On model-checking trees generated by higher-order recursion schemes

C.-H. L. Ong*
Oxford University Computing Laboratory

Abstract

We prove that the modal mu-calculus model-checking problem for (ranked and ordered) node-labelled trees that are generated by order-n recursion schemes (whether safe or not, and whether homogeneously typed or not) is n-EXPTIME complete, for every $n \geq 0$. It follows that the monadic second-order theories of these trees are decidable.

There are three major ingredients. The first is a certain transference principle *from the tree generated by the scheme – the* value tree *– to an auxiliary* computation tree*, which is itself a tree generated by a related order-0 recursion scheme (equivalently, a regular tree). Using innocent game semantics in the sense of Hyland and Ong, we establish a strong correspondence between* paths *in the value tree and* traversals *in the computation tree. This allows us to prove that a given alternating parity tree automaton (APT) has an (accepting) run-tree over the value tree iff it has an (accepting)* traversal-tree *over the computation tree. The second ingredient is the simulation of an (accepting) traversal-tree by a certain set of annotated paths over the computation tree; we introduce* traversal-simulating *APT as a recognising device for the latter. Finally, for the complexity result, we prove that traversal-simulating APT enjoy a* succinctness property*: for deciding acceptance, it is enough to consider run-trees that have a reduced branching factor. The desired bound is then obtained by analysing the complexity of solving an associated (finite) acceptance parity game.*

1. Introduction

What classes of finitely-presentable infinite-state systems have decidable monadic second-order (MSO) theories? This is a basic problem in Computer-Aided Verification that is important to practice because standard temporal logics such as LTL, CTL and CTL* are embeddable in MSO logic. One of the best known examples of such a class are the *regular trees* as studied by Rabin in 1969. A notable advance occurred some fifteen years later, when Muller and Shupp [13] proved that the *configuration graphs of pushdown systems* have decidable MSO theories. In the 90's, as finite-state technologies matured, researchers embraced the challenges of software verification. A highlight from this period was Caucal's result [5] that *prefix-recognizable graphs* have decidable MSO theories. In 2002 a flurry of discoveries significantly extended and unified earlier developments. In a FOSSACS'02 paper [11], Knapik, Niwiński and Urzyczyn studied the infinite hierarchy of term-trees generated by higher-order recursion schemes that are *homogeneously typed* and satisfy a syntactic constraint called *safety*. They showed that for every $n \geq 0$, trees generated by order-n safe schemes are exactly those that are accepted by *order-n pushdown automata*; further they have decidable MSO theories. Later in the year at MFCS'02 [6], Caucal introduced a tree hierarchy and a graph hierarchy that are defined by mutual recursion, using a pair of powerful transformations that preserve decidability of MSO theories. Caucal's tree hierarchy coincides with the hierarchy of trees generated by higher-order pushdown automata.

Knapik *et al.* [11] have asked if the safety assumption is really necessary for their MSO decidability result. A partial answer has recently been obtained by Aehlig, de Miranda and Ong; they showed at TLCA'05 [2] that all trees up to order 2, whether safe or not, have decidable MSO theories. Independently, Knapik, Niwiński, Urzyczyn and Walukiewicz obtained a sharper result: they proved at ICALP'05 [12] that the modal mu-calculus model-checking problem for trees generated by order-2 recursion schemes (whether safe or not) is 2-EXPTIME complete. In this paper we give a complete answer to the question:

Theorem 1. *The modal mu-calculus model-checking problem for trees generated by order-n recursion schemes (whether safe or not, and whether homogeneously typed or not) is n-*EXPTIME *complete, for every $n \geq 0$. Thus these trees have decidable MSO theories.*

Our approach is to transfer the algorithmic analysis from the tree generated by a recursion scheme, which we call *value tree*, to an auxiliary *computation tree*, which is itself a tree generated by a related order-0 recursion scheme (equivalently, a regular tree). The computation tree recovers useful intensional information about the computational

*`users.comlab.ox.ac.uk/luke.ong/index.html`

Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)
0-7695-2631-4/06 $20.00 © 2006 IEEE

process behind the construction of the value tree. Using innocent game semantics [9], we then establish a strong correspondence (Theorem 3) between *paths* in the value tree and (what we call) *traversals* over the computation tree. In the language of game semantics, paths in the value tree correspond exactly to plays in the strategy-denotation of the recursion scheme; a traversal is then (a representation of) the *uncovering* of such a play. The path-traversal correspondence allows us to prove that a given alternating parity tree automaton (APT) has an accepting run-tree over the value tree if and only if it has an accepting *traversal-tree* over the computation tree (Corollary 4).

Our problem is then reduced to finding an effective way of recognising a set of infinite traversals (over a given computation tree) that satisfy the parity condition. This requires a new idea as a traversal is most unlike a path; it can jump all over the tree and may even visit certain nodes infinitely often. Our solution exploits the game-semantic connection. It is a property of traversals that their *P-views* are paths (in the computation tree). This allows us to simulate a traversal over a computation tree by (the P-views of its prefixes, which are) annotated paths of a certain kind in the same tree. The simulation is made precise in the notion of *traversal-simulating* APT. We establish the correctness of the simulation by proving that a given *property*[1] APT has an accepting traversal-tree over the computation tree if and only if the associated *traversal-simulating* APT has an accepting run-tree over the computation tree (Theorem 5). Note that decidability of the modal mu-calculus model-checking problem for trees generated by recursion schemes follows at once since computation trees are regular, and the APT acceptance problem for regular trees is decidable.

To prove n-EXPTIME completeness of the decision problem, we first establish a certain *succinctness property* (Proposition 6) for traversal-simulating APT: if a traversal-simulating APT $\mathcal{C}$ has an accepting run-tree, then it has one with a reduced branching factor. The desired time bound is then obtained by analysing the complexity of solving an associated (finite) acceptance parity game, which is an appropriate product of the traversal-simulating APT and a finite deterministic graph that unfolds to the computation tree in question.

Using a novel finitary semantics of the lambda calculus, Aehlig [3] has shown that model-checking trees generated by recursion schemes (whether safe or not) against all properties expressible by non-deterministic tree automata with the trivial acceptance condition is decidable (i.e. acceptance simply means that the automaton has a run-tree).

This paper is an extended abstract. The reader is directed to the preprint [14] for further details, including proofs.

[1] *Property* APT because the APT corresponds to the property described by a given modal mu-calculus formula.

2. Preliminaries

Types are generated from the base type o using the arrow constructor $\rightarrow$. Every type A can be written uniquely as $A_1 \rightarrow \cdots \rightarrow A_n \rightarrow o$ (arrows associate to the right), for some $n \geq 0$ which is called its *arity*; we shall often write A simply as $(A_1, \cdots, A_n, o)$. We define the *order* of a type by: $ord(o) = o$ and $ord(A \rightarrow B) = \max(ord(A) + 1, ord(B))$. Let Σ be a *ranked alphabet* i.e. each Σ-symbol f has an arity $ar(f) \geq 0$ which determines its type $(\underbrace{o, \cdots, o}_{ar(f)}, o)$. Further we shall assume that each symbol $f \in \Sigma$ is assigned a finite set $\mathsf{Dir}(f)$ of exactly $ar(f)$ *directions*, and we define $\mathsf{Dir}(\Sigma) = \bigcup_{f \in \Sigma} \mathsf{Dir}(f)$. Let D be a set of directions; a *D-tree* is just a prefix-closed subset of D^*, the free monoid of D. A ***Σ-labelled tree*** is a function $t : \mathsf{Dom}(t) \longrightarrow \Sigma$ such that $\mathsf{Dom}(t)$ is a $\mathsf{Dir}(\Sigma)$-tree, and for every node $\alpha \in \mathsf{Dom}(t)$, the Σ-symbol $t(\alpha)$ has arity k if and only if α has exactly k children and the set of its children is $\{ \alpha\, i : i \in \mathsf{Dir}(t(\alpha)) \}$ i.e. t is a *ranked*[2] tree. Henceforth we shall assume that the ranked alphabet Σ contains a distinguished nullary symbol $\perp$ which will be used exclusively to label "undefined" nodes.

