A Game-Theoretic Approach to Deciding Higher-Order Matching

Citation for published version:

Digital Object Identifier (DOI):
10.1007/11787006_30

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Automata, Languages and Programming

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 10. Nov. 2018
A Game-Theoretic Approach to Deciding
Higher-Order Matching

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

Abstract. We sketch a proof using a game-theoretic argument that the higher-order matching problem is decidable.

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

1 Introduction

Higher-order unification is given an equation \( t = u \) containing free variables, is there a solution substitution \( \theta \) such that \( t\theta \) and \( u\theta \) have the same normal form? Terms \( t \) and \( u \) are from the simply typed \( \lambda \)-calculus and the same normal form is \( \beta\eta \)-equality. Higher-order matching is the particular instance when the term \( u \) is closed, can \( t \) be pattern matched to \( u \)? Although higher-order unification is undecidable, higher-order matching was conjectured to be decidable by Huet [4] (and, if so then it has non-elementary complexity [11, 13]). Decidability has been proved for the general problem up to order 4 and for various special cases [7-10, 2]. Loader showed that matching is undecidable for the variant definition when \( \beta \)-equality is the same normal form [5].

We propose a game-theoretic technique that leads to decidability of matching. It starts with Padovani’s reduction to the dual interpolation problem [8]. We then define a game on a closed \( \lambda \)-term \( t \) where play moves around it relative to a dual interpolation problem. The game captures the dynamics of \( \beta \)-reduction on \( t \) without changing it (using substitution). Small pieces of a solution term, that we call “tiles”, can be classified according to their subplays and how they, thereby, contribute to solving it. Two transformations that preserve solution terms are introduced. With these, we show that 3rd-order matching is decidable via the small model property: if there is a solution to a problem then there is a small solution to it. For the general case, the key idea is “tile lowering”, copying regions of a term down its branches. A systematic method for tile lowering uses unfolding which is similar to unravelling a model in modal logic. Unfolding requires a non-standard interpretation of game playing where regions of a term are to be understood using suffix subplays. At this point, we step outside terms of typed \( \lambda \)-calculus. Refolding returns us to such terms. The detailed proof of decidability uses unfolding followed by refolding and from their combinatorial properties the small model property follows. However, here we can only outline the method with an example. For all the details and proofs, the reader is invited to access “Decidability of higher-order matching” from the author’s web page.
2 Matching and dual interpolation

Assume simply typed 𝜆-calculus with base type 0 and the definitions of α-equivalence, β and 𝜈-reduction. A type is 0, with order 1, or A₁ → ... → Aₙ → 0, with order k + 1 where k is the maximum of the orders of the Aᵢ’s. Assume a countable set of typed variables x, y, ... and typed constants, a, f, .... The simply typed terms is the smallest set T such that if x (f) has type A then x : A ∈ T (f : A ∈ T); if t : B ∈ T and x : A ∈ T, then 𝜆x.t : A → B ∈ T; if t : A → B ∈ T and u : A ∈ T then tu : B ∈ T. The order of a term is the order of its type and it is closed if it does not contain free variables.

A matching problem is v = u where v, u : 0 and u is closed. The order is the maximum of the orders of the free variables x₁, ..., xₙ in v. A solution is a sequence of terms t₁, ..., tₙ such that v{t₁/x₁, ..., tₙ/xₙ} = 𝜈 u. Given a matching problem the decision question is, does it have a solution?

We slightly change the syntax of types and terms. A₁ → ... → Aₙ → 0 is rewritten (A₁, ..., Aₙ) → 0 and all terms in normal form are in 𝜈-long form: if t : 0 then it is u : 0 where u is a constant or a variable, or u(t₁, ..., tₖ) where u : (B₁, ..., Bₖ) → 0 is a constant or a variable and each tᵢ : Bᵢ is in 𝜈-long form; if t : (A₁, ..., Aₙ) → 0 then t is 𝜆y₁ ... yₙ.t’ where each yᵢ : Aᵢ and t’ : 0 is in 𝜈-long form. A term is well-named if each occurrence of a variable y within a 𝜆-abstraction is unique.

Definition 1. Assume u : 0 and vᵢ : Aᵢ, 1 ≤ i ≤ n, are closed terms in normal form and x : (A₁, ..., Aₙ) → 0. x(v₁, ..., vₙ) = u (≠ u) is an interpolation equation (disequation). A dual interpolation problem P is a finite family of interpolation equations and disequations, i : 1 ≤ i ≤ n, x(vᵢ₁, ..., vᵢₙ) ≈ i uᵢ, with the same free variable x and each ≈ i ∈ {=, ≠}. The type and order of P are the type and order of x. A solution of P of type A is a closed term t : A in normal form, such that for each equation t(vᵢ₁, ..., vᵢₙ) = β uᵢ and for each disequation t(vᵢ₁, ..., vᵢₙ) ≠ β uᵢ. We abbreviate t solves P to t |= P.

Padovani shows that a matching problem of order n reduces to a dual interpolation problem of the same order [8]: given P, is there a solution t |= P? We assume a fixed dual interpolation problem P of type A whose order is greater than 1 (as an order 1 problem is easily decided) where the normal form terms vᵢ are well-named and no pair share bound variables.

A right term u of a (dis)equation may contain bound variables. If X = {x₁, ..., xₖ} are its bound variables then let C = {c₁, ..., cₖ} be a fresh set of constants with corresponding types. The ground closure of w with bound variables in X, with respect to C, Cl(w, X, C), is: if w = a : 0, then Cl(w, X, C) = {a}; if w = f(w₁, ..., wₙ), then Cl(w, X, C) = {w} ∪ \bigcup Cl(wᵢ, X, C); if w = 𝜆xᵢ ... xᵣ, u then Cl(w, X, C) = Cl(u{cᵢ/xᵢ, ..., cᵣ/xᵣ}, X, C). For u = f(λx₁x₂x₃, x₁(x₂), a) with respect to {c₁, c₂, c₃}, it is {u, c₁(c₂), c₃, a}.

