assume $\pi \in \mathsf{G}(t, E)$ and t solves E. We define a *partition* of π into stages as follows. Stage 1 consists of $\pi[1, 2]$. Assume stage N finishes at a lambda node of t at position π_{i-1}; stage $N + 1$ is defined by cases on n_i.

1. constant node: $\pi[i, i + 1]$,
2. variable node: if $\pi_{j_1}, \ldots, \pi_{j_{2p}}$, $p \geq 0$, is the longest chain with the recurrence property where $\pi_{j_1} = \pi_{i+2}$ such that $\pi[j_1, j_{2p}]$ does not contain a child of π_i then $\pi[i, j]$ where $j = i + 3$ if $p = 0$ and $j_{2p} + 2$ otherwise.

The first stage $\pi[1, 2]$ of the partition of π finishes at the root of the solution term t. Every stage except the final stage finishes at a lambda node of t. Stage $N+1$ starts at the successor node of where stage N finishes; assume π_i is this position. If n_i is a constant node then this stage is $\pi[i, i+1]$ which is the final stage if the constant is $a : \mathbf{0}$; otherwise n_{i+1} is a successor of n_i. When n_i is a variable node, play jumps to a lambda node n_{i+1} of w_q for some q. By Proposition 29 there is a largest $r \geq 0$ such that $\pi_{j_1}, \ldots, \pi_{j_{2r}}$ is a chain with the recurrence property when $\pi_{j_1} = \pi_{i+2}$: if for $p < r$, $\pi_{j_{2p}+2}$ is a child of π_i then we take this subchain or let $p = r$. If $p = 0$ then this stage is $\pi[i, i+3]$ and otherwise $\pi[i, j_{2p} + 2]$: the partition can only finish in w_q at a constant $a : \mathbf{0}$ when the stage is final; otherwise, it must finish at a lambda node of t.

Example 14. Consider the partition of the single play π of Figure 7 on the interpolation tree of Figure 6. There are nine stages $S_1 - S_9$ as follows.

S_1	$\pi[1, 2]$	S_2	$\pi[3, 6]$	S_3	$\pi[7, 10]$
S_4	$\pi[11, 14]$	S_5	$\pi[15, 20]$	S_6	$\pi[21, 24]$
S_7	$\pi[25, 40]$	S_8	$\pi[41, 44]$	S_9	$\pi[45, 48]$

S_3 starts at the level 1 node (4) at position 7, jumps to (17) the root of w_1: the chain in this case is when $p = 0$ as play descends to (8) and at the next position 10 is a child of position 7. This is similar for S_2 and S_4. Stage S_5 starts at position 15 and finishes at a child π_{20} of it: in this case, $p = 1$ as there is the chain π_{17}, π_{18} where n_{17} is the constant node (32). The most interesting stage is S_7: it starts at node (6) and jumps to (25) in w_2: again, $p = 1$ with the chain π_{27}, π_{38}; n_{27} is the (embedded) node (26) and $\pi[28, 37]$ corresponds to the earlier $\pi[6, 15]$ (and π_{38} is a child of π_{27}). Stage S_9 is final. Both stages S_6 and S_8 finish at the same node (and, so, stages S_7 and S_9 start at the same node). □

The following is a direct consequence of Proposition 29.

Fact 33. If $\pi \in \mathsf{G}(t, E)$ and t solves E then there is a unique partition of π.

The following uses Definition 40.

Definition 44. Assume $\pi[i, j]$ is stage N of the partition of $\pi \in \mathsf{G}(t, E)$. Stage N is *final* if $[r_j] = [\exists]$. If stage N is not final then it *contributes* (to the solution) when $\pi[i, j]$ contributes. If stage N is final then it *contributes* when $\pi[i, j - 1]$ contributes.

A stage contributes to the solution if it contains a position that is at a node labelled with a constant f with arity > 0. In any play, the number of stages of its partition that contribute to a solution is at most depth(u), where u is the goal term. In the case of Example 14, 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.

Proposition 34. Assume $\pi[i, j]$ 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 level 1 or a constant node then π_j is a child of π_i.
3. If stage N does not contribute and n_i is embedded then π_j is a child of π_i.
4. If stage N is not final and n_i is a variable node then n_j is a successor of a node in the same family as n_i.