Note. We write $[m]$ as a shorthand for $\{ 1, \cdots, m \}$. *Henceforth we fix a ranked alphabet Σ for the rest of the paper*, and set $\mathsf{Dir}(f) = [ar(f)]$ for each $f \in \Sigma$; hence we have $\mathsf{Dir}(\Sigma) = [ar(\Sigma)]$, writing $ar(\Sigma)$ to mean $\max\{ ar(f) : f \in \Sigma \}$.

For each type A, we assume an infinite collection Var^A of variables of type A, and write Var to be the union of Var^A as A ranges over types. A (deterministic) ***recursion scheme*** is a tuple $G = \langle \Sigma, \mathcal{N}, \mathcal{R}, S \rangle$ where Σ is a ranked alphabet of *terminals*; $\mathcal{N}$ is a set of typed *non-terminals*; $S \in \mathcal{N}$ is a distinguished *start symbol* of type o; $\mathcal{R}$ is a finite set of rewrite rules – one for each non-terminal $F : (A_1, \cdots, A_n, o)$ – of the form

$$F\, \xi_1 \cdots \xi_n \;\rightarrow\; e$$

where each ξ_i is in Var^{A_i}, and e is an *applicative term*[3] of type o constructed from elements of $\Sigma \cup \mathcal{N} \cup \{ \xi_1, \cdots, \xi_n \}$. The *order* of a recursion scheme is the highest order of its non-terminals.

We use recursion schemes as generators of Σ-labelled trees. The ***value tree*** of (or the tree *generated* by) a recursion scheme G is a possibly infinite applicative term, but viewed as a Σ-labelled tree, *constructed from the terminals*

[2] In the sequel, we shall have occasions to consider unordered trees whose nodes are labelled by symbols of an *unranked* alphabet Γ. To avoid confusion, we shall call these trees ***Γ-labelled unranked trees***.

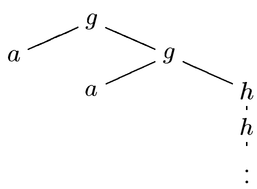
[3] ***Applicative terms*** are terms constructed from the generators using the application rule: if $d : A \rightarrow B$ and $e : A$ then $(de) : B$. Standardly we identify finite Σ-labelled trees with applicative terms of type o generated from Σ-symbols endowed with 1st-order types *as given by their arities*.

in Σ, that is obtained by unfolding the rewrite rules of G *ad infinitum*, replacing formal by actual parameters each time, starting from the start symbol S.

Example 2.1 (***Running***)**.** [*The simple recursion scheme defined here will be used to illustrate various concepts throughout the paper.*] Let G be the order-2 (unsafe) recursion scheme with rewrite rules:

$$\begin{array}{rcl} S & \rightarrow & H\,a \\ H\,z^{o} & \rightarrow & F\,(g\,z) \\ F\,\varphi^{(o,o)} & \rightarrow & \varphi\,(\varphi\,(F\,h)) \end{array}$$

where the arities of the terminals g, h, a are $2, 1, 0$ respectively. The value tree $[\![\, G \,]\!]$ is the Σ-labelled tree defined by the infinite term $g\,a\,(g\,a\,(h\,(h\,(h\,\cdots))))$:



The only infinite ***path*** in the tree is the node-sequence $\epsilon \cdot 2 \cdot 22 \cdot 221 \cdot 2211 \cdots$ (with the corresponding ***trace*** $g\,g\,h\,h\,h \cdots \in \Sigma^{\omega}$).

This paper is concerned with the decision problem: *Given a modal mu-calculus formula φ and an order-n recursion scheme G, does $[\![\, G \,]\!]$ satisfy φ (at ϵ)?* The problem is equivalent [7] to deciding whether a given alternating parity tree automaton $\mathcal{B}$ has an accepting run-tree over $[\![\, G \,]\!]$. To fix notation, an ***alternating parity tree automaton*** (or APT for short) over Σ-labelled trees is a tuple $\langle\, \Sigma,\ Q,\ \delta,\ q_0,\ \Omega \,\rangle$ where Σ is the input ranked alphabet, Q is a finite state-set, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \longrightarrow \mathsf{B}^{+}(\mathsf{Dir}(\Sigma) \times Q)$ is the transition function whereby for each $f \in \Sigma$ and $q \in Q$, we have $\delta(q, f) \in \mathsf{B}^{+}(\mathsf{Dir}(f) \times Q)$ where $\mathsf{B}^{+}(X)$ is the set of positive boolean formulas over elements of X, and $\Omega : Q \longrightarrow \mathbb{N}$ is the priority function.

3. Computation trees and traversals

The ***long transform***, $\overline{G}$, of a recursion scheme G is an order-0 recursion scheme. Its rules are obtained from those of G by applying the following four-stage transformation in turn. For each G-rule:

1. *Expand the RHS to its η-long form*: We hereditarily η-expand every subterm – even if it is of ground type so that $e : o$ expands to $\boldsymbol{\lambda}.e$ – provided it is the *operand* of an occurrence of the application operator.

2. *Insert long-apply symbols $@_A$*: Replace each *ground-type* subterm $D\,e_1 \cdots e_n$ by $@_A\,D\,e_1 \cdots e_n$ where $A = ((A_1, \cdots, A_n, o), A_1, \cdots, A_n, o)$.

3. *Curry the rewrite rule*. I.e. transform the rule $F\,\varphi_1\,\cdots\,\varphi_n \rightarrow e$ to $F \rightarrow \boldsymbol{\lambda}\varphi_1\,\cdots\,\varphi_n.e$.

4. *Rename bound variables afresh*.

$\overline{G}$ is an order-0 recursion scheme with respect to an enlarged ranked alphabet Λ_G, which is Σ augmented by certain variables and lambdas (of the form $\boldsymbol{\lambda}\overline{\xi}$ which is a short hand for $\boldsymbol{\lambda}\xi_1 \cdots \xi_n$ where $n \geq 0$) *but regarded as terminals*. The alphabet Λ_G is a finite subset of the set

$$\underbrace{\Sigma\ \cup\ \mathit{Var}\ \cup\ \{\, @_A : A \in \mathit{ATypes} \,\}}_{\textbf{\textit{Non-lambdas}}} \cup \underbrace{\{\, \boldsymbol{\lambda}\overline{\xi} : \overline{\xi} \subseteq \mathit{Var} \,\}}_{\textbf{\textit{Lambdas}}}$$

where *ATypes* is the set of types of the shape $((A_1, \cdots, A_n, o), A_1, \cdots, A_n, o)$ with $n \geq 1$. Symbols in Λ_G are ranked as follows. A symbol $\varphi : (A_1, \cdots, A_n, o)$ from *Var* has arity n. The *long-apply* $@_A$ where $A = ((A_1, \cdots, A_n, o), A_1, \cdots, A_n, o)$ has arity $n + 1$. Lambdas $\boldsymbol{\lambda}\overline{\xi}$ have arity 1. Further, for $f \in \Lambda_G$, we define

$$\mathsf{Dir}(f) \ = \ \begin{cases} [ar(@_A) - 1] \cup \{\, 0 \,\} & \text{if } f = @_A \\ [ar(f)] & \text{otherwise} \end{cases}$$

For technical convenience, the leftmost child of an @-node is its 0-child, but for all other nodes, the leftmost child is the 1-child. The *non-terminals* of $\overline{G}$ are exactly those of G, except that they are all of type o. We can now define the ***computation tree***[4] $\lambda(G)$ to be $[\![\, \overline{G} \,]\!]$. Thus $\lambda(G)$ is the Λ_G-labelled (ranked and ordered) tree that is obtained by unfolding the $\overline{G}$-rules *ad infinitum* (note that no "β-redex" is contracted in the process).

Example 3.1. Let G be as defined in Example 2.1. We present its long transform $\overline{G}$ as follows and the computation tree $\lambda(G)$ in Figure 1.

$$\overline{G} \ : \ \begin{cases} S & \rightarrow \ \boldsymbol{\lambda}.@\,H\,(\boldsymbol{\lambda}.a) \\ H & \rightarrow \ \boldsymbol{\lambda}z.@\,F\,(\boldsymbol{\lambda}y.g\,(\boldsymbol{\lambda}.z)\,(\boldsymbol{\lambda}.y)) \\ F & \rightarrow \ \boldsymbol{\lambda}\varphi.\varphi\,(\boldsymbol{\lambda}.\varphi\,(\boldsymbol{\lambda}.@\,F\,(\boldsymbol{\lambda}x.h\,(\boldsymbol{\lambda}.x)))) \end{cases}$$

In Figure 1, for ease of reference, we give nodes of $\lambda(G)$ numeric names (in square-brackets).

