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

(*Draft of 22 May 2006*)

C.-H. L. Ong[1]

Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, ENGLAND

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* in the sense that 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 appropriate (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).

Categories and Subject Descriptors: F.4.1 [**Theory of Computation**]: Mathematical Logic

General Terms: Foundations of Verification

Additional Key Words and Phrases: Decidability, Monadic Second-Order Logic, Modal Mu-Calculus, Higher-Order Recursion Schemes, Game Semantics.

Contents

[1] www.comlab.ox.ac.uk/oucl/people/luke.ong.html

1. INTRODUCTION

1.1 Background and related work

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 the modal mu-calculus, and hence) 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 [Muller and Schupp 1985] 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 in this period was Caucal's result [Caucal 1996] that *prefix-recognisable graphs* have decidable MSO theories. Such graphs can be characterized [Stirling 2000] as those that are obtained from the configuration graphs of pushdown systems by factoring out ϵ-transitions. In 2002, a flurry of discoveries significantly extended and unified earlier developments. In a FOSSACS'02 paper [Knapik et al. 2002], Knapik, Niwiński and Urzyczyn introduced the class $\boldsymbol{SafeRecTree}_n\Sigma$ of term-trees (more precisely, Σ-labelled trees, where Σ is a ranked alphabet) generated by order-n recursion schemes that are *homogeneously typed* and satisfy a syntactic constraint called *safety*. They showed that for every $n \geq 0$, Σ-labelled trees generated by order-n safe recursion schemes are exactly those that are generated by *order-n pushdown automata* i.e. $\boldsymbol{SafeRecTree}_n\Sigma = \boldsymbol{PushdownTree}_n\Sigma$; further they have decidable MSO theories. Thus $\boldsymbol{SafeRecTree}_0\Sigma$ is the class of regular trees (i.e. trees generated by finite-state transducers), and $\boldsymbol{SafeRecTree}_1\Sigma$ is the class of trees generated by deterministic pushdown automata. Later in the year, Caucal [Caucal 2002] introduced a tree hierarchy, the nth level of which, $\boldsymbol{CaucalTree}_n\Sigma$, consists of Σ-labelled trees obtained from regular Σ-labelled -trees (i.e. trees from $\boldsymbol{PushdownTree}_0\Sigma$) by iterating n-times the operation of inverse deterministic rational mapping followed by unfolding. A major result in Caucal's work (Theorem 3.5 of *op. cit.*) is that $\boldsymbol{SafeRecTree}_n\Sigma = \boldsymbol{CaucalTree}_n\Sigma$. To summarize

THEOREM 1 (KNAPIK, NIWINSKI, URZYCZYN AND CAUCAL 2002). *For any ranked alphabet Σ, and for every $n \geq 0$, we have $\boldsymbol{SafeRecTree}_n\Sigma = \boldsymbol{PushdownTree}_n\Sigma = \boldsymbol{CaucalTree}_n\Sigma$.*

The starting point of our work is the following striking result [Knapik et al. 2002]:

THEOREM 2 (KNAPIK, NIWINSKI AND URZYCZYN 2002). *For every $n \geq 0$, trees in $\boldsymbol{SafeRecTree}_n\Sigma$ have decidable MSO theories.*

Though a rather awkward syntactic constraint, *safety*[2] plays an important algorithmic rôle. Knapik *et al.* have asked if the safety assumption is really necessary for their decidability result. A partial answer

[2]The *safety* constraint (on lambda terms) may be regarded as a reformulation of the constraint on lambda terms imposed by Damm's *derived types*, first introduced in his major study on the semantics of Algol-like languages [Damm 1982]. To define ***safety***, we first need to introduce ***homogeneous types***: The base type o is *homogeneous*; a function type $A_1 \to (A_2 \to \cdots \to (A_n \to o)\cdots)$ is *homogeneous* just if each A_i is homogeneous, and $order(A_1) \geq order(A_2) \geq \cdots \geq order(A_n)$. We say that a term (or a rewrite rule or a recursion scheme) is *homogeneously typed* just if all types that occur in it are homogeneous. Knapik *et al.* [Knapik et al. 2002] define safety as follows: A homogeneously-typed term of order $k > 0$ is said to be *unsafe* if it contains an occurrence of a parameter of order strictly less than k, otherwise the term is *safe*. An occurrence of an unsafe term t, as a subexpression of a term t', is ***safe*** if it occurs in an operand position (i.e. it is in the context $\cdots(ts)\cdots$), otherwise the occurrence is *unsafe*. A recursion scheme is ***safe*** if no unsafe term has an unsafe occurrence in the righthand side of any rewrite rule. Note that it follows from the definition that all recursion schemes of order at most 1 are safe. See Example 2.6 for an example of unsafe order-2 recursion scheme.

to the question has recently been obtained by Aehlig, de Miranda and Ong at TLCA 2005 [Aehlig et al. 2005a]; they showed that all trees up to order 2, whether safe or not, and whether homogeneously typed or not, have decidable MSO theories. Independently, Knapik, Niwiński, Urzyczyn and Walukiewicz obtained a somewhat sharper result (see their ICALP 2005 paper [Knapik et al. 2005]); they proved that the modal mu-calculus model checking problem for trees generated by order-2 homogeneously-typed recursion schemes (whether safe or not) is 2-EXPTIME complete. In this paper we extend their result in two ways: we remove the type-homogeneity assumption, and prove decidability for all finite orders. We prove:

THEOREM 3. *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* $n \geq 0$*. It follows (from e.g. [Janin and Walukiewicz 1996]) that such 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 i.e. a regular tree. The computation tree recovers useful intensional information about the computational process behind the construction of the value tree. Using innocent game semantics [Hyland and Ong 2000], we then establish a strong correspondence (Theorem 7) 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 the 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 8).

Our problem is then reduced to finding an effective way of recognising certain sets of infinite traversals (over a given computation tree) that satisfy the parity condition. This requires a new idea as a traversal in a computation tree 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*[3] 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 20).

To prove decidability (indeed n-EXPTIME completeness) of the problem, we first establish a certain *succinctness property* (Proposition 19) 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 appropriate (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.

Related work

For work on "unsafe trees", in addition to our TLCA 2005 paper and the ICALP 2005 paper of Knapik *et al.*, de Miranda's forthcoming doctoral dissertation [de Miranda 2006] will include, *inter alia*, a systematic account of the decidability of MSO theories of order-2 trees, and a proof that there is no *inherently* unsafe word language at order 2 (precisely, for every word language generated by an arbitrary order-2 recursion scheme, there is an order-2 non-deterministic safe recursion scheme that generates the same language). Using a novel finitary semantics of the simply-typed lambda calculus, Aehlig [Aehlig 2006] has shown that model-checking trees generated by recursion schemes (whether safe or not) against all properties expressible by a non-deterministic tree automaton with the trivial acceptance condition is decidable (i.e. acceptance simply means that the automaton has a run-tree).

An extended abstract of the work reported here has appeared in the Proceedings of LICS 2006 [Ong 2006b].

[3] *Property* APT because the APT corresponds to the property described by some modal mu-calculus formula.