We also identify subterms of left terms vᵢ of a (dis)equation relative to a set C: however, these need not be of ground type and may also contain free variables. The subterms of w relative to C, Sub(w, C), is defined using
an auxiliary set \( \text{Sub}'(w, C) \); if \( w \) is a variable or a constant, then \( \text{Sub}(w, C) = \text{Sub}'(w, C) = \{ w \} \); if \( w = x(w_1, \ldots, w_n) \) then \( \text{Sub}(w, C) = \text{Sub}'(w, C) = \{ w \} \cup \bigcup \text{Sub}(w_i, C) \); if \( w = f(w_1, \ldots, w_m) \), then \( \text{Sub}(w, C) = \text{Sub}'(w, C) = \{ w \} \cup \bigcup \text{Sub}'(w_i, C) \); if \( w = \lambda y_1 \ldots y_n. v \), then \( \text{Sub}(w, C) = \{ w \} \cup \text{Sub}(v, C) \) and \( \bigcup \{ \text{Sub}(v[c_1/y_1, \ldots, c_n/y_n], C) : c_i \in C \} \) is the set \( \text{Sub}'(w, C) \). If \( v = \lambda z. f(\lambda z_1 z_2 z_3 z_4(z_2), z) \) and \( z_2, z_3 : 0 \) then \( \text{Sub}(v, \{ c_1, c_2, c_3 \}) \) is \( \{ v, f(\lambda z_1 z_2 z_3 z_4(z_2), z), c_1(c_2), c_2(c_3), c_2(c_3, z) \} \).

Given the problem \( P \), let \( X_i \) be the (possibly empty) set of bound variables in \( u_i \) and let \( C_i \) be a corresponding set of new constants (that do not occur in \( P \)), the forbidden constants.

**Definition 2.** Assume \( P \) is the fixed problem of type \( A \). \( T \) is the set of subtypes of \( A \) including \( A \) and subterms of \( u_i \). For \( i \), the right subterms are \( R_i = \text{Cl}(u_i, X_i, C_i) \) and \( R = \bigcup R_i \). For \( i \), the left subterms are \( L_i = \bigcup \text{Sub}(v_i, C_i) \cup C_i \) and \( L = \bigcup L_i \). The arity, \( a_i \), of \( P \) is the largest \( k \) where \( (A_1, \ldots, A_k) \to B \in T \). The right size \( \delta(u) \) relative to \( C \) is: if \( u = a : 0 \) then \( \delta(u) = 0 \); if \( u = f(w_1, \ldots, w_n) \) then \( \delta(u) = 1 + \sum \delta(w_i) \); if \( u = \lambda x_1 \ldots x_n. w \), then \( \delta(u) = \delta(w.[c_1/x_1, \ldots, c_n/x_n]) \). The right size for \( P \), \( \delta \), is \( \sum \delta(u) \) of its right terms.

So, \( \delta(h(a)) = 1 \). If \( \delta \) for \( P \) is 0, then each (dis)equation contains a right term that is a constant \( a_i : 0 \); Padovani proved decidability for this special case [7].

### 3 Tree-checking games

We present a game-theoretic characterization of interpolation inspired by model-checking games (such as in [12]) where a model, a transition graph, is traversed relative to a property. Similarly, in the following game the model is a putative solution term \( t \) that is traversed relative to the dual interpolation problem.

A potential solution \( t \) for \( P \) has the right type, is in normal form, is well-named (with variables that are disjoint from those in \( P \)) and does not contain forbidden constants. Term \( t \) is represented as a tree, \( \text{tree}(t) \). If \( t = y : 0 \) or \( a : 0 \) then \( \text{tree}(t) \) is the single node labelled with \( t \). For \( u(v_1, \ldots, v_n) \) when \( u \) is a variable or a constant, a dummy \( \lambda \) with the empty sequence of variables is placed before any subterm \( v_i \) : 0 in its tree representation. If \( t = u(v_1, \ldots, v_n) \), then \( \text{tree}(t) \) consists of the root node labelled \( u \) and \( n \)-successor nodes labelled with \( \text{tree}(v_i) \) : \( t \downarrow \), \( t \) represents that \( t' \) is the \( i \)-th successor of \( u \). If \( t = \lambda y. v \), where \( y \) could be empty, then \( \text{tree}(t) \) consists of the root node labelled \( \lambda y \) and a single successor node \( \text{tree}(v) \) : \( \lambda y. \downarrow \) \( \text{tree}(v) \). Each node labelled with an occurrence of a variable \( y_j \) has a backward arrow \( \uparrow j \) to the \( \lambda y \) that binds it: the index \( j \) tells us which element is pointed at in \( y \). We use \( t \) to be the \( \lambda \)-term \( t \), or its \( \lambda \)-tree or the label (a constant, variable or \( \lambda y \)) at its root node. Dummy \( \lambda \)'s are central to the analysis in later sections. We also assume that each node of a tree \( t \) is uniquely identified.