We define a family of binary relations $\vdash_i$, where $i \in \mathsf{Dir}(\Lambda_G)$, between nodes of a computation tree $\lambda(G)$, called ***enabling***, as follows:

- Every lambda-labelled node β, that is the i-child of its parent node α, is i-*enabled* by α.

- A variable node β (labelled ξ_i, say) is i-*enabled* by its ***binder***, which is defined to be the largest prefix of β that is labelled by a lambda $\boldsymbol{\lambda}\overline{\xi}$, for some list $\overline{\xi} = \xi_1 \cdots \xi_n$ in which ξ_i occurs as the i-element.

[4] In recent work on deciding higher-order matching [15], Colin Stirling has introduced *property checking game* over a kind of trees determined by lambda terms. His trees are exactly the same as our computation trees.

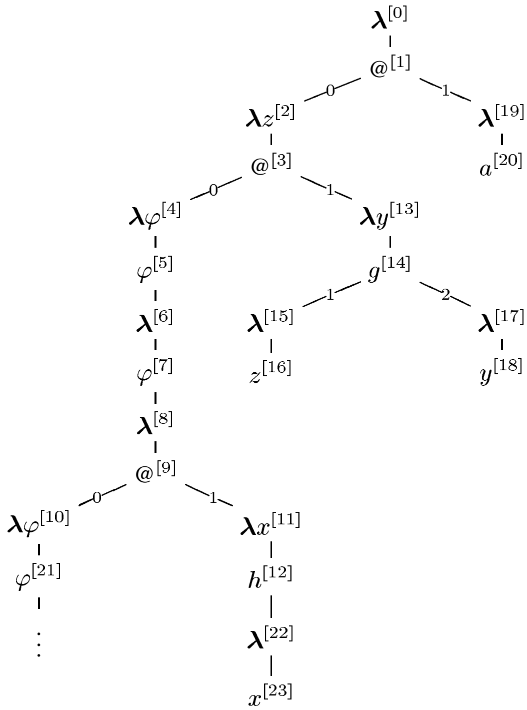


Figure 1. An order-2 computation tree.

We say that β is ***enabled*** by α just if β is i-enabled by α, for some (necessarily unique) i. A node of $\lambda(G)$ is ***initial*** if it is not enabled by any node. The initial nodes of a computation tree are the root node and all nodes labelled by a long-apply or Σ-symbol. A ***justified sequence*** over $\lambda(G)$ is a possibly infinite, lambda / non-lambda alternating sequence of nodes that satisfies the *pointer condition*: Each non-initial node that occurs in it has a pointer to some earlier occurrence of the node that enables it.

Notation $\cdots\ n_0 \ \cdots \ n\ \cdots$ means that n points to n_0 and n is j-enabled by n_0. We say that n is *j-justified* by n_0 in the justified sequence.

The notion of *view* (of a justified sequence) and the condition of *Visibility* were first introduced in game semantics [9]. Intuitively the *P-view* of a justified sequence is a certain subsequence consisting of moves which player P considers relevant for determining his next move in the play. In the setting here, the lambda nodes are the O-moves, and the non-lambda moves are the P-moves.

The ***P-view***, $\ulcorner t \urcorner$, of a justified sequence t is a subsequence defined by recursion as follows: we let n range over non-lambda nodes

$$\begin{aligned}\ulcorner \boldsymbol{\lambda} \urcorner &= \boldsymbol{\lambda}\\ \ulcorner t\ n\ \cdots\ \boldsymbol{\lambda}\overline{\xi}\ \urcorner &= \ulcorner t \urcorner\ n\ \ \boldsymbol{\lambda}\overline{\xi}\\ \ulcorner t\, \boldsymbol{\lambda}\overline{\xi}\, n \urcorner &= \ulcorner t\, \boldsymbol{\lambda}\overline{\xi} \urcorner\, n\end{aligned}$$

In the second clause above, suppose the non-lambda node n points to some node-occurrence l (say) in t; if l appears in $\ulcorner t \urcorner$, then n in $\ulcorner t \urcorner\ n\ \ \boldsymbol{\lambda}\overline{\xi}$ is defined to point to l; otherwise n has no pointer; similarly for the third clause. We say that a justified sequence t satisfies ***P-visibility*** just in case every non-initial non-lambda node that occurs in the sequence points to some (necessarily lambda) node that appears in the P-view at that point.

Definition 3.2. ***Traversals*** over a computation tree $\lambda(G)$ are justified sequences defined by induction over the following rules. In the following, we refer to nodes of $\lambda(G)$ by their labels, and we let n range over non-lambda nodes.

(Root) The singleton sequence, comprising the root node of $\lambda(G)$, is a traversal.

(App) If $t\ @$ is a traversal, so is $t\ @\ \ \boldsymbol{\lambda}\overline{\xi}$.

(Sig) If $t\,f$ is a traversal, so is $t\ f\ \ \boldsymbol{\lambda}$ for each $1 \leq i \leq ar(f)$ with $f \in \Sigma$

(Var) If $t\, n\, \boldsymbol{\lambda}\overline{\xi}\ \cdots\ \xi$ is a traversal, so is $t\ n\ \ \boldsymbol{\lambda}\overline{\xi}\ \cdots\ \xi\ \ \boldsymbol{\lambda}\overline{\eta}$.

(Lam) If $t\, \boldsymbol{\lambda}\overline{\xi}$ is a traversal and $\ulcorner t\, \boldsymbol{\lambda}\overline{\xi}\, n \urcorner$ is a path in $\lambda(G)$, then $t\, \boldsymbol{\lambda}\overline{\xi}\, n$ is a traversal.

Thus the way that a traversal can grow is deterministic (and determined by $\lambda(G)$), except when the last node in the justified sequence is a Σ-symbol f of arity $k > 1$, in which case, the traversal can grow in one of k possible directions in the next step.

Lemma 2. *Traversals are well-defined justified sequences that satisfy P-visibility (and O-visibility). Further, the P-view of a traversal is a path in the computation tree.*

Example 3.3. The following are maximal traversals (pointers omitted) over the computation tree shown in Figure 1:

$0 \cdot 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 13 \cdot 14 \cdot 15 \cdot 16 \cdot 19 \cdot 20$
$0 \cdot 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 13 \cdot 14 \cdot 17 \cdot 18 \cdot 6 \cdot 7 \cdot 13 \cdot 14 \cdot 15 \cdot 16 \cdot 19 \cdot 20$

The preceding traversals have the same P-view, namely, $0 \cdot 1 \cdot 19 \cdot 20$. The P-view of $0 \cdots 16$ (i.e. the prefix of the 2nd traversal above that ends in 16) is $0 \cdot 1 \cdot 2 \cdot 3 \cdot 13 \cdot 14 \cdot 15 \cdot 16$.

We state an important result that underpins our approach.

Theorem 3. *Let G be a recursion scheme. There is a one-one correspondence, $p \mapsto t_p$, between maximal paths p in the value tree $[\![G]\!]$ and maximal traversals t_p over the computation tree $\lambda(G)$. Further for every maximal path p in $[\![G]\!]$, we have $t_p \upharpoonright \Sigma^- = p \upharpoonright \Sigma^-$, where $s \upharpoonright \Sigma^-$ denotes the subsequence of s consisting of only Σ^--symbols with $\Sigma^- = \Sigma \setminus \{ \perp \}$.*

Using the language of game semantics, we are claiming (in the Theorem) that the traversal t_p is (a representation of) the *uncovering* of the path p viewed as a play. The proof is by innocent game semantics [9].