1.2 Trees generated by higher-order recursion schemes

Types and applicative terms. Types are generated from the base type o using the arrow constructor $\rightarrow$. By convention $\rightarrow$ associates to the right. Thus every type A can be written uniquely as $A_1 \rightarrow \cdots \rightarrow A_n \rightarrow o$, for some $n \geq 0$ which is called its *arity*; we shall often write A simply as $(A_1, \cdots, A_n, o)$, identifying (o) with o in case $n = 0$. The ***order*** (or *level*) of a type measures how deeply nested it is on the left of the arrow constructor; we define $order(o) = 0$, and for $n \geq 1$, $order(A_1, \cdots, A_n, o) = 1 + \max\{\, order(A_1), \cdots, order(A_n) \,\}$. We write *Types* for the set of types, and $Types_n$ for the set of order-n types. Let Γ be a set of typed symbols i.e. each symbol $\gamma \in \Gamma$ is given a type A, written $\gamma : A$. The set of ***applicative terms*** generated by Γ, $\mathcal{T}(\Gamma)$, is the least set X containing Γ that is closed under the rule: if $s : A \rightarrow B$ and $t : A$ are in X then $(st) : B$ is in X. By convention application associates to the left, so that $s_1 \cdots s_n$ is a shorthand for $(((s_1 s_2) s_3) \cdots) s_n$ for $n \geq 2$.

Σ-labelled ranked trees. Let Σ be a *ranked alphabet* i.e. each Σ-symbol f has an arity $ar(f) \geq 0$ which determines its type $(\underbrace{o, \cdots, o}_{ar(f)}, o)$ (which we sometimes write as $o^{ar(f)} \rightarrow o$). 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. For $\alpha, \beta \in D^*$, we write $\alpha \leq \beta$ to mean α is a prefix of β; and $\alpha \prec \beta$ to mean $\beta = \alpha\, d$ for some $d \in D$. A *path* in a D-tree is a finite or infinite sequence of nodes $\alpha_0 \prec \alpha_1 \prec \alpha_2 \cdots$ such that $\alpha_0 = \epsilon$. 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*[4] and ordered 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.

Notation. We write $[m]$ as a shorthand for the set $\{\, 1, \cdots, m \,\}$, and $[m]_0$ for $[m] \cup \{\, 0 \,\}$. *Henceforth we fix a ranked alphabet Σ for the rest of the paper* where $\mathsf{Dir}(f) = [ar(f)]$ for each $f \in \Sigma$; hence we have $\mathsf{Dir}(\Sigma) = [ar(\Sigma)]$, writing $ar(\Sigma)$ to mean $\max\{\, ar(f) : f \in \Sigma \,\}$. Thus Σ-labelled trees t are *ordered* i.e. the children of a node $\alpha \in \mathsf{Dom}(t)$, namely, $\alpha\, 1, \alpha\, 2, \cdots, \alpha\, ar(t(\alpha))$, are totally ordered in the usual way. In the following, we shall use lower-case letters f, g, h, a, etc. to range over Σ-symbols.

We introduce a notion of *limit* of a sequence of Σ-labelled trees. Let t be a Σ-labelled tree and $n \geq 0$, we write $t \upharpoonright n$ to mean t truncated at level n i.e. it is the function t restricted to $\{\, w \in \mathsf{Dom}(t) : |w| \leq n \,\}$. Suppose $t_0, t_1, \cdots$ is a sequence of Σ-labelled trees such that for all $l \geq 0$, there is an m_l such that for all $n, n' \geq m_l$ we have $t_n \upharpoonright l = t_{n'} \upharpoonright l$. Then the *limit* of the sequence, written $\lim\langle\, t_n : n \in \omega \,\rangle$, is defined to be $\bigcup_{n \in \omega} t_n \upharpoonright m_n$.

Higher-order recursion schemes. For each type A, we assume an infinite set Var_A of variables of type A, such that Var_A and Var_B are disjoint whenever $A \neq B$; and we write $Var = \bigcup_{A \in Types} Var_A$. We use letters $x, y, \varphi, \psi, \chi, \xi$ to range over variables. A (deterministic) ***recursion scheme*** is a 4-tuple $G = \langle\, \Sigma, \mathcal{N}, \mathcal{R}, S \,\rangle$ where

—Σ is a ranked alphabet of *terminals*

—$\mathcal{N}$ is a finite set of *non-terminals*, each of a fixed type; we use upper-case letters F, H, etc. to range over 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 a variable of type A_i, and e is an applicative term of type o in $\mathcal{T}(\Sigma \cup \mathcal{N} \cup \{\, \xi_1, \cdots, \xi_n \,\})$.

The *order* of a recursion scheme is the highest order of its non-terminals.

[4]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*.

Value tree $[\![G]\!]$ *of a recursion scheme* G. In this paper we use recursion schemes as generators of Σ-labelled trees. Informally the *value tree*[5] of (or the tree *generated* by) a recursion scheme G is a possibly infinite applicative term *constructed from the terminals in* Σ, which is obtained by unfolding the rewrite rules of G *ad infinitum*, replacing formal by actual parameters each time, starting from the start symbol S.

To define $[\![G]\!]$, we first introduce a map $(\cdot)^\perp : \mathcal{T}(\Sigma \cup \mathcal{N}) \longrightarrow \mathcal{T}(\Sigma)$ that takes an applicative term and replaces each non-terminal, together with its arguments, by $\perp$. We define $(\cdot)^\perp$ by structural recursion as follows: we let f range over Σ-symbols, and F over non-terminals in $\mathcal{N}$

$$\begin{aligned} f^\perp &= f \\ F^\perp &= \perp \\ (st)^\perp &= \begin{cases} \perp & \text{if } s^\perp = \perp \\ (s^\perp t^\perp) & \text{otherwise.} \end{cases} \end{aligned}$$

Clearly if $s \in \mathcal{T}(\Sigma \cup \mathcal{N})$ is of ground type then $s^\perp \in \mathcal{T}(\Sigma)$ is of ground type. Henceforth we shall identify ground-type terms in $\mathcal{T}(\Sigma)$ with finite trees in $\mathcal{T}^\infty(\Sigma)$.

Next we define a one-step reduction relation $\rightarrow_G$ which is a binary relation over terms in $\mathcal{T}(\Sigma \cup \mathcal{N})$. Informally $s \rightarrow_G s'$ just if s' is obtained from s by replacing some occurrence of a non-terminal F by the righthand side of its rewrite rule in which all formal parameters are in turn replaced by their respective actual parameters, subject to the proviso that the F must occur at the head of a subterm of ground type. Formally $\rightarrow_G$ is defined by induction over the following rules:

—$Ft_1 \cdots t_n \rightarrow_G e[t_1/\xi_1, \cdots, t_n/\xi_n]$ where $F\xi_1 \cdots \xi_n \rightarrow e$ is a rewrite rule of G.
—If $t \rightarrow_G t'$ then $(st) \rightarrow_G (st')$ and $(ts) \rightarrow_G (t's)$.

The relation $\downarrow_G$ between terms and trees is then defined as follows: We say that $s \downarrow_G t$ where $s \in \mathcal{T}(\Sigma \cup \mathcal{N})$ and $t \in \mathcal{T}^\infty(\Sigma)$ just if

—there is a finite reduction sequence $s = t_0 \rightarrow_G \cdots \rightarrow_G t_n = t$, and t is a finite tree, none of whose node is labelled $\perp$; or
—there is an infinite reduction sequence $s = t_0 \rightarrow_G t_1 \rightarrow_G t_2 \cdots$ such that $t = \lim\langle t_i^\perp : i \in \omega \rangle$, and t may be a finite tree (in which case, some of t's nodes are labelled $\perp$) or an infinite tree.