**Example 1.** A solution term \( t \) from [1] for the problem \( x(v) = f(a) \) where \( v = \lambda y_1 y_2 . y_1(y_2) \) is \( \lambda z. z(\lambda x. f(z(\lambda u. x, b)), z(\lambda y. z((\lambda s. s, y), a))). \) The tree for \( t \) (without backward edges and indexed forward edges) is in Figure 1. \( \square \)
Innocent game semantics following Ong in [6] provides a possible game-theoretic foundation. Given \( t \) and a (dis)equation from \( P \), there is the game board \( t@(v_1^j, \ldots, v_n^j) \) \( u_i \). Player Opponent chooses a branch of \( u_i \). There is a finite play that starts at the root of \( t \) and may repeatedly jump in and out of \( t \) and the \( v_j^j \)'s. At a constant \( a : 0 \) play ends. At other constants \( f \), player Proponent tries to match Opponent’s choice of branch. Proponent wins, when the play finishes, if the sequence of constants encountered matches the chosen branch. Play may reach \( y \) in \( t \) and then jump to \( \lambda \varphi \) in \( v_j^j \), as it is this subtree that is applied to \( \lambda \varphi \), and then when at \( z \) in \( v_j^j \) play may return to \( t \) to a successor of \( y \). Game semantics models \( \beta \)-reduction on the fixed game board without changing it using substitution. This is the rationale for the tree-checking game. However, it starts from the assumption that only \( t \) is the common structure for the problem \( P \). So, play will always be in \( t \). Jumping in and out of the \( v_j^j \)'s is coded using states. The game avoids justification pointers, using iteratively defined look-up tables.
The game \( G(t, P) \) is played by player \( \forall \), the \textit{requester}, who attempts to show that \( t \) is not a solution of \( P \). It uses a finite set of states involving elements of \( L \) and \( R \) from Definition 2. An \textit{argument} state \( q(l_1, \ldots, l_k, r) \) where each \( l_j \in L \) (and \( k \) can be 0) and \( r \in R \) occurs at a node labelled \( \lambda z_1 \ldots z_k \) in \( t \) where each \( l_j \) has the same type as \( z_j \); \( l_1, \ldots, l_k \) are the subterms applied to \( \lambda z_1 \ldots z_k \). A \textit{value} state \( q[l, r] \) where \( l \in L \) and \( r \in R \) is associated with a node labelled with \( y \) in \( t \) where \( y \) and \( l \) share the same type; \( l \) is the subterm of some \( v^j_l \) that play at \( y \) would jump to in game semantics. A \textit{final} state is \( q[\forall] \) or \( q[\exists] \).

A. \( t_m = \lambda y_1 \ldots y_j \) and \( t_m \Downarrow u \) and \( q_m = q[\theta_1, \ldots, l_j, r] \).

So, \( t_{m+1} = u, \ theta_{m+1} = theta_m \{ l_1 \eta_m/y_1, \ldots, l_j \eta_m/y_j \} \) and \( \eta_{m+1} \) are defined by cases on \( t_{m+1} \).

1. \( a : 0 \). \( \eta_{m+1} = \eta_m \). If \( r = a \) then \( q_{m+1} = q[\exists] \) else \( q_{m+1} = q[\forall] \).
2. \( f : (B_1, \ldots, B_k) \rightarrow 0 \). \( \eta_{m+1} = \eta_m \). If \( r = f(s_1, \ldots, s_k) \) then \( q_{m+1} = q_m \) else \( q_{m+1} = q[\forall] \).
3. \( y : B \). If \( \theta_{m+1}(y) = \eta_y \), then \( \eta_{m+1} = \eta_y \) and \( q_{m+1} = q[l, r] \).

B. \( t_m = f : (B_1, \ldots, B_k) \rightarrow 0 \) and \( q_m = q[l_1, \ldots, l_j, f(s_1, \ldots, s_k)] \).

So, \( \theta_{m+1} = \theta_m, \eta_{m+1} = \eta_m \) and \( q_{m+1} \) are decided as follows.

1. \( \forall \) chooses a direction \( d : 1 \leq d \leq k \) and \( t_m \Downarrow d u \). So, \( t_{m+1} = u \).

If \( s_d : 0 \), then \( q_{m+1} = q(l_i, s_d) \).
If \( s_d \) is \( \lambda x_i_1 \ldots x_i_n, s \) then \( q_{m+1} = q(c_{i_1}, \ldots, c_{i_n}, s[x_i_1, \ldots, x_i_n]) \).

C. \( t_m = y \) and \( q_m = q[l, r] \).

If \( l = \lambda z_1 \ldots z_k, w \) and \( t_m \Downarrow u, \) for \( i : 1 \leq i \leq j \), then \( \eta_{m+1} = \eta_m \{ u \theta_m/z_1, \ldots, u \theta_m/z_j \} \) else \( \eta_{m+1} = \eta_m \). Elements \( t_{m+1} \), \( q_{m+1} \) and \( \theta_{m+1} \) are by cases on \( l \).

1. \( a : 0 \) or \( \forall \alpha \). \( \theta_{m+1} = \theta_m \). If \( r = a \) then \( q_{m+1} = q[\exists] \) else \( q_{m+1} = q[\forall] \).
2. \( c : (B_1, \ldots, B_k) \rightarrow 0 \). \( \theta_{m+1} = \theta_m \). If \( r \neq c(s_1, \ldots, s_k) \) then \( t_{m+1} = t_m \) and \( q_{m+1} = q[\forall] \). If \( r = c(s_1, \ldots, s_k) \) then \( t_{m+1} = t_m \) and \( q_{m+1} = q[\forall] \). If \( r \) chooses a direction \( d : 1 \leq d \leq k \) and \( t_m \Downarrow d u \). So, \( t_{m+1} = u \).

If \( s_d : 0 \), then \( q_{m+1} = q[l_i, s_d] \).
If \( s_d \) is \( \lambda x_i_1 \ldots x_i_n, s \) then \( q_{m+1} = q(c_{i_1}, \ldots, c_{i_n}, s[x_i_1, \ldots, x_i_n]) \).