Example 3.4. To illustrate Theorem 3, consider the computation tree in Figure 1. The two (maximal) traversals over $\lambda(G)$ given in Example 3.3 correspond respectively to the (maximal) paths $g \cdot a$ and $g \cdot g \cdot a$ in $[\![G]\!]$. The traversal $0{\cdot}1{\cdot}2{\cdot}3{\cdot}4{\cdot}5{\cdot}13{\cdot}14{\cdot}17{\cdot}18{\cdot}6{\cdot}7{\cdot}13{\cdot}14{\cdot}17{\cdot}18{\cdot}8{\cdot}9{\cdot}10{\cdot}21{\cdot}11{\cdot}12$ corresponds to the path $g \cdot g \cdot h$.

Relative to a *property* APT $\mathcal{B} = \langle \Sigma, Q, \delta, q_0, \Omega \rangle$ over Σ-labelled trees, an (accepting) traversal-tree of $\mathcal{B}$ over $\lambda(G)$ plays the same rôle as an (accepting) run-tree of $\mathcal{B}$ over $[\![G]\!]$. A path in a traversal-tree is a traversal in which each node is annotated by an element of Q. Formally, we have:

Definition 3.5. A ***traversal-tree*** of a property APT $\mathcal{B}$ over a Λ_G-labelled tree $\lambda(G)$ is a $(\mathsf{Dom}(\lambda(G)) \times Q)$-labelled unranked tree $t : \mathsf{Dom}(t) \longrightarrow \mathsf{Dom}(\lambda(G)) \times Q$, satisfying $t(\varepsilon) = (\varepsilon, q_0)$, and for every $\beta \in \mathsf{Dom}(t)$ with $t(\beta) = (\alpha, q)$:

- If $\lambda(G)(\alpha)$ is an @, then $t(\beta\, 1) = (\alpha\, 0, q)$.
- If $\lambda(G)(\alpha)$ is a Σ-symbol f, then there is some $S \subseteq [ar(f)] \times Q$ such that S satisfies $\delta(q, f)$ – and we pick the smallest such S; and for each $(i, q') \in S$, there is some $1 \leq j \leq ar(\Sigma) \times |Q|$, such that $t(\beta\, j) = (\alpha\, i, q')$.
- If $\lambda(G)(\alpha)$ is a variable, and α is i-justified by α_1 with $t(\beta_1\, 1) = (\alpha_1, q_1)$ for some β_1 and q_1, then $t(\beta\, 1) = (\gamma\, i, q)$ where $t(\beta_1) = (\gamma, q_1)$.

$$\begin{array}{ccccccc} \cdots & \psi \text{ or } @ & \boldsymbol{\lambda}\overline{\xi} & \cdots & \xi & \boldsymbol{\lambda}\overline{\varphi} \\ & (\gamma, q_1) & (\alpha_1, q_1) & & (\alpha, q) & (\gamma\, i, q) \\ & \beta_1 & \beta_1\, 1 & & \beta & \beta\, 1 \end{array}$$

Note that $\gamma\, i$ is a lambda node that is i-justified by γ which is labelled by either an @-symbol or a variable.

- If $\lambda(G)(\alpha)$ is a lambda, then $t(\beta\, 1) = (\alpha\, 1, q)$.

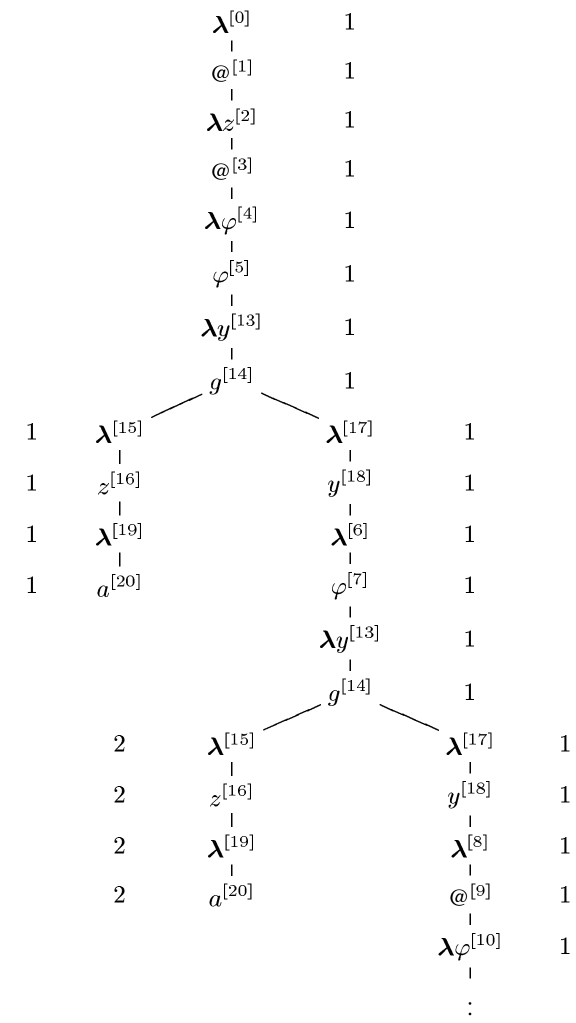


Figure 2. A traversal-tree of an APT over $\lambda(G)$.

A traversal-tree t is ***accepting*** if all infinite traces $(\alpha_0, q_0)\,(\alpha_1, q_{i_1})\,(\alpha_2, q_{i_2})\cdots$ through it satisfy the parity condition, namely, $\lim\sup\langle \Omega(q_{i_j}) : j \geq 0 \rangle$ is even.

It follows from the definition that (the element-wise first-projection of) every trace of a traversal-tree is a traversal over the computation tree.

Example 3.6. Take G as defined in Example 2.1. Consider an APT $\mathcal{B}$ over Σ-labelled trees with state-set $Q = \{1, 2\}$ where 1 is the initial state, and states 1 and 2 have priorities 1 and 2 respectively. The transition map $\delta : Q \times \Sigma \longrightarrow \mathsf{B}^+([ar(\Sigma)] \times Q)$ is defined as follows:

$$\delta \; : \; \left\{ \begin{array}{lcl} (1, g) & \mapsto & ((1,1) \wedge (2,1)) \;\vee\; ((1,2) \wedge (2,1)) \\ (1, a) & \mapsto & \mathsf{true} \\ (2, a) & \mapsto & \mathsf{true} \end{array} \right.$$

Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)
0-7695-2631-4/06 \$20.00 © 2006 IEEE

In Figure 2, we present a traversal-tree of $\mathcal{B}$ over $\lambda(G)$.

We state a straightforward consequence of Theorem 3:

Corollary 4. *There is a one-one correspondence between*

(i) accepting run-trees of $\mathcal{B}$ over $[\![G]\!]$

(ii) accepting traversal-trees of $\mathcal{B}$ over $\lambda(G)$.

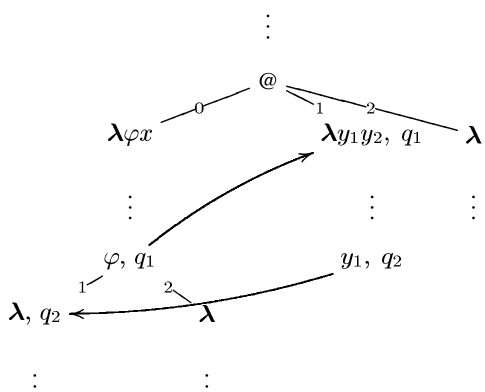
Our task is therefore reduced to that of effectively recognising accepting traversal-trees.

4. The traversal-simulating APT

An informal explanation

We want to find a device that can recognise accepting traversal-trees of a property APT $\mathcal{B}$ over a computation tree. This is far from trivial since a traversal can jump all over the tree and may even visit some nodes infinitely often. Our idea is to exploit Lemma 2: The P-view of a traversal is a path. Thus a maximal traversal can be simulated by the set of P-views of all its finite prefixes. The challenge is then to define an alternating parity automaton (which we will call *traversal-simulating* in order to distinguish it from the *property* APT) that recognises precisely the set of paths of the computation tree that simulate an *accepting* traversal-tree of $\mathcal{B}$.