Recall that $\mathcal{T}^\infty(\Sigma)$ is a complete partial order with respect to the approximation ordering $\sqsubseteq$ as defined by: $t \sqsubseteq t'$ just if $\mathsf{Dom}(t) \subseteq \mathsf{Dom}(t')$ and for all $w \in \mathsf{Dom}(t)$, we have $t(w) = \perp$ or $t(w) = t'(w)$ (i.e. t' is obtained from t by replacing some $\perp$-labelled nodes by Σ-labelled trees). We can finally define the Σ-labelled ranked tree $[\![G]\!]$, called the ***value tree*** of (or the tree *generated* by) G, as follows:

$$[\![G]\!] = \sup\{ t \in \mathcal{T}^\infty(\Sigma) : S \downarrow_G t \}.$$

The supremum is well-defined because the set in question is directed, which is a consequence of the Church-Rosser property of G viewed as a rewrite system. We write $\boldsymbol{RecTree}_n\Sigma$ for the class of value trees $[\![G]\!]$ where G ranges over order-n recursion schemes.

1.3 Monadic second-order logic, modal mu-calculus and alternating parity tree automata

This paper is concerned with the decision problem:

> *Given a modal mu-calculus formula* φ *and an order-*n *recursion scheme* G*, does (the root node of)* $[\![G]\!]$ *satisfy* φ*?*

Thanks to Emerson and Jutla [Emerson and Jutla 1991], the problem is equivalent to deciding whether a given alternating parity tree automaton $\mathcal{B}$ has an accepting run-tree over $[\![G]\!]$. Introduced by Muller and Schupp [Muller and Schupp 1987], alternating automata on trees are a generalization of non-deterministic

[5] We would like to refer to the Σ-labelled tree generated by a recursion scheme as its *value* tree, because the name is a good counterpoint to *computation tree* which we will introduce in Section 2. We have in mind here the distinction between *value* and *computation* emphasized by Moggi [Moggi 1989]. The idea is that the value tree is obtained from the computation tree by a (possibly infinite) process of evaluation.

tree automata. For a finite set X, let $\mathsf{B}^+(X)$ be the set of *positive Boolean formulas* over X i.e. for x ranging over X, we have

$$\mathsf{B}^+(X) \ni \theta \quad ::= \quad \mathsf{true} \mid \mathsf{false} \mid x \mid \theta \wedge \theta \mid \theta \vee \theta$$

For a set $Y \subseteq X$ and a formula $\theta \in \mathsf{B}^+(X)$, we say that Y *satisfies* θ just in case assigning true to elements in Y and false to elements in $X \setminus Y$ makes θ true. It follows that true is satisfied by all subsets of X, and false by none. Since $\theta \in \mathsf{B}^+(X)$ is positive, if a set Y satisfies θ, so does every superset of Y. An ***alternating parity automaton*** over Σ-labelled trees (or APT for short) is a tuple

$$\mathcal{A} \;=\; \langle\, \Sigma,\; Q,\; \delta,\; q_0,\; \Omega \,\rangle$$

where

—Σ is the input alphabet which is assumed to be ranked
—Q is a finite set of states, and $q_0 \in Q$ is the initial state
—$\delta : Q \times \Sigma \longrightarrow \mathsf{B}^+(\mathsf{Dir}(\Sigma) \times Q)$ is the transition function where, for each $f \in \Sigma$ and $q \in Q$, we have $\delta(q, f) \in \mathsf{B}^+(\mathsf{Dir}(f) \times Q)$
—$\Omega : Q \longrightarrow \mathbb{N}$ is the priority function.

A run-tree of an alternating parity automaton over a Σ-labelled ranked tree t is a just a $(\mathsf{Dom}(t) \times Q)$-labelled unranked tree[6] r. A node in r, labelled by (α, q), describes a copy of the automaton that is in state q and reading the node α of the tree t. Note that many nodes of r can correspond to the same node of t; there is no one-to-one correspondence between the nodes of the run-tree r and the nodes of t. Formally a ***run-tree*** of an alternating parity automaton $\mathcal{A}$ over a Σ-labelled ranked tree t is a $(\mathsf{Dom}(t) \times Q)$-labelled unranked tree r satisfying:

(i) $\varepsilon \in \mathsf{Dom}(r)$ and $r(\varepsilon) = (\varepsilon, q_0)$
(ii) For every $\beta \in \mathsf{Dom}(r)$ with $r(\beta) = (\alpha, q)$ and $\delta(q, t(\alpha)) = \theta$, there is a set $S \subseteq \mathsf{Dir}(\Sigma) \times Q$ that satisfies θ; and for each $(a, q') \in S$, there is some b such that $\beta\, b \in \mathsf{Dom}(r)$ and $r(\beta\, b) = (\alpha\, a, q')$.

A run-tree r is *accepting* if all its infinite paths $\beta_0\, \beta_1\, \beta_2\, \cdots$ through $\mathsf{Dom}(r)$ satisfy the *parity* acceptance condition: $\limsup \langle\, \Omega'(\beta_i) : i \geq 0 \,\rangle$ is even, where $\Omega' : \beta \mapsto \Omega(\pi_2(r(\beta)))$ and π_2 is the second-projection map i.e. the largest number that occurs infinitely often in the numeric sequence $\Omega'(\beta_0) \cdot \Omega'(\beta_1) \cdot \Omega'(\beta_2) \cdot \cdots$ is even. (For ease of reading, we use $\cdot$ as item separator in the sequence.)

Remark 1.1. Let Γ be an alphabet (which is not assumed to be ranked) and let t be a Γ-labelled tree (which is not assumed to be ranked or ordered) (e.g. take t to be a run-tree of an APT). The ***trace language*** of t, written $\mathsf{Traces}(t)$, is the set of non-null sequences $a_1 \cdots a_n \in \Gamma^*$ where $n \geq 1$ such that there is a path $\alpha_1 \cdots \alpha_n$ in the tree $\mathsf{Dom}(t)$ with $t(\alpha_i) = a_i$ for each $1 \leq i \leq n$. In other words, the trace of a Γ-labelled tree is the "language of its node labels". We shall refer to elements of $\mathsf{Traces}(t)$ as ***traces*** of t.

We can equivalently define a *run-tree* of an alternating parity automaton over a Σ-labelled tree t to be a set $\mathcal{R}$ of non-null sequences of elements of $\mathsf{Dom}(t) \times Q$, closed under non-null prefix, such that

(i) the least element of $\mathcal{R}$ is the singleton sequence (ϵ, q_0)
(ii) for every sequence $(\epsilon, q_0)\,(\alpha_1, q_{i_1}) \cdots (\alpha_n, q_{i_n})$ in $\mathcal{R}$, its element-wise *first-projection*, namely $\epsilon\, \alpha_1 \cdots \alpha_n$, is a path in the tree $\mathsf{Dom}(t)$
(iii) for every sequence $p\,(\alpha, q)$ in $\mathcal{R}$, there is a set $S \subseteq \mathsf{Dir}(\Sigma) \times Q$ such that S satisfies $\delta(q, t(\alpha))$, and for each $(i, q') \in S$, we have $p\,(\alpha, q)\,(\alpha\, i, q') \in \mathcal{R}$.