Proof. Part 1 is clear since t solves E. For 2, assume stage N is not final. If n_i is a constant node then $j = i+1$ and so π_j is a child of π_i. If n_i is level 1 then n_{i+1} is the root node of some w_q: consider the chain that determines stage N, $\pi_{j_1}, \ldots, \pi_{j_{2p}}$, $p \geq 0$ where $\pi_{j_1} = \pi_{i+2}$. By definition $\pi[j_1, j_{2p}]$ does not contain a child of π_i; therefore, every node $n_{j_{2m-1}}$ is a constant node and because stage N is not final, it follows that $n_{i+1} \downarrow n_{j_{2p}+1}$ and so, π_j is a child of π_i. For 3 assume stage N does not contribute; the result now follows from Fact 27. For 4 assume that stage N is not final, n_i is a variable node and $\pi_{j_1}, \ldots, \pi_{j_{2p}}$, $p \geq 0$, is the chain in w_q that determines this stage. Now we make use of Proposition 28 as follows. We extend this chain so that $\pi_{l_1}, \ldots, \pi_{l_{2r}}, \pi_{j_1}, \ldots, \pi_{j_{2p}}$ is the chain that starts at the successor of the root node of w_q; so, n_{l_1-1} is the root node of w_q and one of the positions $\pi_{l_1-1}, \pi_{l_2}, \ldots, \pi_{j_{2p}}$, say π_k, is the parent of π_{j-2}; so, n_j is a successor of a node in the same family as n_i, as it is a successor of n_{k-1}. □

S_2, S_3 and S_4 of the partition in Example 14 on the tree in Figure 6 start at level 1 nodes and finish with a child position at some successor. 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) labelled z_2; node (2) is the common ancestor of (6) and (12).

Now we will build the partition tree for $\mathsf{G}(t, E)$.

Definition 45. Assume $\pi[i, j]$ is stage N of the partition of $\pi \in \mathsf{G}(t, E)$.

1. The *comb* at stage N of π is the node n_i of t and its successors $n_i1, \ldots, n_ip$ for $p \geq 0$.
2. The *path* at stage N of π is defined by cases on n_i: if n_i is a constant node or labelled @ then it is ε (the empty sequence); if n_i is a variable node and $\pi_{j_1}, \ldots, \pi_{j_{2p}}$, $p \geq 0$, is the chain for π_{j-2} where $j_1 = i + 2$ then it is the path $n_{i+1}, n_{j_1}, \ldots, n_{j_{2p}}, n_{j-1}$ in some w_q.

In the case of S_3 in Example 14 on the tree in Figure 6 the comb is the node (4) and its successors (5) and (7) and the path is (17), (18) of the first argument w_1. The comb for S_5 is node (12) and its successors (13) and (15); its path is (31), (32), (33), (34) in the second argument w_2. In the case of S_7, its comb is the singleton node (6) and its path is (25), (26), (29), (30) of w_2.

We transfer notations from nodes to stages: two stages belong to the same family if the root nodes of their combs do; a stage is embedded if the root node of its comb is and so on for a constant stage or a level k stage. We also write $M < N$ to mean M is an earlier stage than N (in the partition of π).

Nodes of the partition tree are pairs combs and paths at stages of the partitions of plays. The root is the comb at @ and the empty path. We assume an edge $\xrightarrow{\pi}$ from node at stage $N+1$ of π to the lambda node of the comb at an earlier stage where stage N finishes. For each play π, there is an edge from stage 2 to the first successor of the comb at stage 1. In this way the term t can be recovered from the tree when $\xrightarrow{\pi}$ is a successor on combs (when the comb at stage $N+1$ is placed beneath the lambda node of the earlier stage). In more detail, assume that $\pi[i, j]$ is stage N of the partition of π, stage N is not final and consider its comb and path. If n_i is a constant node then there is an edge from stage $N + 1$ to n_j of the comb at stage N: to avoid heavy notation we write $N+1 \xrightarrow{\pi} N$. If n_i is a variable node then we examine its path $n_{i+1}, n_{j_1}, \ldots, n_{j_{2p}}, n_{j-1}$ at stage N which is in some w_q: we extend this if necessary to include the binder of n_{j-1} (by extending the chain and possibly including the root node of w_q). Stage N, by Proposition 34, finishes at a successor of a node in the same family as n_i: it is, therefore, at a node of a comb of one of the following stages $\{K \mid K \leq N, \exists M.N \xrightarrow{\pi}_* M, K \xrightarrow{\pi}_* M$ and K, M belong to the same family$\}$ (where $\xrightarrow{\pi}_*$ is the reflexive and transitive closure of $\xrightarrow{\pi}$); if K is this element then $N + 1 \xrightarrow{\pi} K$. A simple consequence of Proposition 34 is that $N + 1 \xrightarrow{\pi} N$ if N is a constant, level 1 or embedded stage.