Fix a property APT $\mathcal{B} = \langle\, \Sigma, Q, \delta, q_0, \Omega \,\rangle$ with p priorities. Suppose a traversal jumps from a node labelled φ with simulating state $q_1 \in Q$ to a subtree (denoting the actual parameter of that formal parameter φ) rooted at a node labelled $\boldsymbol{\lambda} y_1 y_2$; suppose it subsequently exits the subtree through y_1 with simulating state q_2, and rejoins the original subtree through the first $\boldsymbol{\lambda}$-child of the φ-labelled node, as follows:



We simulate the traversal by *paths* in the computation tree, making appropriate *guesses*, which will need to be verified subsequently:

- When reading the node φ with simulating state q_1, the automaton, having *guessed* that the jump to $\boldsymbol{\lambda} y_1 y_2$ will eventually return to the 1-child of the node φ with simulating state q_2, descends in direction 1.
- In order to verify the guess, an automaton is *spawn* to read the root of the subtree that denotes the actual parameter of φ (i.e. the node labelled by $\boldsymbol{\lambda} y_1 y_2$).

At a node α that is labelled by @, in addition to the main simulating automaton that descends in the direction of the leftmost child labelled by $\boldsymbol{\lambda}\xi_1 \cdots \xi_n$ (say), we *guess*, for each variable $\xi_i : A_i$ in the list of formal parameters $\xi_1 \cdots \xi_n$, a number of quadruples of the shape (ξ_i, q, m, c), which we call *profiles* for ξ_i, where

- $q \in Q$ is the state that is simulated when a ξ_i-labelled node (a descendent of α) is encountered by the descending automaton, simulating the traversal
- $m \in [p]$ is the maximal priority that will have been seen at that point, since reading the node labelled by $\boldsymbol{\lambda}\xi_1 \cdots \xi_n$
- The *interface* c, which is a subset of $\bigcup_{i=1}^{n} \mathbf{VP}_G^{\mathcal{B}}(A_i)$, where $\mathbf{VP}_G^{\mathcal{B}}(A)$ is the set of profiles of variables of type A occurring in $\lambda(G)$ with respect to the property APT $\mathcal{B}$, captures the manner in which the traversal, which now jumps to a neighbouring subtree denoting the actual parameter of ξ_i, will eventually return to the children of the ξ_i-labelled node (i.e. with what simulating state, and through which child of ξ_i).

Formal definition

Henceforth we fix a recursion scheme G and its associated computation tree $\lambda(G)$, and fix a *property* APT

$$\mathcal{B} = \langle\, Q,\ \Sigma,\ \delta : Q \times \Sigma \longrightarrow \mathsf{B}^+([ar(\Sigma)] \times Q),\ q_0,\ \Omega \,\rangle$$

with p priorities, over Σ-labelled trees. Let Var_G^A be the (finite) set of variables of type A that occur as labels in $\lambda(G)$.

Definition 4.1. (i) The set $\mathbf{VP}_G^{\mathcal{B}}(A)$ of ***profiles*** for variables of type A in $\lambda(G)$ relative to $\mathcal{B}$ are defined as follows:

$$\mathbf{VP}_G^{\mathcal{B}}(A_1, \cdots, A_n, o) \;=\; Var_G^A \times Q \times [p] \times \mathcal{P}\Big(\bigcup_{i=1}^{n} \mathbf{VP}_G^{\mathcal{B}}(A_i)\Big)$$

If $n = 0$, we have $\mathbf{VP}_G^{\mathcal{B}}(o) = Var_G^o \times Q \times [p] \times \mathcal{P}(\varnothing)$. For every variable $\xi : A$ that occurs as a label in $\lambda(G)$, we write $\mathbf{VP}_G^{\mathcal{B}}(\xi : A)$ for the set of profiles for ξ. Take any $(\xi, q, m, c) \in \mathbf{VP}_G^{\mathcal{B}}(\xi : A)$; we shall refer to m as the ***priority*** and c the ***interface*** of the profile respectively.

(ii) An ***active profile*** is a pair θ^b where θ is a profile and $b \in \{\,\mathtt{t}, \mathtt{f}\,\}$. The boolean value b is the answer to the question:

Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)
0-7695-2631-4/06 $20.00 © 2006 IEEE

"Is the highest priority seen thus far (since the creation of the active profile) equal to m?" An ***environment*** is a set of active profiles for variables that occur as labels in $\lambda(G)$.

Notations. Take an active profile $(\xi, q, m, c)^b$. For any priority $l \le p$, we define an *update* function of b:

$$(\xi, q, m, c)^b \uparrow l \;=\; \begin{cases} (\xi, q, m, c)^{b \vee [l=m]} & \text{if } l \le m \\ \text{undefined} & \text{otherwise} \end{cases}$$

where $[l{=}m]$ denotes the Boolean value of the equality test "$l = m$". For any profile θ, we define $\theta \uparrow m$ (by abuse of notation) to be $\theta^{\mathsf{f}} \uparrow l$. Let ρ be an environment. We define $\rho \uparrow l$ by point-wise extension i.e. we say that $\rho \uparrow l$ is defined just if $\theta^b \uparrow l$ is defined for all active profiles $\theta^b \in \rho$, and is equal to $\{\, \theta^b \uparrow l : \theta^b \in \rho \,\}$.

Definition 4.2. The auxiliary ***traversal-simulating alternating parity automaton*** (w.r.t. $\mathcal{B}$) over Λ_G-labelled trees is given by $\mathcal{C} = \langle\, \Lambda_G,\ Q_{\mathcal{C}},\ \delta_{\mathcal{C}},\ q_0\,\varnothing,\ \Omega_{\mathcal{C}} \,\rangle$ where $Q_{\mathcal{C}}$ consists of pairs $q\,\rho$ and triples $q\,\rho\,\theta$ such that $q \in Q$ is the $\mathcal{B}$-state being simulated – called the *simulating state*, ρ is an environment, and θ is a variable profile; the pair $q_0\,\varnothing$ is the initial state. The priority of a $\mathcal{C}$-state, or ***$\mathcal{C}$-priority***, is defined by cases:

$$\Omega_{\mathcal{C}} \;:\; \begin{cases} q\,\rho & \mapsto \quad \Omega(q) \\ q\,\rho\,\theta & \mapsto \quad m, \quad \text{where } m \text{ is the priority of } \theta. \end{cases}$$

Given a $\mathcal{C}$-state $d = q\,\rho$ or $q\,\rho\,\theta$, we say that its ***$\mathcal{B}$-priority*** is $\Omega(q)$.

Definition of the transition function $\delta_{\mathcal{C}}$

The automaton starts by reading the root node ε of $\lambda(G)$ with the initial state $q_0\,\varnothing$. Rather than giving the positive Boolean formula $\delta_{\mathcal{C}}(d, l)$ for each $d \in Q_{\mathcal{C}}$ and $l \in \Lambda_G$, we describe the action of the automaton with state $d = q\,\rho$ or $q\,\rho\,\theta$ reading a node α of the computation tree, by a case analysis of $l = \lambda(G)(\alpha)$.

Cases of the label l:

Case 1: l is a Σ-symbol f of arity $r \ge 0$, and $d = q\,\rho$.

If $\delta(q, f) \in \mathsf{B}^{+}([ar(f)] \times Q)$ is not satisfiable, the automaton aborts; otherwise, guess a satisfying set, say

$$S \;=\; \{\, (i_1, q_{j_1}), \cdots, (i_k, q_{j_k}) \,\}$$

where $k \ge 0$ (with $k = 0$ iff $S = \varnothing$), and guess environments $\rho_1, \cdots, \rho_k$, such that

$$\bigcup_{i=1}^{k} \rho_i \;=\; \rho. \tag{1}$$

Spawn k automata with states

$$q_{j_1}\ \rho_1 \uparrow \Omega(q_{j_1}), \quad \cdots, \quad q_{j_k}\ \rho_k \uparrow \Omega(q_{j_k})$$

in directions $i_1, \cdots, i_k$ respectively provided $\rho_i \uparrow \Omega(q_{j_i})$ is defined for all i, otherwise the automaton aborts.

Note. In case the arity $r = 0$, since $\delta(q, f) \in \mathsf{B}^{+}([0] \times Q)$ and $[0] = \varnothing$, we have $\delta(q, f)$ is either true or false. If the former, note that true is satisfied by the every set in $\mathcal{P}([0] \times Q)$, namely $\varnothing$; it follows that equation (1) can only be satisfied provided $\rho = \varnothing$.