3. \( f(w_1, \ldots, w_k) \) or \( \lambda \forall \alpha f(w_1, \ldots, w_k) \). \( t_{m+1} = t_m \) and \( \theta_{m+1} = \theta_m \). If \( r \neq f(s_1, \ldots, s_k) \) then \( q_{m+1} = q[\forall] \). If \( r = f(s_1, \ldots, s_k) \) then \( t_{m+1} = t_m \) and \( q_{m+1} = q[\forall] \). If \( r \) chooses a direction \( d : 1 \leq d \leq k \). If \( s_d : 0 \) then \( q_{m+1} = q[w_d, s_d] \).
If \( s_d \) is \( \lambda y_1 \ldots y_k, s \) and \( s_d \) is \( \lambda x_i_1 \ldots x_i_n, s \) then \( q_{m+1} = q[w(c_{i_1}, \ldots, c_{i_n}, y_1, \ldots, y_k), s[x_i_1, \ldots, x_i_n]] \).

4. \( x(l_1, \ldots, l_k) \) or \( \lambda \forall \alpha x(l_1, \ldots, l_k) \). \( \eta_{m+1}(x) = \theta_l \) then \( \theta_{m+1} = \theta_l \) and \( t_{m+1} = t \) and \( q_{m+1} = q[l_1, \ldots, l_k, r] \).

**Fig. 2.** Game moves

There are two kinds of free variables, in \( t \) and in the left terms of states. Free variables in \( t \) are associated with left terms and free variables in states are associated with nodes of \( t \). So, the game appeals to a sequence of supplementary look-up tables \( \theta_k \) and \( \eta_k, k \geq 1 \); \( \theta_k \) is a partial map from variables in \( t \) to \textit{pairs} \( l \eta_j \) where \( l \in L \) and \( j < k \), and \( \eta_k \) is a partial map from variables in elements of
L to pairs $t'\theta_j$ where $t'$ is a node of the tree $t$ and $j < k$. Initially, $\theta_1$ and $\eta_1$ are both empty.

A play of $G(t, P)$ is $t_1q_1\theta_1\eta_1, \ldots, t_nq_n\theta_n\eta_n$, where $t_1$ is (the label of) a node of $t$, $t_1 = \lambda\gamma \eta$ is the root node of $t$, $q_1$ is a state and $\eta_1$ is a final state. A node $t'$ of $t$ may repeatedly occur in a play. For the initial state, $\forall$ chooses a (dis)equation $x(v_1', \ldots, v_n') \approx_{i} u_i$ from $P$ and $q_1 = q([v_1', \ldots, v_n'], u_i)$, similar to that in game semantics except $v'_i$ and $u_i$ are now part of the state (and the choice of branch in $u_i$ happens as play proceeds). If the current position is $t_{m+1}q_{m+1}\theta_{m+1}\eta_{m+1}$ and $q_m$ is not final, then $t_{m+1}q_{m+1}\theta_{m+1}\eta_{m+1}$ is determined by a unique move in Figure 2. Moves are divided into groups depending on $t_m$. Group A covers when it is a $\lambda\gamma$, B when it is a constant $f$ whose type is not 0) and C when it is a variable $y$. In B1, C2 and C3 the constants $c_\nu$ belong to the forbidden set $C_i$; these are also the only rules where $\forall$ can exercise choice (by carving out a branch). The look-up tables are used in A3 and C4 to interpret the two kinds of free variables. If $t_m$ is a $\lambda$ node, $t_{m+1}$ and $t_{m+1}$ is the variable $y$, then $\eta_{m+1}$ and $\eta_{m+1}$ are determined by the entry for $y$ in $\theta_{m+1}$. For C4, if $t_m = y$, $q_m = q[l, r]$ and $l = x(l_1, \ldots, l_k)$ or $\Lambda x(l_1, \ldots, l_k)$, then $\theta_{m+1}$ and $\theta_{m+1}$ are determined by the entry for $x$ in the table $\eta_{m+1}$: if the entry is the pair $t', \theta_i$, then $t_{m+1} = t'$ and $\theta_{m+1} = \theta_i$. It is this rule that allows play to jump elsewhere in the term tree (always to a node labelled with a $\lambda$). In contrast, for A1-A3, B1 and C2 control passes down the term tree while it remains stationary in the case of C1 and C3.

A play of $G(t, P)$ finishes with final state $q[\forall]$ or $q[\exists]$. Player $\forall$ wins it if the final state is $q[\forall]$ and she loses it if it is $q[\exists]$. $\forall$ loses the game $G(t, P)$ if for each equation she loses every play whose initial state is from it and if for each disequation she wins at least one play whose initial state is from it.

**Proposition 1.** $\forall$ loses $G(t, P)$ if, and only if, $t \models P$.

Assume $t_0 \models P$, so $\forall$ loses the game $G(t_0, P)$. The single play for Example 1 is in Figure 3. The number of different plays is at most the sum of the number of branches in the right terms $u_i$ of $P$. Let $d : 0$ be a constant that is not forbidden and does not occur in any right term of $P$. We can assume that $t_0$ only contains $d$ and constants that occur in a right term.

We also allow $\pi$ to range over subplays, consecutive subsequences of positions of any play of $G(t_0, P)$. The length of $\pi, |\pi|$, is its number of positions. The $i$th position of $\pi$ is $\pi(i)$ and $\pi(i, j)$, $i \leq j$, is the interval $\pi(i), \ldots, \pi(j)$. We write $t \in \pi(i), q \in \pi(i), \theta \in \pi(i)$ and $\eta \in \pi(i)$ if $\pi(i) = tq\theta\eta$ and $t \not\in \pi(i)$ if $\pi(i) = t'q\theta\eta$ and $t \neq t'$. If $q = q[l_1, \ldots, l_k, r]$ or $q[l, r]$ then its right term is $r$.

**Definition 3.** A subplay $\pi$ is ri, right term invariant, if $q \in \pi(1)$ and $q' \in \pi(|\pi|)$ share the same right term $r$. It is nri if it is not ri and $q' \in \pi(|\pi|)$ is not final.