The partition tree for Example 14 is in Figure 12 (where we omit the index π from $\xrightarrow{\pi}$). Both stages S_6 and S_8 finish at node (5) of the comb at stage (3). When $\longrightarrow$ is viewed as the successor relation then the combs of this partition tree form the solution term in Figure 6: for instance, the comb of S_4 is beneath node (7) of the comb at S_3 and the comb of S_7 (and S_9) is beneath node (5) (of the comb at node (3)).

The following is a simple consequence of Definition 43.

$$\begin{tikzcd}
 & & S_7 \arrow[drrrr, bend left] & & & & & & \\
 & & & S_6 \arrow[r] & S_5 \arrow[r] & S_4 \arrow[r] & S_3 \arrow[r] & S_2 \arrow[r] & S_1 \\
 & S_8 \arrow[urrr, bend right] & & & & & & & \\
S_9 \arrow[uurrrrrr, bend right] & & & & & & & &
\end{tikzcd}$$

Fig. 12. Partition tree for Example 14

Fact 35. Assume $\pi[i, j]$ is stage N of the partition of $\pi \in \mathsf{G}(t, E)$ which contributes. Then one of the following is true.

1. N is a constant stage,
2. the path at N contains a constant node,
3. there is $k : i < k < j$, and a stage $K < N$ in the same family as N such that n_k is a constant node and n_k occurs in K.

6.4. *Unfolding*

In this section we complete the proof of decidability of matching by introducing a transformation that applies to individual stages of a partition tree: we call it "unfolding" as it is reminiscent of unravelling a model as in modal logic.

Given a partition tree, we define compositions of combs and paths to form larger units: compositions of combs (unfolding a comb into others) produce larger subterms; compositions of paths produce trees (subtrees of some w_q). To give the reader a clear idea we first present an example.

Consider the partition tree of Figure 12 for Example 14 for the play of Figure 7 on the tree in Figure 6. S_5 contributes and S_2 is in the same family. However, S_3 and its descendants under binding, S_6 and S_8, do not contribute. So, we can unfold S_3 into S_6 and S_3 into S_8. These are pictured in Figure 13. The unfolding of S_3 into S_6 places the comb at S_6 directly beneath node (7) of the comb of S_3: we also include the subtree which is the composition of the paths of S_3 and S_6 next to it: the path for S_3 is (17), (18) and (21), (22) for S_6; it is slightly different for S_3 and S_8.

The idea is to replace stage S_6 with the new S'_6, S_3 unfolded into S_6, whose enlarged comb and subtree are depicted in Figure 13; similarly, S_8 is replaced by S'_8, the unfolding of S_3 into S_8. Both S'_6 and S'_8 start at the root of their enlarged combs and finish at node (5); their associated trees show that the new stages consist of appending S_6 and S_8 to S_3. So, the new partition tree is as in Figure 14. Although the initial solution tree has increased in size because of the enlarged combs at S'_6 and S'_8 bindings are *more local*; so, the first occurrence of node (7) in S_3 no longer binds (14) and (16). Now we can apply transformation T_2 of Section 6.2 to the expanded tree; in fact, before unfolding, we could