Case 2: l is a variable $\varphi : (A_1, \cdots, A_n, o)$ where $n \ge 0$, and $d = q\,\rho\,\theta$.

We check that θ has the shape (φ, q, m, c) for some interface c and $m \le p$ such that $(\varphi, q, m, c)^{\mathsf{t}} \in \rho$; otherwise the automaton aborts. Suppose

$$c \;=\; \{\, \underbrace{(\xi_{i_j}, q_{l_j}, m_j, c_j)}_{\theta_j} \mid 1 \le j \le r \,\}$$

for some $r \ge 0$ (with $c = \varnothing$ iff $r = 0$). (In case φ is order 2 or higher, we may assume that $\xi_j : A_j$ so that we have $1 \le i_j \le n$.)

Guess ρ' to be one of ρ or $\rho \setminus \{\, (\varphi, q, m, c)^{\mathsf{t}} \,\}$. For each $1 \le j \le r$, guess distinct environments $\rho_{j1}, \cdots, \rho_{jr_j}$ with $r_j \ge 1$, such that

$$\bigcup_{j=1}^{r} \bigcup_{k=1}^{r_j} \rho_{jk} \;=\; \rho'. \tag{2}$$

For each $1 \le j \le r$ and each $1 \le k \le r_j$, spawn an automaton with $\mathcal{C}$-state

$$q_{l_j} \quad (\rho_{jk} \uparrow m_j) \cup (c_j \uparrow \Omega(q_{l_j})) \quad \theta_j$$

in direction i_j, provided $(\rho_{jk} \uparrow m_j) \cup (c_j \uparrow \Omega(q_{l_j}))$ is defined for all j and k, otherwise the automaton aborts.

Note. If φ is order 0, the interface c in θ is necessarily empty (i.e. $r = 0$). Thus, for equation (2) to hold, we must have $\rho' = \varnothing$; it follows that we must have $\rho = \{\, (\varphi, q, m, \varnothing) \,\}$.

Case 3: l is @ of type $((A_1, \cdots, A_n, o), A_1, \cdots, A_n, o)$ where $n \ge 1$, and $d = q\,\rho$.

Guess a set of profiles $c \subseteq \bigcup_{i=1}^{n} \mathbf{VP}_G^{\mathcal{B}}(\xi_i : A_i)$ and spawn an automaton with state $q\ \ c \uparrow \Omega(q)$ in direction 0, with

$$c \;=\; \{\, \underbrace{(\xi_{i_j}, q_{l_j}, m_j, c_j)}_{\theta_j} : 1 \le j \le r \,\}$$

(say) where $r \ge 0$ (with $r = 0$ iff $c = \varnothing$). Note that $1 \le i_j \le n$. For each $1 \le j \le r$, guess distinct environments $\rho_{j1}, \cdots, \rho_{jr_j}$ with $r_k \ge 1$ such that

$$\bigcup_{j=1}^{r} \bigcup_{k=1}^{r_j} \rho_{jk} \;=\; \rho. \tag{3}$$

For each $1 \leq j \leq r$ and $1 \leq k \leq r_j$, spawn an automaton with $\mathcal{C}$-state

$$q_{l_j} \quad (\rho_{jk} \uparrow m_j) \cup (c_j \uparrow \Omega(q_{l_j})) \quad \theta_j$$

in direction i_j, provided $(\rho_{jk} \uparrow m_j) \cup (c_j \uparrow \Omega(q_{l_j}))$ is defined for all j and k, otherwise the automaton aborts.

Case 4: l is a lambda, with state $d = q\,\rho$ or $q\,\rho\,\theta$.

Spawn an automaton in direction 1 with $\mathcal{C}$-state e where $e = q\,\rho\,\tau$ for some $\tau^b \in \rho$ if the guess is that the label of the child node is a variable, otherwise $e = q\,\rho$.

Example 4.3. Take the computation tree $\lambda(G)$ and the property APT $\mathcal{B}$ as defined in Example 3.6. In Table 1 we give an initial part of an (accepting) run-tree of the corresponding traversal-simulating APT $\mathcal{C}$. We shall see in the sequel that the run-tree is a simulation (in the sense of Theorem 5) of the traversal-tree in Figure 2.

5. Correctness of the simulation

For the rest of the paper, we shall fix a recursion scheme G and an associated computation tree $\lambda(G)$. We shall also fix a property APT $\mathcal{B} = \langle \Sigma, Q, \delta, q_0, \Omega \rangle$ over Σ-labelled trees, and write $\mathcal{C}$ as the associated traversal-simulating APT over Λ_G-labelled trees. Our notion of simulation is correct, in the following sense:

Theorem 5. *The following are equivalent:*

(i) There is an accepting traversal-tree of $\mathcal{B}$ over $\lambda(G)$.

(ii) There is an accepting run-tree of $\mathcal{C}$ over $\lambda(G)$.

Since $\lambda(G)$ is a regular tree, an immediate corollary of the Theorem is that the modal mu-calculus model-checking problem for trees generated by arbitrary recursion schemes is decidable. In this Section we briefly sketch a proof of the Theorem.

From traversal-trees of $\mathcal{B}$ to run-trees of $\mathcal{C}$

Suppose there is an accepting traversal-tree $\mathbf{t}$ of the property APT $\mathcal{B}$ over $\lambda(G)$. Recall that $\mathbf{t}$ is a $(\mathsf{Dom}(\lambda(G)) \times Q)$-labelled unranked tree. We first perform a succession of annotation operations on $\mathbf{t}$, transforming it eventually to a $(\mathsf{Dom}(\lambda(G)) \times Q_{\mathcal{C}})$-labelled unranked tree $\widehat{\mathbf{t}}$, which has the same underlying tree as $\mathbf{t}$ i.e. $\mathsf{Dom}(\widehat{\mathbf{t}}) = \mathsf{Dom}(\mathbf{t})$. We then show that the set of P-views of traces of $\widehat{\mathbf{t}}$ gives an accepting run-tree of the traversal-simulating APT $\mathcal{C}$.

Run-trees of a traversal-simulating APT can have a rather large (though necessarily bounded) branching factor. Fortunately we can prove a kind of *succinctness result*: We show that if a traversal-simulating APT has an accepting run-tree, then it has a "narrow" accepting run-tree in the sense that it has a reduced branching factor.

Definition 5.1. A ***narrow run-tree*** of a traversal-simulating APT $\mathcal{C}$ is a run-tree satisfying the rules of Definition 4.2 except that in (2) of Case 2, for each $1 \leq j \leq r$, we guess exactly one environment $\rho_j = \rho_{j1}$ (so that $r_j = 1$) such that $\bigcup_{j=1}^{r} \rho_j = \rho$; similarly in (3) of Case 3. (Note that a narrow run-tree of $\mathcal{C}$ is *a fortiori* a run-tree of $\mathcal{C}$ in the sense of Definition 4.2.)

Proposition 6. *If the traversal-simulating APT $\mathcal{C}$ has an accepting run-tree then it has one that is narrow. The branching factor of a narrow run-tree is bounded above by the number of distinct variable profiles.*

From run-trees of $\mathcal{C}$ to traversal-trees of $\mathcal{B}$

Take an accepting run-tree $\mathbf{r}$ of $\mathcal{C}$ over $\lambda(G)$. We first construct an annotated traversal-tree $\mathbf{t}$, which is a $(\mathsf{Dom}(\lambda(G)) \times Q_{\mathcal{C}})$-labelled unranked tree. Let $\mathbf{t}^-$ be the $(\mathsf{Dom}(\lambda(G)) \times Q)$-labelled unranked tree that is obtained from $\mathbf{t}$ by replacing the $\mathcal{C}$-state that annotates each node by the $\mathcal{B}$-state that is simulated. It is straightforward to show that $\mathbf{t}^-$ is a traversal-tree of $\mathcal{B}$ over $\lambda(G)$; the tricky part is to prove that $\mathbf{t}^-$ is accepting, which follows from:

Proposition 7. *Every infinite path w in the traversal-tree $\mathbf{t}^-$ determines an infinite path p_w in the accepting run-tree $\mathbf{r}$ such that the highest $\mathcal{B}$-priority that occurs infinitely often in the former coincides with the highest $\mathcal{C}$-priority that occurs infinitely often in the latter.*

To prove the Proposition, we first need to construct p_w from a given w. Note that an infinite path w in $\mathbf{t}$ is just an infinite ($\mathcal{C}$-state annotated) traversal in $\lambda(G)$. We define a binary relation $\preccurlyeq$ over prefixes of a traversal w, called ***view order***, as follows. Let $u, v \leq w$. We say that $u \preccurlyeq v$ just in case u is a prefix of v, and $\mathbf{l}(u)$ – the last node of u – and hence every node in the P-view of u, appear in the P-view of v. (Note that the last clause implies, but is not implied by, $\ulcorner u \urcorner \leq \ulcorner v \urcorner$.)