In Figure 3, $\pi(1, 4)$ is ri whereas $\pi(1, 6)$ is nri. Ri subplays are an important ingredient in the decidability proof as they do not immediately contribute to the solution of $P$.

**Proposition 2.** If $t, q, \theta_1\eta_1, \ldots, t_nq_n\theta_n\eta_n$ is ri, $t_n = \lambda\gamma$ and $q[r'/r] \in \pi$ with right term $r'$ instead of $r$, then $t, q; \{r'/r\} \theta_1\eta_1, \ldots, t_nq_n\{r'/r\}\theta_n\eta_n$ is an ri play.
Definition 4. If $\pi \in G(t_0, P)$ and $\pi(i)$'s look-up table is called when move $A3$ or $C4$ produces $\pi(j)$. $j > i$, then position $\pi(j)$ is a child of position $\pi(i)$. If $\pi(i + 1)$ is the result of move $B1$ or $C2$, then $\pi(i + 1)$ is a child of $\pi(i)$. A look-up table $\beta'$ extends $\beta$ if for all $x \in \text{dom}(\beta)$, $\beta'(x) = \beta(x)$.

Proposition 3. If $\pi \in G(t_0, P)$. $j > 1$, $\pi(j)$ is not a final position and $\lambda y$ or $y \in \pi(j)$, then there is a unique $\pi(i)$, $i < j$, such that $\pi(j)$ is a child of $\pi(i)$. If $\pi(j)$ is a child of $\pi(i)$ then $\theta_j \in \pi(j)$ extends $\theta_i \in \pi(i)$ and $\eta_j \in \pi(j)$ extends $\eta_i \in \pi(i)$.

4 Tiles and subplays

Assume $t_0 \models P$. The aim is to show there is a small $t' \models P$. Although the number of plays in $G(t_0, P)$ is bounded, there is no bound in terms of $P$ on the length of a play. However, a long play contains $t_0$ subplays: across all plays, the right term of a state can change at most $\delta$ times, Definition 2. To obtain a small solution term $t'$, $t_0$ subplays will be manipulated. First, we need to relate the static structure of $t_0$ with the dynamics of play.

Definition 5. Assume $B = (B_1, \ldots, B_k) \rightarrow 0 \in T$. $\lambda$ is an atomic leaf of type 0. If $x_j : B_j$, $1 \leq j \leq k$, then $\lambda x_1 \ldots x_k$ is an atomic leaf of type $B$. If $u : 0$ is a constant or variable then $u$ is a simple tile. If $u : B$ is a constant or a variable and $t_j : B_j$, $1 \leq j \leq k$, are atomic leaves then $u(t_1, \ldots, t_k)$ is a simple tile.

Term $t_0$ without its very top $\lambda y$ consists of simple tile occurrences. Nodes (2),(3) and (11) of Figure 1 form the simple tile $z(\lambda r, \lambda)$ and the leaf (16) is
also a simple tile; node (2) by itself and node (2) with (3), are not simple tiles. Tiles can be composed to form composite tiles. If \( t(\lambda \tau) \) is a tile with leaf \( \lambda \tau \) and \( t' \) is a simple tile, then \( t(\lambda \tau \cdot t') \) is a composite tile. A (composite) tile is basic if it contains one occurrence of a free variable and no occurrences of constants, or one occurrence of a constant and no occurrences of free variables. The free variable or constant in a basic tile is its head element. Contiguous regions of \( t_0 \) are occurrences of basic tiles. In Figure 1 the region \( \varphi(\lambda s, s) \) is a basic tile rooted at (14). Throughout, we assume our use of tile in \( t_0 \) means “tile occurrence” in \( t_0 \). We write \( t(\lambda \tau_1, \ldots, \lambda \tau_k) \) if \( t \) is a basic tile with atomic leaves \( \lambda \tau_1, \ldots, \lambda \tau_k \).

**Definition 6.** Assume \( t = t(\lambda \tau_1, \ldots, \lambda \tau_k) \) is a tile in \( t_0 \). \( t \) is a top tile in \( t_0 \) if its free variable \( y \) is bound by the initial lambda \( \lambda \gamma \) of \( t_0 \). \( t \) is \( j \)-end in \( t_0 \) if every free variable below \( \lambda \tau_j \) in \( t_0 \) is bound above \( t \). It is an end tile in \( t_0 \) if it is \( j \)-end for all \( j \). \( t \) is a constant tile if its head is a constant or its free \( y \) is bound by \( \lambda \gamma \) that is an atomic leaf of a simple constant tile. Two basic tiles \( t \) and \( t' \) in \( t_0 \) are equivalent, \( t \equiv t' \), if they have the same number and type of atomic leaves and the same free variable \( y \) bound to the same \( \lambda \gamma \) in \( t_0 \).

The tile \( \varphi(\lambda x, \lambda) \) in Figure 1 is a top tile which is also 2-end and \( \varphi(\lambda u, \lambda) \) is both a top and an end tile: these tiles are equivalent.

We can also classify tiles in terms of their dynamic properties.

**Definition 7.** \( \pi \) is a play on the simple tile \( u(\lambda \tau_1, \ldots, \lambda \tau_k) \) in \( t_0 \) if \( u \in \pi(1) \). \( \lambda \tau_i \in \pi([\pi]) \) for some \( i \) and \( \pi([\pi]) \) is a child of \( \pi(1) \). It is a j-play if \( \lambda \tau_j \in \pi([\pi]) \).

A play on a simple constant tile \( u(\lambda \tau_1, \ldots, \lambda \tau_k) \) is a pair of positions \( \pi(i, i + 1) \) with \( u \in \pi(i) \) and \( \lambda \tau_j \in \pi(i + 1) \) for some \( j \) (by moves B1 or C2 of Figure 2).