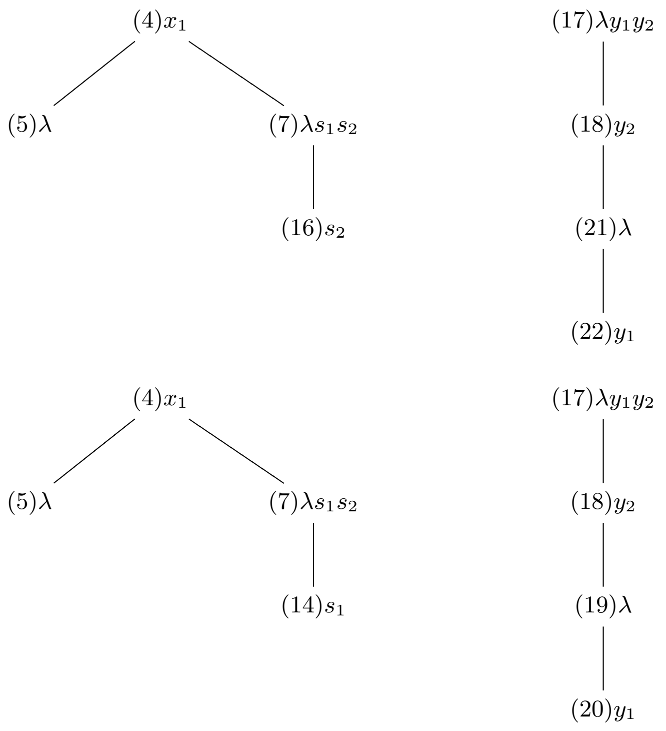

Fig. 13. Unfolding of S_3 into S_6, S_3 into S_8 and their trees

$$\begin{tikzcd}
 & & S_7 \arrow[r] & S_6' \arrow[r] & S_5 \arrow[r] & S_4 \arrow[r] & S_3 \arrow[r] & S_2 \arrow[r] & S_1 \\
S_9 \arrow[r] & S_8' \arrow[urrr, bend right] & & & & & & &
\end{tikzcd}$$

Fig. 14. Unfolded partition tree of Figure 12

have omitted the comb at S_4 so node (12) would be directly below (7) because the end path (8), (11) is redundant as described in Section 6.2. T_2 now applies to the combs at S_3, S_6' and S_8' as they now involve redundant end paths: the path (4), (7) in S_3, and (4), (5) in both S_6' and S_8'. Therefore, the resulting term after these applications of T_2, in Figure 15, is a “small” solution to the interpolation equation.

Definition 46. Assume N_i, $1 \leq i \leq k$, are stages of the partition of $\pi \in \mathsf{G}(t, E)$ where t_i is the comb at N_i and p_i is the path at stage i and which obey the following conditions.

(a) $N_1, \ldots, N_k$ belong to the same family, do not contribute and $N_i < N_j$, $i \leq j$,
(b) for all $1 \leq i < j \leq k$, $t_i \neq t_j$ and $p_i \neq p_j$,
(c) for all $j \leq k$, there is an $i \leq j$ such that $N_j + 1 \xrightarrow{\pi} N_i$, except for $j = k$ if N_k is final for π,

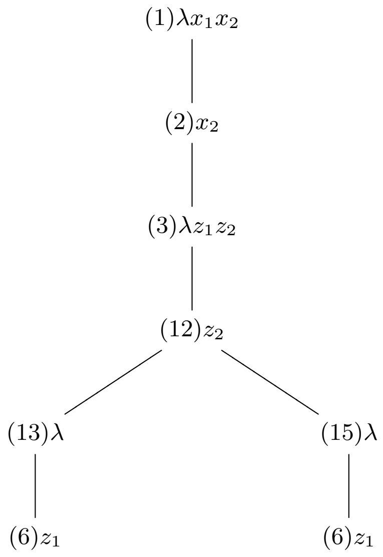

Fig. 15. Reduced solution term for Example 14

(d) for all $j \leq k$ there is an $i < j$ such that the root of p_j is a successor of a leaf node of p_i.

We iteratively define N_1 *unfolded* into $N_2, \ldots, N_j$ for $j \leq k$ as $(N_1 \ldots N_j)$ with *enlarged comb* $(t_1 \ldots t_j)$ and *tree* $(p_1 \ldots p_j)$ as follows.

1. $(N_1) = N_1$, $(t_1) = t_1$ and $(p_1) = p_1$,
2. Assume $(N_1 \ldots N_j)$ with $(t_1 \ldots t_j)$ and $(p_1 \ldots p_j)$. t_{j+1} is placed beneath the leaf of $(t_1 \ldots t_j)$ where N_j finishes to form $(t_1 \ldots t_{j+1})$; and p_{j+1} is placed below the leaf of $(p_1 \ldots p_j)$ so it is still a subtree of some w_q to form $(p_1 \ldots p_{j+1})$.

The *play of* $(N_1 \ldots N_j)$, the *new stage* N'_j for π, starts from the root of $(t_1 \ldots t_j)$ and finishes at a leaf of $(t_1 \ldots t_j)$.