An infinite strictly-increasing (w.r.t. prefix ordering) sequence of prefixes of w, namely $u_1 < u_2 < u_3 < \cdots$, is called a ***spinal decomposition*** of w just if

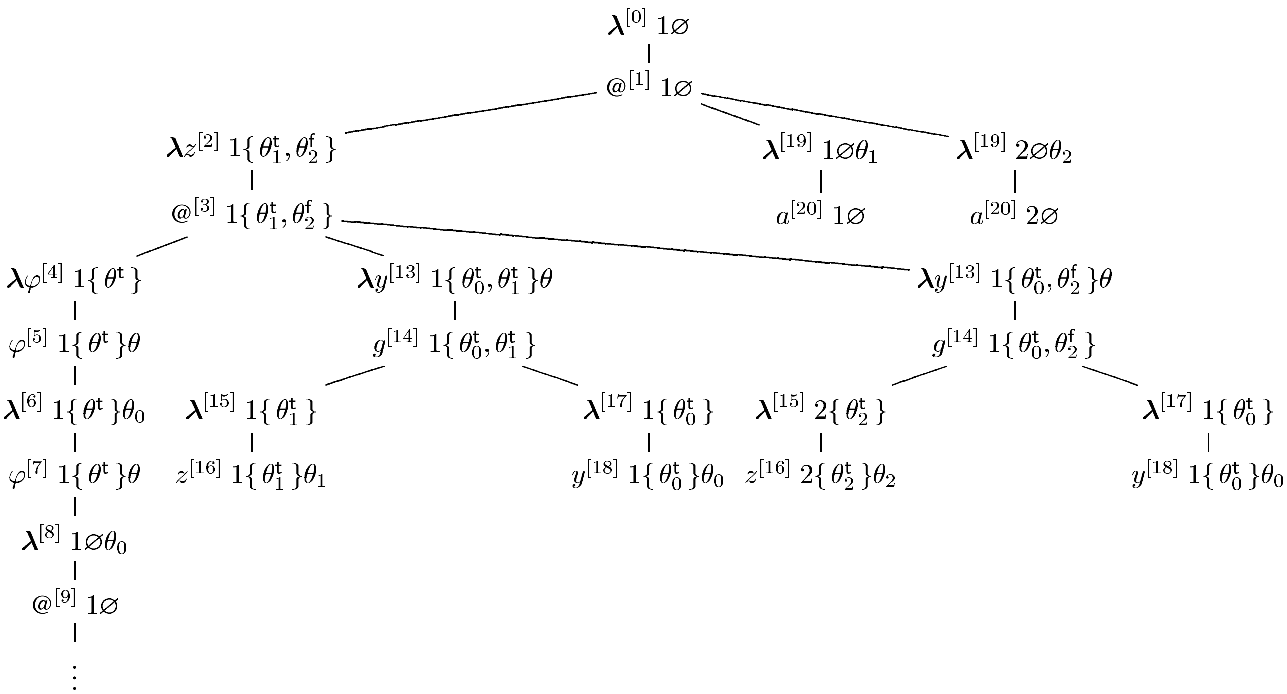
(i) $u_1 \preccurlyeq u_2 \preccurlyeq u_3 \preccurlyeq \cdots$, and

(ii) $|\ulcorner u_1 \urcorner| < |\ulcorner u_2 \urcorner| < |\ulcorner u_3 \urcorner| < \cdots$ ($|...|$ means length)

We set p_w in the above Proposition to be the infinite path in $\lambda(G)$ defined by the infinite strictly-increasing sequence $\ulcorner u_1 \urcorner < \ulcorner u_2 \urcorner < \ulcorner u_3 \urcorner < \cdots$, which we call the (associated) ***spine*** of the spinal decomposition. (Note that neither (i) nor (ii) above is a consequence of the other.)

Lemma 8. *(i) The highest $\mathcal{B}$-priority that occurs infinitely often in w coincides with the highest $\mathcal{C}$-priority that occurs infinitely often in p_w.*

(ii) Every infinite traversal w has a spinal decomposition.



Shorthand notation: $\theta = (\varphi, 1, 1, \{\,\theta_0\,\})$ $\quad \theta_0 = (y, 1, 1, \varnothing)$ $\quad \theta_1 = (z, 1, 1, \varnothing)$ $\quad \theta_2 = (z, 2, 2, \varnothing)$.

Table 1. A run-tree of the traversal-simulating APT associated with the property APT in Example 3.6.

6. Complexity analysis

We briefly sketch a proof that the modal mu-calculus model-checking problem for trees generated by order-n recursion scheme is n-EXPTIME complete. The n-EXPTIME hardness of the problem follows from Cachat's result [4] that the (sub)problem of model-checking trees generated by *safe* order-n recursion schemes is n-EXPTIME hard. We prove n-EXPTIME decidability by analysing the complexity of solving an associated acceptance parity game $\mathbf{G}(Gr(G), \mathcal{C})$, which is an appropriate product of the traversal-simulating APT $\mathcal{C} = \langle\, \Lambda_G, Q_{\mathcal{C}}, \delta_{\mathcal{C}}, q_0, \Omega_{\mathcal{C}}\,\rangle$ and a (finite) Λ_G-labelled deterministic directed graph

$$Gr(G) \;=\; \langle V,\ \rightarrow\, \subseteq V \times V,\ \lambda_G : V \longrightarrow \Lambda_G,\ v_0 \in V\,\rangle$$

which unfolds to the Λ_G-labelled computation tree $\lambda(G)$. The graph $Gr(G)$ has root v_0, and λ_G is the vertex-labelling function; it is *ranked* in the sense that the edge-set $\rightarrow\ = \bigcup_{i \in \mathsf{Dir}(\Lambda_G)} \rightarrow_i$, where each $\rightarrow_i\, \subseteq V \times V$ is a partial function such that $\rightarrow_i(v)$ is well-defined for each $v \in V$ and $i \in \mathsf{Dir}(\lambda_G(v))$.

For each $v \in V$ and $P \subseteq \mathsf{Dir}(\lambda_G(v)) \times Q_{\mathcal{C}}$, we write $[P]_v \;=\; \{\,(u, q) : (i, q) \in P \,\wedge\, \rightarrow_i(v) = u\,\}$.

Definition 6.1. The underlying digraph of the ***acceptance parity game*** $\mathbf{G}(Gr(G), \mathcal{C})$ has two kinds of vertices. *A-Vertices* (A for Abelard) are sets of the form $[P]_v$, with $v \in V$ and $P \subseteq \mathsf{Dir}(\lambda_G(v)) \times Q_{\mathcal{C}}$; and *E-Vertices* (E for Eloise) are pairs of the form (v, q) with $v \in V$ and $q \in Q_{\mathcal{C}}$. The *source vertex* is the E-vertex (v_0, q_0). The edges are defined as follows.

- For each A-vertex $[P]_v$, and for each $(u, q) \in [P]_v$, there is an edge from $[P]_v$ to (u, q).
- For each E-vertex (v, q), and for each $P \subseteq \mathsf{Dir}(\lambda_G(v)) \times Q_{\mathcal{C}}$ such that P satisfies $\delta_{\mathcal{C}}(q, \lambda_G(v))$, there is an edge from (v, q) to $[P]_v$.