It follows from the definition that a run-tree is a certain (nonnull-prefix closed) set of Q-annotated *paths* in the tree t i.e. every node of each path is annotated with an element of Q.

Let $\partial\mathcal{R}$ be the set of infinite sequences $w = (\epsilon, q_0)\,(\alpha_1, q_{i_1})\,(\alpha_2, q_{i_2}) \cdots$ over the alphabet $\mathsf{Dom}(t) \times Q$ such that every non-null finite prefix of w is in $\mathcal{R}$. We say that the run-tree $\mathcal{R}$ is *accepting* just if for every $w \in \partial\mathcal{R}$, we have $\limsup \langle\, \Omega(q_{i_j}) : j \geq 0 \,\rangle$ is even. We shall refer to elements of $\partial\mathcal{R}$ as ***infinite traces*** of $\mathcal{R}$.

[6]Let Γ be a finite *unranked* alphabet. A ***Γ-labelled unranked tree*** r is a function $r : \mathsf{Dom}(r) \longrightarrow \Gamma$ such that $\mathsf{Dom}(r)$ is a D-tree, where D a finite set of directions. The unranked trees that we shall consider in the paper are also *unordered*; note that the out-degrees of the nodes of $\mathsf{Dom}(r)$ are bounded above by $|D|$.

Outline of the paper. We fix an order-n recursion scheme G and an alternating parity tree automaton (APT) $\mathcal{B}$ over Σ-labelled trees. We introduce the *computation tree* $\lambda(G)$ of the recursion scheme G, which is a Λ_G-labelled tree, in Section 2. We then define *traversal* over a computation tree $\lambda(G)$, and the related notion of (accepting) *traversal-tree* of $\mathcal{B}$ over $\lambda(G)$, and show that $\mathcal{B}$ has an accepting run-tree over $[\![G]\!]$ if and only if $\mathcal{B}$ has an accepting traversal-tree over $\lambda(G)$ (Corollary 8). In Section 3 we present, for a given $\mathcal{B}$, the associated *traversal-simulating* alternating parity automaton over Λ_G-labelled trees, which we call $\mathcal{C}$ throughout the paper. To avoid confusion, we shall refer to $\mathcal{B}$ as the *property APT* (as it corresponds to a property described by some modal mu-calculus formula). In Section 4, we prove that every accepting run-tree of the traversal-simulating APT $\mathcal{C}$ determines an accepting traversal-tree of the property APT $\mathcal{B}$. In Section 5, we prove the opposite direction: Every accepting traversal-tree of a property APT $\mathcal{B}$ determines an accepting run-tree of the associated traversal-simulating APT $\mathcal{C}$. To summarise, we have:

$$
\begin{array}{ll}
 & \text{Property APT } \mathcal{B} \text{ accepts the value tree } [\![G]\!] \\
\iff & \{ \text{ Definition of APT acceptance } \} \\
 & \mathcal{B} \text{ has an accepting run-tree over } [\![G]\!] \\
\iff & \{ \text{ Corollary 8 } \} \\
 & \mathcal{B} \text{ has an accepting traversal-tree over the computation tree } \lambda(G) \\
\iff & \{ \text{ Theorem 20 } \} \\
 & \text{Traversal-simulating APT } \mathcal{C} \text{ has an accepting run-tree over } \lambda(G)
\end{array}
$$

By definition $\lambda(G)$ is the value tree of an order-0 recursion scheme i.e. it is a regular tree. Since the APT Acceptance Problem for regular trees is decidable, we can conclude that the modal mu-calculus model checking problem for trees generated by order-n recursion schemes is decidable. In Section 6 we analyse the complexity of the modal mu-calculus model checking problem. We first establish a certain *succinctness property* (Proposition 19) for accepting run-trees of a traversal-simulating APT. The desired time bound is then obtained by analysing the complexity of solving an appropriate (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).

Acknowledgements

Colin Stirling introduced me to the verification of finitely-presentable infinite structures, and has given me helpful advice over the years. I have benefitted much from numerous discussions with Klaus Aehlig and Jolie de Miranda, comments from Moshe Vardi, and from a close reading of a draft by Andrzej Murawski. I thank them all. Part of the work reported here was done while on sabbatical leave at the Isaac Newton Institute for Mathematical Sciences, Cambridge, for which I thank the Institute, Merton College and the University of Oxford. I gratefully acknowledge the financial support of the Engineering and Physical Science Research Council (in the form of research grants GR/R88861/01 *Algorithmic Game Semantics and its Applications*, and EP/C514645/1 *Pushdown Automata and Game Semantics*).

2. COMPUTATION TREES AND TRAVERSALS

In this section we introduce the computation tree $\lambda(G)$ of a recursion scheme G, and define traversals over the computation tree. A highlight is the correspondence result between maximal paths in the value tree $[\![G]\!]$ and maximal traversals over the computation tree $\lambda(G)$. Finally, given a property APT $\mathcal{B}$ over Σ-labelled trees, we define *traversal-tree*, and explain what it means for a traversal-tree to be accepted by $\mathcal{B}$.

2.1 Long transform $\overline{G}$ and computation tree $\lambda(G)$

As the value tree $[\![G]\!]$ is the "extensional" outcome of a potentially infinite computational process, it has little useful structure for our purpose. Our approach is to consider an auxiliary tree, $\lambda(G)$, which we call the *computation tree* of G. Roughly speaking, evaluating the computation tree (by a potentially infinite process of β-reduction) returns the value tree. Here give a concrete view of how computation trees are constructed. (The more abstract, game-semantic account, is given in an accompanying paper [Ong 2006a]).

The long transform: from G to $\overline{G}$. Fix a recursion scheme G. Rules of the new recursion scheme $\overline{G}$ (which, we shall see, can be regarded as order 0) are obtained from those of G by applying the following four operations in turn, which we call ***long transform***. For each G-rule:

1. *Expand the RHS to its η-long form.* I.e. we hereditarily η-expand every subterm – even if it is of ground type – provided it occurs in an *operand position* (i.e. it is the second argument of some occurrence of the application operator). Note that each applicative term $s \in \mathcal{T}(\Sigma \cup \mathcal{N} \cup \{\, \xi_1, \cdots, \xi_l \,\})$ can be written uniquely as $\dagger\, s_1 \cdots s_m$ where $\dagger$ is either a variable (i.e. some ξ_j) or a non-terminal or a terminal. Suppose $\dagger\, s_1 \cdots s_m : (A_1, \cdots, A_n, o)$. First we define
$$\lceil \dagger\, s_1 \cdots s_m \rceil \quad = \quad \boldsymbol{\lambda}\overline{\varphi}.\dagger\, \lceil s_1 \rceil \cdots \lceil s_m \rceil\, \lceil \varphi_1 \rceil \cdots \lceil \varphi_n \rceil$$
where $\overline{\varphi}$ is a list $\varphi_1 \cdots \varphi_n$ of (fresh) pairwise-distinct variables (which is a null list iff $n = 0$) of types $A_1, \cdots, A_n$ respectively, none of which occurs free in $\dagger\, \lceil s_1 \rceil \cdots \lceil s_m \rceil$. Take any $e = \dagger\, s_1 \cdots s_m$ of ground type. The ***η-long form*** of e is defined to be $\dagger\, \lceil s_1 \rceil \cdots \lceil s_m \rceil$.