Many assumptions are needed to define unfolding of stage N_1 into stages $N_2, \ldots, N_k$ of a play π. The stages belong to the same family, do not contribute and have distinct combs and paths. Each stage N_j finishes at a node of a comb of some N_i, $i \leq j$, (except N_k if it is final for π). The root of each p_j is below a leaf of some p_i, $i < j$. Therefore, we can iteratively compose the combs and paths into subtrees: in the latter case they are expanding subtrees of some w_q. The examples above of S_3 unfolded into S_6 and S_3 into S_8 are instances of this definition. The following is clear from the definition of unfolding.

Fact 36. Assume $N_1, \ldots, N_k$ are stages of the partition of $\pi \in \mathsf{G}(t, E)$ and N_1 is unfolded into $N_2, \ldots, N_k$. For each $1 \leq j \leq k$, the play of $(N_1 \ldots N_j)$ does not contribute, and finishes at node n of t_i in $(t_1 \ldots t_j)$ iff stage N_j finishes at n of t_i.

To finish the proof of decidability of matching, we show that given a large solution term to an equation such that transformations T_1 and T_2 do not apply to it, we can find a smaller solution term: first, the partition tree is built and then unfolded and then transformations T_1 and T_2 are applied. Unfolding happens with embedded combs; we need to generalise this notion to larger combs.

Definition 47. The comb $(t_1 \dots t_k)$ is *embedded* if its only free variable occurrence is that of the head of t_1 and there is a comb above it in the partition tree $(s_1 \dots s_k)$ which is α-equivalent to it.

Therefore if $(t_1 \dots t_k)$ is embedded each variable node in t_j, $j > 1$, is bound by a node within some t_i, $i < j$. We extend this notation to stage $(N_1 \dots N_k)$.

The central theorem is a small solution property (see Definitions 2, 8 and 11 for arity and depth and where $|t|$ is the size, the number of nodes, of 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 34. Therefore, there is a tree automaton such that A_E is non-empty iff E has a (canonical) solution.

Theorem 37. If E, $xw_1 \dots w_k = u$, 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 $4(n^2 m^{2n-2}|u|)$.

Proof. Assume E, $xw_1 \dots w_k = u$, 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 the number of branches in (u) which is bounded by $|u|$. Term t can contain nodes whose level is at most n. In any path of t, the number of occurrences of constants whose arity is more than 0 is bounded by depth(u); otherwise transformation T_1 of Section 6.2 would apply to t and produce a smaller solution term. Also T_2 does not apply for the same reason. Consider the partition tree for the game $\mathsf{G}(t, E)$. Each play π has the same first stage and shares the same comb at stage 2. Stage N is a *separator* if there are two different plays that share the same combs before and including N, and finish stage N at different nodes.

In the partition tree we apply the unfolding wherever possible starting with a stage N_1 that has no descendants under binding that contribute. In particular, a suitable N_1 is level 1 or embedded. If in a path of t there are at least $(k+1)m^j$ level j nodes then at least k are embedded and labelled with the same variable: if they are also end then $k \leq$ depth(u) by Proposition 32. If $k > m$ then at least one, say N_1, can be unfolded into N_2 such that the enlarged comb $(N_1 N_2)$ is embedded. We now repeat this argument for $k > m^l$; N_1 can be unfolded into the embedded $(N_2 \dots N_j)$ for $j \leq l+1$. The maximum depth of an enlarged comb 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). Clearly, if stage N of π contributes then it still contributes after all unfoldings; and no new unfolded stage can contribute by construction. 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 number of stages that can contribute is at most $|u|$ and therefore the number of later stages that start from a node that is a descendant under binding of a node of a contributing stage and that is not embedded is at most nm^{n-1}. We now consider for which stages is T_2 not applicable. These are the stages that contribute and those later stages that start from descendant nodes under binding and that are not embedded. There is also the issue that in the partition tree the same node may have more than one edge entering it from two different later stages of the same or different plays; we need to guarantee the same subterm is rooted at those later stages. To do this, we may need extra later stages that are also play separators; the number of such separators is bounded by $nm^{n-1}|u|$ which is at most the number of stages that either contribute or start from a descendant under binding that is not embedded. This, therefore, gives an upper bound of $4(n^2m^{2n-2}|u|)$ on the depth of t. □

The bound in Theorem 37 is very crude. For instance, for Example 12 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.