A play \( \pi \) on a simple non-constant tile \( \gamma(\lambda \tau_1, \ldots, \lambda \tau_k) \) in \( t_0 \) can be of arbitrary length. It starts at \( y \) and finishes at a leaf \( \lambda \tau_j \). In between, flow of control can be almost anywhere in \( t_0 \). Crucially, the look-up tables of \( \pi([\pi]) \) extend those of \( \pi(1) \) by Fact 3: this means that the free variables in the subtree of \( t_0 \) rooted at \( y \) and the free variables in \( w \) when \( q[\lambda z_1 \ldots z_k, w, r] \in \pi(1) \) preserve their values.

If \( \pi \in G(t_0, P) \) and \( y \in \pi(i) \) then there can be zero or more plays \( \pi(i, j) \) on \( \gamma(\lambda \tau_1, \ldots, \lambda \tau_k) \) in \( t_0 \): simple tiles \( u : 0 \) have no plays. If \( \pi(i, m) \) is a j-play on \( \gamma(\lambda \tau_1, \ldots, \lambda \tau_k) \) and \( \pi(i, n) \) with \( n > m \), is also a play on this tile, then there is a position \( \pi(m', m < m' < n) \), that is a child of \( \pi(m) \). In the case of \( \pi \) in Figure 3 on the tree in Figure 1, \( \pi(2, 3) \) is a 1-play on \( \varphi(\lambda x, \lambda) \) and \( \pi(2, 9) \) is also a play on this tile: it is \( \pi(8) \) that is the (only) child of \( \pi(3) \).

A play \( \pi \) on a basic tile consists of consecutive subplays on the simple tiles that are on the branch between the top of the tile and a leaf \( \lambda \tau_i \in \pi([\pi]) \).

**Definition 8.** Assume \( \pi \) is a j-play (play) on tile \( t \) in \( t_0 \). It is a shortest j-play (play) if no proper prefix of \( \pi \) is a j-play (play) on \( t \) and it is an ri j-play (play) if \( \pi \) is also ri. It is an internal j-play (play) when for any \( i \) if \( t' \in \pi(i) \) then \( t' \) is a node of \( t \). Assume \( t = t(\lambda \tau_1, \ldots, \lambda \tau_k) \) is in \( t_0 \) and \( \pi \) is a subplay. We inductively define when \( t \) is j-directed in \( \pi \); if \( t \notin \pi(i) \) for all \( i \), then \( t \) is j-directed in \( \pi \); if \( \pi(i) \) is the first position with \( t \in \pi(i) \) and there is a shortest j-play \( \pi(i, m) \) on \( t \) and \( \pi(i, m) \) is ri and \( t \) is j-directed in \( \pi(m + 1, [\pi]) \), then \( t \) is j-directed in \( \pi \). Tile \( t \) is j-directed in \( t_0 \) if it is j-directed in every \( \pi \in G(t_0, P) \).
\(\pi(2, 3)\) of Figure 3 is a shortest play on tile \(z(\lambda x, \lambda)\) of Figure 1: this play is ri, internal and a shortest 1-play. Although \(\pi(2, 9)\) is a shortest 2-play, it is neither a shortest play nor an internal play. If \(t\) is \(j\)-directed in \(t_0\) then each \(\pi \in G(t_0, P)\) contains a (unique) sequence of ri intervals which are shortest \(j\)-plays on \(t\).

Assume \(t = t(\lambda x_1, \ldots, \lambda x_k)\) is a top tile in \(t_0\) and \(\pi \in G(t_0, P)\). Consider two positions \(t \in \tau(\pi, i)\) and \(t \in \tau'(\pi, i)\). The states \(q \in \tau(\pi, i)\) and \(q' \in \tau(\pi, i')\) have the form \(q[v, r]\) and \(q[v', r']\) where \(v\) is a closed left term (a \(v_j\) from a (dis)equation of \(P\)). Therefore, a shortest play \(\tau(\pi, i, i + m)\) on \(t\) is internal (as a jump outside \(t\) requires there to be a free variable in \(v\) via move \(C4\) of Figure 2). If the play \(\tau(\pi, i, i + m)\) is ri then there is a corresponding ri play \(\tau'(\pi, i, i + m)\) on \(t\) consisting of the same sequence of positions in \(t\) and states (except for their right terms \(r\) and \(r'\)). Tile \(t\) is, therefore, \(j\)-directed in \(\pi\) when \(\lambda \tau_j \in \pi(\pi, i + m)\). If the play \(\tau(\pi, i, i + m)\) is \(\text{nri}\) then there is a subplay \(\tau(\pi, i, i' + m')\) where control is never outside \(t\) that is either a shortest play on \(t\) and \(\text{nri}\) or \(|\pi| \leq i' + m'\). If \(t\) is a \(j\)-end (end) tile and \(t \in \pi(\pi, i)\) then there can be at most one \(j\)-play (play) \(\pi(i, i)\) on \(t\).

Tile \(t'\) is \(j\)-below \(t(\lambda x_1, \ldots, \lambda x_k)\) in \(t_0\) if there is a branch in \(t_0\) from \(\lambda \tau_j\) to \(t'\). If two tiles \(t_1\) and \(t_2\) are equivalent, \(t_1 \equiv t_2\) and \(t_2\) is \(j\)-below \(t_1\) in \(t_0\), then \(t_2\) is an embedded tile. Shortest plays on the embedded tile \(t_2\) are constrained by earlier shortest plays on \(t_1\) and in the case of embedded end tiles there is a stronger property that is critical to the decidability proof.

**Proposition 4.** If \(t_1 \equiv t_2\) are end tiles in \(t_0\) and \(t_2\) is \(j\)-below \(t_1\), then either \(t_2\) is \(j\)-directed in \(t_0\), or there are \(\pi, \pi' \in G(t_0, P)\), an \(\text{nri}\) \(j\)-play \(\tau(m_1, m_1 + n_1)\) on \(t_1\) and a subplay \(\tau'(m_2, m_2 + n_2)\) where \(m_2 > m_1 + n_1, n_2 \leq n_1\) and \(\tau'(m_2) = \pi(m_2)\) and either \(m_2 = n_1\) and \(\pi(m_2) = \pi(m_2)\) or \(m_2 + n_2 = |\pi'|\).