   For example the η-long form of $g\, a : o$ is $g\, (\boldsymbol{\lambda}.a)$; we shall see that the "dummy lambda-abstraction"[7] $\boldsymbol{\lambda}.a$ (that binds a *null* list of variable) plays a useful rôle in the syntactic representation of the game semantics of a recursion scheme.
2. *Insert long-apply symbols $@_A$*: Replace each ground-type subterm of the shape $D\, e_1 \cdots e_n$, where $D : (A_1, \cdots, A_n, o)$ is a non-terminal and $n \geq 1$ (i.e. D has order at least 1), by $@_A\, D\, e_1 \cdots e_n$ where $A = ((A_1, \cdots, A_n, o), A_1, \cdots, A_n, o)$. In the following, we shall often omit the type tag A from $@_A$.
3. *Curry the rewrite rule.* I.e. we transform the rule $F\, \varphi_1 \cdots \varphi_n \rightarrow e'$ to
$$F \rightarrow \boldsymbol{\lambda}\varphi_1 \cdots \varphi_n.e'.$$
   In case $n = 0$, note that the curried rule has the form $F \rightarrow \boldsymbol{\lambda}.e'$.
4. *Rename bound variables afresh*, so that any two variables that are bound by different lambdas have different names.

The *computation tree* $\lambda(G)$ is the infinite term-tree that is obtained by unfolding the rewrite rules in the long transform $\overline{G}$ *ad infinitum* (note that no β-redex is contracted in the process). Before presenting the definition, we consider an example.

Example 2.1 (Construction of an order-2 computation tree). We first transform G to its long form $\overline{G}$ by performing the four operations in turn:
$$G : \begin{cases} S \rightarrow F\, H \\ F\, \varphi \rightarrow \varphi\, (F\, \varphi) \\ H\, x \rightarrow f x\, x \end{cases} \quad \mapsto \quad \overline{G} : \begin{cases} S \rightarrow \boldsymbol{\lambda}.@\, F\, (\boldsymbol{\lambda}x.@\, H\, \boldsymbol{\lambda}.x) \\ F \rightarrow \boldsymbol{\lambda}\varphi.\varphi(\boldsymbol{\lambda}.@\, F\, (\boldsymbol{\lambda}y.\varphi(\boldsymbol{\lambda}.y)))) \\ H \rightarrow \boldsymbol{\lambda}z.f(\boldsymbol{\lambda}.z)(\boldsymbol{\lambda}.z) \end{cases}$$
The computation tree $\lambda(G)$, as presented in Figure 1, is the term-tree that is obtained by infinitely unfolding the $\overline{G}$-rules from S. For another example of the long transform and computation tree of an order-2 recursion scheme, see Example 2.6. □

Definition 2.2. For any recursion scheme G, the system of transformed rules in $\overline{G}$ defines an order-0 recursion scheme – called the ***long transform*** of G – 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
$$\underbrace{\Sigma \;\cup\; \mathit{Var} \;\cup\; \{\, @_A : A \in \mathit{ATypes} \,\}}_{\textbf{\textit{Non-lambdas}}} \quad \cup \quad \underbrace{\{\, \boldsymbol{\lambda}\overline{\xi} : \overline{\xi} \subseteq \mathit{Var} \,\}}_{\textbf{\textit{Lambdas}}}$$
where *ATypes* is the set of types of the shape $((A_1, \cdots, A_n, o), A_1, \cdots, A_n, o)$ with $n \geq 1$. We rank the symbols in Λ_G as follows:

[7] To my knowledge, Colin Stirling was the first to use a tree representation of lambda terms in which "dummy lambdas" are employed; see his CSL 2005 paper [Stirling 2005]. Motivated by property-checking games in Verification, he has introduced a game that is played over such trees as a characterization of higher-order matching [Stirling 2006].

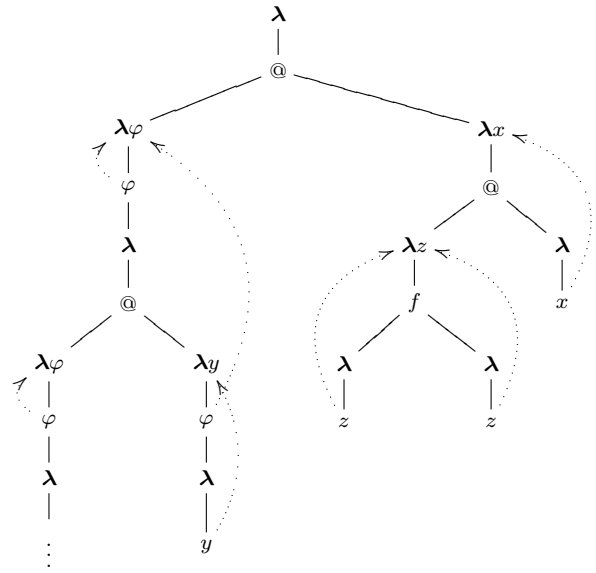


Fig. 1. An example of an order-2 computation tree.

—variable symbol $\varphi : (A_1, \cdots, A_n, o)$ in *Var* has arity n
—long-apply symbol $@_A$ where $A = ((A_1, \cdots, A_n, o), A_1, \cdots, A_n, o)$ has arity $n + 1$
—lambda symbol $\boldsymbol{\lambda}\overline{\xi}$ has arity 1, for every list of variables $\overline{\xi} \subseteq Var$.

Further, for $f \in \Lambda_G$, we define

$$\mathsf{Dir}(f) \;=\; \begin{cases} [ar(@_A) - 1]_0 & \text{if } f = @_A \\ [ar(f)] & \text{otherwise} \end{cases}$$

For technical reasons (to be clarified shortly), the leftmost child of an @-labelled node α is in direction 0 (i.e. it is α's 0-child); for all other nodes, the leftmost child is in direction 1. The *non-terminals* of $\overline{G}$ are exactly those of G, except that each is assigned a new type, namely, o. We can now define the ***computation tree*** $\lambda(G)$ to be the value tree $[\![\,\overline{G}\,]\!]$ of the order-0 recursion scheme $\overline{G}$. It follows that $\lambda(G)$ is a regular tree.

Notation. Sometimes, for convenience, we shall refer to a node α of a labelled tree t by its label $t(\alpha)$.

We emphasize that there is *no renaming* of bound variables (indeed there is *no need* as no β-redex is ever contracted and hence no substitution is ever performed) in the construction of the computation tree $\lambda(G)$. We shall call a node of $\lambda(G)$ a *lambda node* (respectively *non-lambda node*) if it is labelled by a lambda (respectively non-lambda) symbol; similarly, a *variable node* (respectively *@-node*) is a node that is labelled by a variable (respectively long-apply symbol). Note that by construction even-level nodes (we say that the root is at level 0) of a computation tree are non-lambda nodes, odd-level nodes are lambda nodes. Suppose the node α is labelled by a variable ξ; we say that α is ***bound*** by the the node β (equivalently β is the ***binder*** of α) just in case β is the largest prefix of α that is labelled by a lambda symbol $\boldsymbol{\lambda}\overline{\xi}$, for some list $\overline{\xi}$ that contains ξ. The dotted arrows in Figure 1 indicate the binding relationship.