6.5. *Removing restrictions*

The decidability proof presented is for the restricted case when there is a single base type and when the goal term u of an interpolation equation does not contain bound variables. We now describe how the proof can be easily extended to the unrestricted case.

Assume that there is a family $\mathbb{B}$ of base types. The sets of terms, lnfs, matching equations and interpolation equations carry over to this more general setting: for instance, an interpolation equation $xw_1 \dots w_k = u$ now has the type $(A_1, \dots, A_k, B)$ and $u : B$ where $B \in \mathbb{B}$. We extend the tree representation of terms and interpolation equations to include at each node its *result* type: if the node is a variable, constant or lambda node of type $(A_1, \dots, A_l, B)$ or B then B is its result type; in the case of Figure 6 this would mean including $\mathbf{0}$ at each node. Consider consecutive positions of a play on such a tree: if a position is at a lambda node with result type B then its successor node must also have result type B; if a postion is at a variable node with result type B then the next position (involving a jump) must be at a lambda node with result type B; therefore, a change in result type can only occur at a constant node. Therefore, in any play there can only be at most depth(u) changes in the result type. Furthermore, this is also constains the number of times that there can be a change in the result types in a path of a smallest solution term t. Now the ingredients of the proof are easily generalised: for transformation T_1 we assume a constant $b_B : B$ for each $B \in \mathbb{B}$ and for transformation T_2 we assume that each node in an end path has the same result type; a similar restriction applies to the unfolding. The overall effect on Theorem 37 is to replace $|u|$ by $|u|^2$ in the bound to take account of changes in result types.

Next, the case when the goal term u can contain bound variables (also allowed for in (Stirling 2009A)). This requires a change in the definition of the interpolation game. We extend the lok-up tables to include direct entries for binders that are successors of decendants under binding of constant nodes: an entry for such a binder labelled $\lambda v_1 \dots v_p$ has the form $(c_1, \dots, c_p)$ where each c_i is a fresh constant that does not occur in the solution term or in the interpolation equation and which has the same type as v_i. Next, we need to change the rule for a constant f and add rules of Figure 5 for the next position in a play; the new rules are presented in Figure 16. In effect, nodes bound by a binder

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

— $f : (B_1, \ldots, B_p, \mathbf{0})$ if $r \neq fr_1 \ldots r_p$ then $n[\forall]\theta$ else $\forall$ chooses $j \in \{1, \ldots, p\}$ and
  - $r_j : \mathbf{0}$ then $nj\,[r_j]\,\theta$,
  - $r_j = \lambda v_1 \ldots v_m.r'$ then $nj\,[r'\{c_1/v_1, \ldots, c_m/v_m\}]\,\theta\{(c_1, \ldots, c_m)/nj\}$.

— $v_i : \mathbf{0}$, $m \downarrow n$ and $\theta(m) = (c_1, \ldots, c_l)$; if $r = c_i$ then $n\,[\,\exists\,]\,\theta$ else $n\,[\,\forall\,]\,\theta$.

— $v_i : (B_1, \ldots, B_p, \mathbf{0})$, $m \downarrow n$ and $\theta(m) = (c_1', \ldots, c_l')$; if $r \neq c_i' r_1 \ldots r_p$ then $n\,[\forall]\,\theta$ else $\forall$ chooses $j \in \{1, \ldots, p\}$ and
  - $r_j : \mathbf{0}$ then $nj\,[r_j]\,\theta$,
  - $r_j = \lambda s_1 \ldots s_m.r'$ then $nj\,[r'\{c_1/s_1, \ldots, c_m/s_m\}]\,\theta\{(c_1, \ldots, c_m)/nj\}$.

Fig. 16. Amended game moves

that is a successor of a constant or of a descendant under binding of a constant in a term t or a term w_i are really constant nodes: these are the nodes labelled with a variable v_j; the rules associate constants c_j with their binder; moreover in the goal subterm the constant c_j is substituted for the bound variable s_j. Theorem 13 holds for the new game. The decidability proof then proceeds as before: the definitions of the transformations and unfolding remain unchanged.

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.

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. In *Logic, Meaning and Computation*, Synthese Library, 305, 331-342, (2002).

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-Schauß, 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).

Vorobyov, S. The “hardest” natural decidable theory. *Procs LICS 1997*, 294-305, (1997).

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).