The priority map $\Omega_{\mathbf{G}}$ is defined by cases as follows:

$$\Omega_{\mathbf{G}} \;=\; \left\{ \begin{array}{lcl} (v, q) & \mapsto & \Omega_{\mathcal{C}}(q) \\ \ [P]_v & \mapsto & \min\{\, \Omega_{\mathcal{C}}(q) : (u, q) \in [P]_v\,\}. \end{array} \right.$$

A *play* is a (possibly infinite) path in $\mathbf{G}(Gr(G), \mathcal{C})$ of the form $(v_0, q_0) \cdot [P_0]_{v_0} \cdot (v_1, q_1) \cdot [P_1]_{v_1} \cdot \cdots$. (For ease of reading, we use $\cdot$ as item separator in the sequence.)

Eloise resolves the E-vertices, and Abelard the A-vertices. If the play is finite and the last vertex is an A-vertex (respectively E-vertex) which is terminal, Eloise (respectively Abelard) is said to win the play. If the play is infinite, Eloise wins just if the maximum that occurs infinitely often in the following numeric sequence is even.

$$\Omega_{\mathbf{G}}(v_0, q_0) \cdot \Omega_{\mathbf{G}}([P_0]_{v_0}) \cdot \Omega_{\mathbf{G}}(v_1, q_1) \cdot \Omega_{\mathbf{G}}([P_1]_{v_1}) \cdot \cdots$$

Proposition 9. *Eloise has a (history-free) winning strategy in the acceptance parity game $\mathbf{G}(Gr(G), \mathcal{C})$ iff the traversal-simulating APT $\mathcal{C}$ accepts the Λ_G-labelled computation tree $\lambda(G)$, which is the unfolding of $Gr(G)$.*

Let G be an order-n recursion scheme and take a property APT $\mathcal{B}$ as before. For $i < n$ we define $\mathbf{VP}^{\mathcal{B}}_G(i)$ to be the union of sets of the form $\mathbf{VP}^{\mathcal{B}}_G(A)$, as A ranges over order-i types that occur in $\overline{G}$. It follows from the definition of variable profiles that $|\mathbf{VP}^{\mathcal{B}}_G(i)| = \mathsf{exp}_i O(|G| \cdot |Q| \cdot p)$ where $|G|$ is a measure of the recursion scheme G, $|Q|$ is the number of elements of Q, and exp_i is the tower-of-exponentials function of height i. Next we set $\mathbf{VP}^{\mathcal{B}}_G = \bigcup_{i=0}^{n-1} \mathbf{VP}^{\mathcal{B}}_G(i)$ and $Env^{\mathcal{B}}_G = \mathcal{P}(\mathbf{VP}^{\mathcal{B}}_G)$. It follows that $|\mathbf{VP}^{\mathcal{B}}_G| = \mathsf{exp}_{n-1} O(|G| \cdot |Q| \cdot p)$ and $|Env^{\mathcal{B}}_G| = \mathsf{exp}_n O(|G| \cdot |Q| \cdot p)$. Finally, as $Q_{\mathcal{C}} = (Q \times Env^{\mathcal{B}}_G) \cup (Q \times Env^{\mathcal{B}}_G \times \mathbf{VP}^{\mathcal{B}}_G)$, we have $|Q_{\mathcal{C}}| = \mathsf{exp}_n O(|G| \cdot |Q| \cdot p)$.

We appeal to a result due to Jurdziński [10]:

Theorem 10 (Jurdziński)**.** *The winning region of Eloise and her winning strategy in a parity game with $|V|$ vertices and $|E|$ edges and $p \geq 2$ priorities can be computed in time*

$$O\left(p \cdot |E| \cdot \left(\frac{|V|}{\lfloor p/2 \rfloor}\right)^{\lfloor p/2 \rfloor}\right)$$

Suppose the parity acceptance game $\mathbf{G}(Gr(G), \mathcal{C})$ has vertex-set V and edge-set E. The A-vertices of the game are sets of the form $[P]_v$, where $P \subseteq \mathsf{Dir}(l(v)) \times Q_{\mathcal{C}}$ and v ranges over nodes of $Gr(G)$. Thanks to the narrowing transform (see Proposition 6), it is enough to restrict P to subsets of $\mathsf{Dir}(l(v)) \times Q_{\mathcal{C}}$ that have size at most $|\mathbf{VP}^{\mathcal{B}}_G|$. This gives a tighter upper bound on the number of A-vertices of the game, namely, $(|\mathsf{Dir}(\Lambda_G)| \times |Q_{\mathcal{C}}|)^{|\mathbf{VP}^{\mathcal{B}}_G|} = \mathsf{exp}_n O(|G| \cdot |Q| \cdot p)$. It follows that $|V| = \mathsf{exp}_n O(|G| \cdot |Q| \cdot p)$. Since $|E|$ is at most $|V|^2$, time complexity for solving $\mathbf{G}(Gr(G), \mathcal{C})$ is $O\left(p \cdot (|V|)^{\lfloor p/2 \rfloor + 2}\right) = \mathsf{exp}_n O(|G| \cdot |Q| \cdot p)$. Thus[5] we have:

Theorem 11. *The acceptance parity game $\mathbf{G}(Gr(G), \mathcal{C})$ can be solved in time $\mathsf{exp}_n O(|G| \cdot |Q| \cdot p)$.*

7 Further directions

Does safety constrain expressiveness? This is the most pressing open problem. Despite [1], we conjecture that there are *inherently* unsafe trees. I.e.

[5]Though (as far as we know) Jurdziński's bound is the sharpest to date, a relatively coarse time complexity of $|V|^{O(p)}$ (based on an early result of Emerson and Lei [8]) is all that we need to prove Theorem 11.

Conjecture 12. *There is an unsafe recursion scheme whose value tree is not the value tree of any safe recursion scheme.*

Higher-order pushdown automata (PDA) characterize *safe* term-trees. A variant class of higher-order PDA *with links* (in the sense of [1]), which we call *collapsible PDA*, characterize trees generated by arbitrary higher-order recursion schemes. This work will be reported elsewhere.

What is the corresponding hierarchy of *graphs* generated by high-order recursion schemes? Are their MSO theories decidable?

We would like to develop further the pleasing mix of Semantics (games) and Verification (games) in the paper. A specific project, *pace* [3], is to give a denotational semantics of the lambda calculus "relative to an APT". More generally, construct a cartesian closed category, parameterized by APTs, whose maps are witnessed by the *variable profiles* (or "guesses" in Definition 4.1).

References

[1] K. Aehlig, J. G. de Miranda, and C.-H. L. Ong. Safety is not a restriction at level 2 for string languages. In *Proc. FOSSACS'05*, pp. 490–501, 2005. LNCS 3411

[2] K. Aehlig, J. G. de Miranda, and C.-H. L. Ong. The monadic second order theory of trees given by arbitrary level two recursion schemes is decidable. In *Proc. TLCA'05*, pp. 39–54, 2005. LNCS 3461

[3] K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. Submitted. 2006

[4] T. Cachat. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In *Proc. ICALP'03*, pp. 556–569, 2003. LNCS 2719

[5] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *Proc. ICALP'96*, pp. 194–205. 1996.

[6] D. Caucal. On infinite terms having a decidable monadic theory. In *Proc. MFCS'02*, pp. 165–176, 2002. LNCS 2420

[7] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. FOCS'91*, pp. 368–377, 1991.

[8] E. A. Emerson and C. Lei. Efficient model checking in fragments of propositional mu-calculus. In *Proc. LICS'86*, pp. 267–278, 1986.

[9] J. M. E. Hyland and C.-H. L. Ong. On Full Abstraction for PCF: I, II & III. *Info. & Comp.*, 163:285–408, 2000.

[10] M. Jurdziński. Small progress measures for solving parity games. In *Proc. STACS*, pp. 290–301, 2000. LNCS 1770

[11] T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-order pushdown trees are easy. In *Proc. FOSSACS'02*, pp. 205–222, 2002. LNCS Vol. 2303

[12] T. Knapik, D. Niwiński, P. Urzyczyn, and I. Walukiewicz. Unsafe grammars and panic automata. In *Proc. ICALP'05*, pp. 1450–1461. 2005. LNCS 3580

[13] D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *TCS*, 37:51–75, 1985.

[14] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. Preprint, 42 pp. 2006. `http://users.comlab.ox.ac.uk/luke.ong/publications/ntrees.ps`

[15] C. Stirling. A game-theoretic approach to deciding higher-order matching. In *Proc. ICALP06*, LNCS, 2006. To appear.