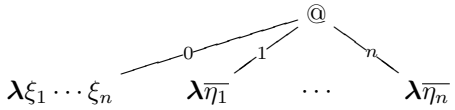
REMARK 2.3. (i) Even though variables φ and long-apply symbols $@_A$ (and, as we shall see shortly, lambda symbols $\boldsymbol{\lambda}\overline{\varphi}$) are intrinsically typed, we stress that when viewed as elements of the order-0 recursion scheme $\overline{G}$, they are just *terminals* of the ranked alphabet Λ_G; hence – when considered in this rôle – they are defined to have an arity, which in turn determines a type, of order at most one. To distinguish the two types, we shall refer to the former as *intrinsic type* and the later as *arity type*.

(ii) Another way to present the computation tree $\lambda(G)$ is to define it as the *unfolding* of a Λ_G*-labelled (finite) deterministic digraph* which is an abstract representation of the associated order-0 recursion scheme

$\overline{G}$ (Proposition 22). We shall make use of this characterization in the complexity analysis of the model checking problem.

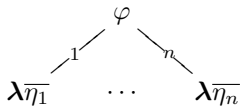
Local structure of a computation tree. The local structure of a computation tree $\lambda(G)$ is determined by the node-labels:

—A node labelled by a long-apply symbol $@_A : A$ (where $A = ((A_1, \cdots, A_n, o), A_1, \cdots, A_n, o)$) has $n+1$ children:



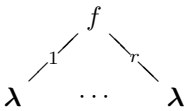
where for each i, we have $\xi_i : A_i = (C_{i1}, \cdots, C_{ir_i}, o)$ and $\eta_{ij} : C_{ij}$. By abuse of language, we say that the symbol $\boldsymbol{\lambda}\xi_1 \cdots \xi_n$ has type $(A_1, \cdots, A_n, o)$ (hence the "dummy lambda" $\boldsymbol{\lambda}$ is of ground type o). Thus it follows that ξ_i and $\boldsymbol{\lambda}\overline{\eta_i}$ have the same type, namely, A_i.

—A node labelled by a variable $\varphi : (A_1, \cdots, A_n, o)$ has n children:



where for each i, we have $\boldsymbol{\lambda}\overline{\eta_i} : A_i$. In case $n = 0$, φ labels a terminal node.

—A node labelled by a Σ-symbol $f : (\underbrace{o, \cdots, o}_{r}, o)$, where $r = ar(f)$, has r children:



In case $r = 0$, f labels a terminal node.

In the above diagrams, the edges are labelled by their respective directions. Note the anomaly: The leftmost child of a variable or Σ-symbol node is (in direction 1, and is thus referred to as) its 1-child, but that of an @-node is (in direction 0 and referred to as) its 0-child. (This has the desirable consequence that for $1 \leq i \leq n$, the type of the label – $\boldsymbol{\lambda}\overline{\eta_i}$ in the figure above – of the i-child of @ coincides with the type of the i-th variable ξ_i bound by the lambda that labels the 0-child of @.) Thus if the node α is labelled by @, its leftmost child is the node $\alpha\,0$. Every node α labelled by a lambda symbol has precisely one child, namely, $\alpha 1$.

We define a binary relation over the node-set of a computation tree $\lambda(G)$, called ***enabling***, as follows (we read $m \vdash_i m'$ as "m i-enables m'", or "m' is i-enabled by m", or "m is the *enabler* of m'")

—Every lambda node, except the root, is *i-enabled* by its parent node in $\lambda(G)$, where the former is the i-child of the latter.

—Every variable node (labelled by ξ_i, say) is *i-enabled* by its *binder* (labelled by $\boldsymbol{\lambda}\overline{\xi}$, say), where ξ_i is the i-th element of the list $\overline{\xi}$.

We say that a node α is *enabled* by β, written $\beta \vdash \alpha$, if for some (necessarily unique) i, we have $\beta \vdash_i \alpha$. We say that a node of $\lambda(G)$ is ***initial*** if it is not enabled by any node. If follows from the definition that the initial nodes of a computation tree are the root-node (necessarily labelled by the lambda symbol $\boldsymbol{\lambda}$), and all nodes labelled by a long-apply or Σ-symbol.

Remark 2.4. We can think of a computation tree as a tree that is constructed by assembling (possibly infinitely many copies of) finite basic parts called *extents*, which are given by the respective RHSs (but less the non-terminals) of the $\overline{G}$-rules. A node in a computation tree is said to be ***prime*** just if it is the leftmost child (i.e. 0-child) of an @-node. Formally an ***extent*** of $\lambda(G)$ is a subtree that is identified by its *root* α, which must be prime; nodes of the extent are exactly those descendants β of α (in $\lambda(G)$) such that α is the only prime node in the path from α to β. Thus a leaf of an extent is labelled by either a nullary Σ-symbol or an order-0 variable. We state the following observations:

(1) Every node in a computation tree, except those from a finite initial subtree, is in a unique extent.
(2) In any given computation tree, there are only finitely many non-isomorphic (with reference to labelled digraphs) extents, as given by the RHSs of the $\overline{G}$-rules but less the non-terminals. (Thus computation trees are regular.) Hence the extents are bounded in size.
(3) Every node in a computation tree is enabled by some node in the same extent. In particular, every variable node is in the same extent as its binder.
(4) Suppose the node γ is a descendant of both an @-node α and its 0-child $\alpha 0$ (i.e. $\alpha 0 \leq \gamma$), and suppose γ is hereditarily enabled by the @-node γ_0. Then $\alpha \leq \gamma_0$. (This is a straightforward corollary of (3)).

2.2 Justified sequences and P-views

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 n that occurs in it has a pointer to some earlier node-occurrence n_0 in the sequence such that $n_0 \vdash_j n$, for some j. We say that the node-occurrence n is ***justified*** by the node-occurrence n_0 in the sequence.

Notation. $\cdots\ n_0 \overset{j}{\leftarrow} \cdots\ n\ \cdots$ means that n points to n_0, and $n_0 \vdash_j n$ holds in $\lambda(G)$. We say that n is *j-justified* by n_0, or n has a *j-pointer* to n_0.

The notion of view (of a justified sequence) and the condition of Visibility were first introduced in game semantics [Hyland and Ong 2000]. Intuitively the *P-view* (respectively *O-view*) of a justified sequence is a certain subsequence consisting of moves which P (respectively O) considers relevant for determining his next move in the play. In our setting, the lambda nodes are the O-moves, and the non-lambda moves are the P-moves.

Definition 2.5. 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

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

In the second clause above, if in $t\ n \overset{i}{\leftarrow} \cdots\ \boldsymbol{\lambda}\overline{\xi}$ the non-lambda node n has a pointer to some node-occurrence l (say) in t, and if l appears in $\ulcorner t \urcorner$, then in $\ulcorner t \urcorner\ n \overset{i}{\leftarrow} \boldsymbol{\lambda}\overline{\xi}$ the node n is defined to point to l; otherwise n has no pointer. Similarly, in the third clause above, if in $t\, \boldsymbol{\lambda}\overline{\xi}\, n$ the non-lambda node n has a pointer to some node-occurrence l (say) in $t\, \boldsymbol{\lambda}\overline{\xi}$, and if l appears in $\ulcorner t\, \boldsymbol{\lambda}\overline{\xi} \urcorner$, then in $\ulcorner t\, \boldsymbol{\lambda}\overline{\xi} \urcorner\, n$ the node n is defined to point to l; otherwise n has no pointer. It is easy to see that the P-view of a justified sequence is always alternating, but it does not necessarily satisfy the pointer condition.