5 Outline of the decision procedure

A transformation \(T\) converts a tree \(s\) into a tree \(t\), written \(sTt\). Let \(t'\) be a subtree of \(t_0\) whose root node is a variable \(y\) or a constant \(f : B \neq 0\). \(G(t_0, P)\) avoids \(t'\) if \(t' \notin \pi(\pi, i)\) for all positions and \(\pi \in G(t_0, P)\). Let \(t_0[t'/t']\) be the result of replacing \(t'\) in \(t_0\) with the tree (of tiles) \(t''\).

**T1** If \(G(t_0, P)\) avoids \(t'\) and \(d : 0\) is a constant then transform \(t_0\) to \(t_0[d/t']\)

**T2** Assume \(t(\lambda x_1, \ldots, \lambda x_k)\) is a \(j\)-directed, \(j\)-end tile in \(t_0\) and \(t'\) is the subtree of \(t_0\) rooted at \(t\). If \(t_j\) is the subtree directly beneath \(\lambda \tau_j\) then transform \(t_0\) to \(t_0[t_j/t']\).

If no play enters a subtree of \(t_0\) then it can be replaced with the constant \(d : 0\). If a tile is both \(j\)-end and \(j\)-directed, Definition 6, then it is redundant and can be removed from \(t_0\). Game-theoretically, the application of \(T_2\) amounts to omission of inessential \(i\) subplays that are structurally associated with regions of a term.

**Example 2.** Consider Example 1 and its single play in Figure 3. The tile \(z(\lambda u, \lambda)\) is \(1\)-end and \(1\)-directed because of \(\pi(6, 7)\). \(T_2\) allows us to remove it, so node \(8\) is directly beneath node \((5)\). The basic tile \(z(\lambda s.s, \lambda)\) is \(1\)-end and \(1\)-directed: the only play \(\pi(14, 17)\) is ri. A second application of \(T_2\) places node \((18)\) directly
beneath node (13). Consequently, the basic tile \( z(\lambda y.\lambda x. f(x), a) \) is also 1-end and 1-directed because of the play \( \pi(12,19) \). The starting term is therefore reduced to the smaller solution term \( \lambda z. z(\lambda x. f(x), a) \).

**Proposition 5.** If \( i \in \{1,2\} \), \( s \vdash t \) and \( s \models P \) then \( t \models P \).

If \( P \) is 3rd-order and \( t_0 \models P \) then \( t_0 \) is a tree of simple tiles: each is a constant tile or a top tile that is also an end tile. Assume that \( \Pi = \{ \pi_1, \ldots, \pi_p \} \) are the plays of \( G(t_0,P) \) and with each such \( \pi \) we associate a unique colour \( c(\pi) \). We define a partition of each \( \pi \in \Pi \) in stages. At stage 1, the initial simple tile \( t_1 \) is \( u(\lambda \pi_1, \ldots, \lambda \pi_p) \) in \( t_0 \), a constant or top tile (where \( k \) may be 0). The initial play on \( t_1 \), if there is one, is \( \pi(i_1,j_1) \) where \( i_1 = 2 \). If there is no play then \( j_1 = |\pi| \) as \( q \in \pi(j_1) \) is final, and for all \( i > 1 \), \( u \in \pi(i) \); \( t_1 \) is final for \( \pi \) and we terminate at this stage. Otherwise, play ends at an atomic leaf of \( t_1 \), \( t_2 \) is the simple tile directly below it in \( t_0 \), \( i_2 = j_1 + 1 \) and if \( \pi(i_1,j_1) \) is not then \( t_1 \) is coloured \( c(\pi) \).

At stage \( n \) and simple tile \( t_n \), \( \pi(i_n,j_n) \) is the shortest play on \( t_n \), if there is one. If there is not then \( j_n = |\pi| \) and \( t_n \) is final for \( \pi \). If \( \pi(i_n,j_n) \) is not then \( t_n \) is coloured \( c(\pi) \). If it ends at an atomic leaf of \( t_n \) then \( t_{n+1} \) is the simple tile directly below it in \( t_0 \) and \( i_{n+1} = j_n + 1 \). The partition of \( \pi \) descends a branch of \( t_0 \) until it reaches a final tile.

Consider partitioning with respect to all plays \( \pi \in \Pi \). There is a tree of simple tiles, as all plays share the initial tile. Tile \( t \) is coloured if it has at least one colour and \( t \) is final if it is final for at least one play. Each play at stage 1 that ends at the same atomic leaf of \( t_1 \) shares \( t_2 \) at stage 2 and so on. Therefore, branching occurs at a (play) separator \( t_m \) at stage \( m \) if there are plays that end at different atomic leaves of \( t_m \). If a simple tile in \( t_0 \) is coloured, final or a separator then it is special. A simple tile in \( t_0 \) with atomic leaves that is not special is superfluous. Every play avoids it (so, T1 applies) or every subplay that passes through it is \( ri \) and ends at the same atomic leaf (so, T2 applies). There can be at most \( \delta \), the right size for \( P \) of Definition 2, coloured tiles, at most \( p \) final tiles and at most \( p-1 \) separators: \( p \) is bounded by the number of branches in the right terms of \( P \). Decidability of 3rd-order matching, via the small model property, follows directly from partitioning.