Dually the ***O-view***, $\llcorner t \lrcorner$, of a justified sequence t is a subsequence defined by recursion as follows:

$$\begin{aligned} \llcorner \epsilon \lrcorner &= \epsilon \\ \llcorner t\, \boldsymbol{\lambda}\overline{\xi} \lrcorner &= \llcorner t \lrcorner\, \boldsymbol{\lambda}\overline{\xi} \\ \llcorner t\ \boldsymbol{\lambda}\overline{\xi} \overset{i}{\leftarrow} \cdots\ \xi \lrcorner &= \llcorner t \lrcorner\ \boldsymbol{\lambda}\overline{\xi} \overset{i}{\leftarrow} \xi \\ \llcorner t\, n \lrcorner &= n \qquad \text{where } n \text{ is labelled by a long-apply or } \Sigma\text{-symbol} \end{aligned}$$

In the second clause above, if in $t\, \boldsymbol{\lambda}\overline{\xi}$ the lambda node $\boldsymbol{\lambda}\overline{\xi}$ has a pointer to some node-occurrence n' in t, and if n' appears in $\llcorner t \lrcorner$, then in $\llcorner t \lrcorner\, \boldsymbol{\lambda}\overline{\xi}$ the node $\boldsymbol{\lambda}\overline{\xi}$ is defined to point to n'; otherwise it 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 is justified by some (necessarily lambda) node that appears in the P-view at that point: Formally, for each prefix $u\, m \leq t$, if m is a non-initial non-lambda node that points to

some node-occurrence l in u, then l appears in $\ulcorner u \urcorner$. Dually a justified sequence t satisfies ***O-visibility*** just in case every non-initial lambda node that occurs in the sequence is justified by some (necessarily non-lambda) node that appears in the O-view at that point. We say that a justified sequence satisfies ***Visibility*** if it satisfies both P- and O-visibility.

We say that a node-occurrence m is ***hereditarily justified*** by a node-occurrence n in (the justified sequence) t if there are node-occurrences $n_1 = n, \cdots, n_{l+1} = m$ in t such that n_{i+1} points to n_i, for each $1 \leq i \leq l$.

Example 2.6 (Views and Visibility). Let G be the following (unsafe) order-2 recursion scheme:

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

We present its computation tree $\lambda(G)$ in Figure 2.

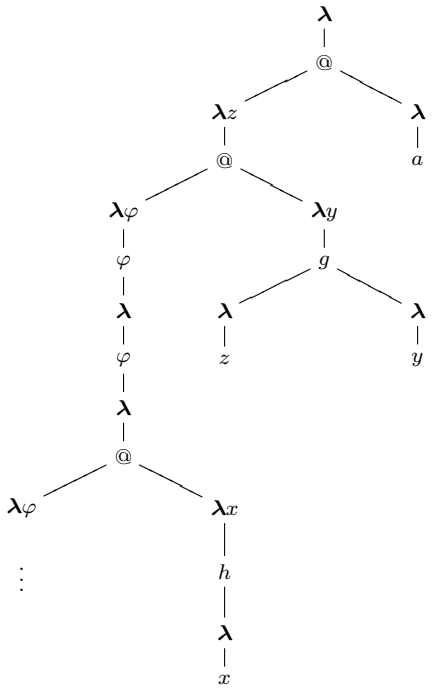
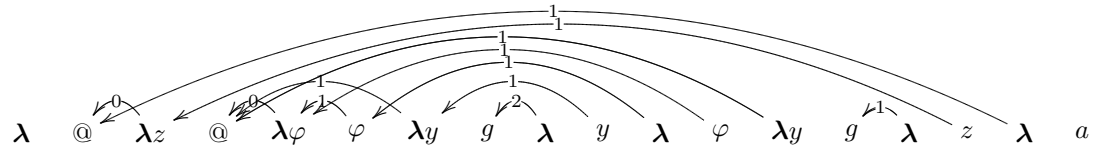


Fig. 2. An order-2 computation tree.

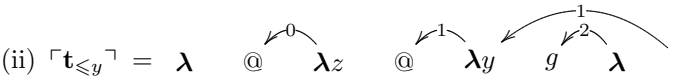
Consider the following justified sequence $\mathbf{t}$ over the computation tree in Figure 2:



We write $\mathbf{t}_{\leqslant m}$ to mean the prefix of $\mathbf{t}$ truncated at (and including) the node m. Let α be the node $\epsilon\,1\,0\,1\,1\,1\,1$ in $\lambda(G)$ (which has label $\boldsymbol{\lambda}$). We have:

(i) $\ulcorner \mathbf{t}_{\leqslant \alpha} \urcorner = \boldsymbol{\lambda}\ @\ \boldsymbol{\lambda}z\ @\ \boldsymbol{\lambda}y\ g\ \boldsymbol{\lambda}$.

The justified sequence $\mathbf{t}$ satisfies P-Visibility: Take the node-occurrence z in t; we have z is justified by $\boldsymbol{\lambda}z$, which appears in the P-view at that point, namely, $\ulcorner \mathbf{t}_{\leqslant \alpha} \urcorner$ (see above).

(ii) $\ulcorner \mathbf{t}_{\leqslant y} \urcorner =$ 

(iii) $\llcorner \mathbf{t}_{\leqslant y} \lrcorner =$ 

LEMMA 4. *If a justified sequence satisfies Visibility, then its P-view and O-view are well-defined justified sequences that satisfy Visibility.*

PROOF. This is a standard result in (innocent) game semantics. See e.g. [Hyland and Ong 2000] for a proof. □

We shall call a justified sequence s (that satisfies Visibility) a ***P-view*** if it is a fixpoint of the P-view operation i.e. $\ulcorner s \urcorner = s$. Equivalently a justified sequence is a P-view if (and only if) every occurrence of a non-initial lambda-node points to the node that immediately precedes it.

LEMMA 5. *For every recursion scheme G, each path in the computation tree $\lambda(G)$ is a justified sequence that satisfies P-visibility and O-visibility, and it is a P-view.*

PROOF. An easy exercise. □

2.3 Traversals over a computation tree

DEFINITION 2.7. ***Traversals*** over a computation tree $\lambda(G)$ are justified sequences of nodes defined by induction over the following rules: we let f range over Σ-symbols, and n range over non-lambda symbols

(Root)	The singleton sequence, comprising the root node ϵ, is a traversal.
(App)	If $t\,@$ is a traversal, so is $t\,@\overset{0}{\leftarrow}\boldsymbol{\lambda}\overline{\xi}$.
(Sig)	If $t\,f$ is a traversal, so is $t\,f\overset{i}{\leftarrow}\boldsymbol{\lambda}$ for each $1 \leq i \leq \mathit{arity}(f)$.
(Var)	If $t\,n\,\boldsymbol{\lambda}\overline{\xi}\overset{i}{\leftarrow}\cdots\ \xi$ is a traversal, so is $t\,n\overset{i}{\leftarrow}\boldsymbol{\lambda}\overline{\xi}\overset{i}{\leftarrow}\cdots\ \xi\ \boldsymbol{\lambda}\overline{\eta}$.
(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.