There is just one level of simple tile that is not a constant tile in a 3rd-order tree: so, game playing is heavily constrained as control can only descend it. With a 4th or 5th-order tree there are two levels of simple non-constant tiles: top tiles \( t \) and end tiles \( t' \) where the variable of \( t' \) is bound in \( t \). At 8th or 9th-order there are four levels. When there is more than one level, game playing may jump around the tree as Figure 3 illustrates. The mechanism for dealing with these terms hinges on the idea of tile lowering, copying tiles down branches.

The mechanism for tile lowering is not a transformation like T2. Instead, it uses an intermediate generalized tree, the unfolding, analogous to unravelling a model in modal logic, which is then refolded into a small tree. Again, a partition of (a subsequence of) \( \pi \) is defined in stages using tiles in \( t_0 \). At each stage \( n \), a simple tile \( t_n \) in \( t_0 \) and a position \( \pi(i_n) \) whose control is at the head of \( t_n \) are examined. The play \( \pi(i_n,j_n) \) is a suffix of a play of a constant or generalized tile \( t'_n \) that contains \( t_n \).
Stage 1 follows the 3rd-order case: \( t_1 \) is the first simple tile in \( t_0 \), \( t'_1 = t_1 \), and relative to \( \pi \), \( \pi(i_1, j_1) \) is defined. If \( t'_1 \) is not final for \( \pi \) then \( t_2 \) is the simple tile directly below it in \( t_0 \) and \( x_2 = j_1 + 1 \). After stage one, the unfolding of \( t_0 \) can be depicted in linear form, \([t'_1 \lambda \lambda_j] \) if \( \pi(i_1, j_1) \) finishes at \( \lambda \lambda_j \). Consider the unfolding after stage \( n \) for \( \pi \) where \( t'_n \) is not final for \( \pi \). There is the sequence \([t'_1 \lambda \lambda_1] \ldots [t'_n \lambda \lambda_n] \) of (generalized) tiles where each \( t_{k+1} \) is directly below \( \lambda \lambda_k \) in \( t_0 \) and there are subplays \( \pi(i_k, j_k) \) that start at \( t_k \) in \( t'_k \) and finish at \( \lambda \lambda_k \). If \( t_{n+1} \) is a top or constant tile, then \( t'_{n+1} = t_{n+1} \) and \( \pi(i_{n+1}, j_{n+1}) \) is either a shortest play on \( t'_{n+1} \) or \( j_{n+1} = |\pi| \). The other case is that \( t_{n+1} = y(\lambda \lambda_1, \ldots, \lambda \lambda_7) \) is directly below \( \lambda \lambda_n \) in \( t_0 \) and \( y \) is bound within an earlier tile \( t'_k \). The position \( \pi(i_{n+1}) \) is a child of a position in the interpretation of \( t'_k \) that is the effect of the suffix play \( \pi(i_k, j_k) \). The tile \( t'_{n+1} \) is \([t'_h \lambda \lambda_h] [t'_{m_1} \lambda \lambda_{m_1}] \ldots [t'_{m_n} \lambda \lambda_{m_n}] t_{n+1} \] where the \( t_{m_j} \) are the minimal number of tiles in \( t_{h+1}, \ldots, t'_{n} \) that are captured in the sense that they involve extra subplays or are final. The interpretation of \( t'_{n+1} \) at position \( \pi(i_{n+1}) \) is \([\pi^1] \ldots [\pi^{n+1}] \pi(i_{n+1}) \] where \( \pi^1 \) is the interpretation of the tile \( t'_h \) and \( \pi^{n+1} \) is that of \( t'_{m_n} \). The play \( \pi(i_{n+1}, j_{n+1}) \) is the continuation, assuming (iterated) suffix playing, on \( t'_{n+1} \) that starts at \( t_{n+1} \) and finishes at an atomic

**Fig. 4. Illustrating unfolding**
leaf of it or is final. The intention is that unfolding will be true by definition assuming a non-standard interpretation of generalized tiles which includes that their plays are suffix plays. As with the 3rd-order case, each play $\pi$ descends a branch of the unfolded tree. The remainder of the proof, the refolding, is how to extract a small term from the tree of generalized tiles. Game-theoretically, unfolding and refolding is justified by recursive permutations, repetitions and omissions of $\textit{ri}$ subplays.

Example 3. In Example 2, the term in Figure 1 is reduced to the left tree in Figure 4. We examine its unfolding. Tile $t'_1 = z(\lambda x, \lambda)$, $\pi(i_1, j_1) = \pi(2, 3)$, $t'_2 = f(\lambda)$ and $\pi(i_2, j_2) = \pi(4, 5)$. Now, $t_3 = x$, so $t'_3 = z(\lambda x, x, \lambda)$ as $t'_1$ is lowered (and there is no capture) and $\pi(i_3, j_3)$ is the suffix play $\pi(8, 9)$ that starts at $x$ and finishes at atomic leaf $\lambda$. Tile $t'_4 = t_4 = a$ is final for $\pi$ and $\pi(i_4, j_4) = \pi(20)$. To make the unfolding into a term tree, the initial $\lambda$ of $t_0$ is added and the constant $\textit{b : 0}$ underneath any atomic leaf that does not have a successor, the tree on the right of Figure 4. The issue is $t'_3$ whose interpretation is a suffix play. We can reinterpret it as a complete play on $t'_3$ because (the prefix play) $\pi(i_1, j_1)$ is $\textit{ri}$; the complete play has a different right term in its states, here we use Fact 2. The top $z(\lambda x, \lambda)$ and basic tile $z(\lambda x, x, \lambda)$ are 1-end and 1-directed; so, by $\text{T2}$, they are removed. The result is the small term $\lambda z. f(a)$. 

References

4. Huet, G. Résolution d'équations dans les langages d'ordre 1, 2, ... ω. Thèse de doctorat d'état, Université Paris VII, (1976).
11. Statman, R. The typed $\lambda$-calculus is not elementary recursive. Theoretical Computer Science, 9, 73-81, (1979).