In the above rules, for ease of reading, we refer to nodes of a computation tree by their labels. For example, the rule (Var) says that if $t\,\alpha\,\beta\overset{i}{\leftarrow}\cdots\ \beta i$ is a traversal such that the node βi is labelled by a variable ξ (say), then $t\,\alpha\overset{i}{\leftarrow}\beta\overset{i}{\leftarrow}\cdots\ \beta i\ \alpha i$ is also a traversal. (Of course we need to show that the node α in the computation tree has an i-child, which is a lambda node. We shall do this in Proposition 6.) We say that an infinite justified sequence (over a computation tree) is a *traversal* if every finite prefix of it is a traversal.

It follows from the definition that the way that a traversal can grow is deterministic (and determined by $\lambda(G)$), except when the last node is an order-1 Σ-symbol $f : o^k \to o$, in which case, the traversal can grow in one of k possible directions in the next step.

PROPOSITION 6. *(Traversals are assumed to be finite in this Proposition.)*

(i) Traversals are well-defined justified sequences that satisfy Visibility.

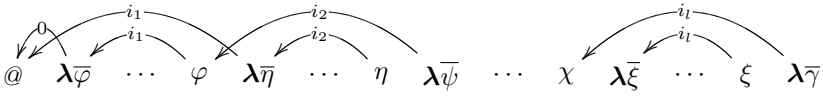
(ii) The P-view of a traversal is a path in the computation tree.

(iii) An odd-length traversal has the shape

$$\epsilon\ \underbrace{\alpha_1 \cdots \beta_1}_{u_1}\ \cdots\ \underbrace{\alpha_n \cdots \beta_n}_{u_n}$$

*where $n \geq 0$, and in each u_i (which we shall call a **basic segment**), the last node β_i, which is labelled by a lambda, is hereditarily justified by the first node α_i, which is labelled by an initial non-lambda symbol.*

The first node α_i of a basic segment u_i is labelled by either a Σ-symbol (in which case, the segment u_i has length 2), or by a long-apply symbol, in which case, u_i has the following shape:



(1)

*where $l \geq 0$, and the pair φ and $\boldsymbol{\lambda}\overline{\eta}$ have the same type, and similarly for the pairs η and $\boldsymbol{\lambda}\overline{\psi}$, $\cdots$, and ξ and $\boldsymbol{\lambda}\overline{\gamma}$ (so that exactly one of the last node $\boldsymbol{\lambda}\overline{\gamma}$ and the penultimate node ξ is hereditarily justified by $\boldsymbol{\lambda}\overline{\varphi}$, though both are hereditarily justified by the first node @). We call segments of the form (1) **(even-length) @-blocks**.*

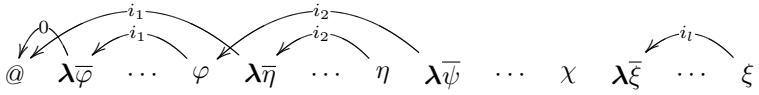
An even-length traversal is either of the shape

$$\epsilon \underbrace{\alpha_1 \cdots \beta_1}_{u_1} \cdots \underbrace{\alpha_n \cdots \beta_n}_{u_n} \alpha$$

where $n \geq 0$, each u_i is a basic segment and α is an initial non-lambda node; or it is of the shape

$$\epsilon \underbrace{\alpha_1 \cdots \beta_1}_{u_1} \cdots \underbrace{\alpha_n \cdots \beta_n}_{u_n} \underbrace{\cdots}_{u}$$

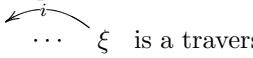
where $n \geq 0$, each u_i is a basic segment and u has the shape

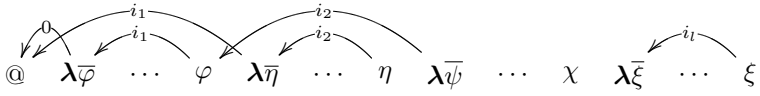


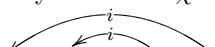
(2)

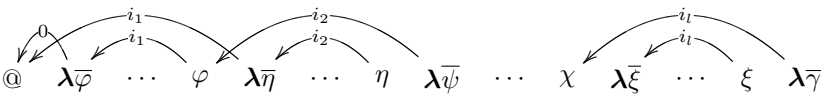
*where $l \geq 1$, and the pair φ and $\boldsymbol{\lambda}\overline{\eta}$ have the same type, and similarly for the pairs η and $\boldsymbol{\lambda}\overline{\psi}$, $\cdots$, and χ and $\boldsymbol{\lambda}\overline{\xi}$ (so that exactly one of the last node ξ and χ is hereditarily justified by the node $\boldsymbol{\lambda}\overline{\varphi}$, though both are hereditarily justified by the first node @). We call segments of the form (2) **(odd-length) @-blocks**.*

Proof. We prove the Proposition by simultaneous rule induction. For each rule, we show that if the sequence in the premise, and all its prefixes, are justified sequences that satisfy (i), (ii) and (iii), then the same is true of the justified sequence in the conclusion. We consider the last two rules.

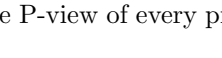
(*Var*): Suppose $s = t\, n$  is a traversal. By the induction hypothesis of (iii), s has the shape $\epsilon\, u_1 \cdots u_r\, u$ where each u_i is a basic segment and u is an odd-length @-block as follows



for some $l \geq 1$ and $i = i_l$ (so that n is the node χ). Since χ and $\boldsymbol{\lambda}\overline{\xi}$ have the same type, there is a lambda-node ($\boldsymbol{\lambda}\overline{\gamma}$ say) that is i_l-enabled by the node χ. By the induction hypothesis of (i), s and all its prefixes satisfy Visibility. Hence $t\, \chi$  is a well-defined odd-length traversal that satisfies O-visibility (and hence Visibility), and has the shape $\epsilon\, u_1 \cdots u_r\, u\, \boldsymbol{\lambda}\overline{\gamma}$ where

$u\, \boldsymbol{\lambda}\overline{\gamma} =$ 

is an even-length @-block as required. Thus we have shown (i) and (iii).

By the induction hypothesis of (ii), the P-view of every prefix of s is a path in the computation tree. In particular we have $\ulcorner s_{\leqslant \chi} \urcorner$ is a path in the computation tree. Since $\ulcorner t\, \chi$  $\urcorner = \ulcorner s_{\leqslant \chi} \urcorner \boldsymbol{\lambda}\overline{\gamma}$, it is also a path in the computation tree.

(*Lam*): Take a traversal of the form $s = t\, \boldsymbol{\lambda}\overline{\xi}$. By the induction hypotheses of (i) and (ii), $t\, \boldsymbol{\lambda}\overline{\xi}$ satisfies P-visibility, and $\ulcorner t\, \boldsymbol{\lambda}\overline{\xi} \urcorner$ is a path in the computation tree. By definition of the computation tree, there

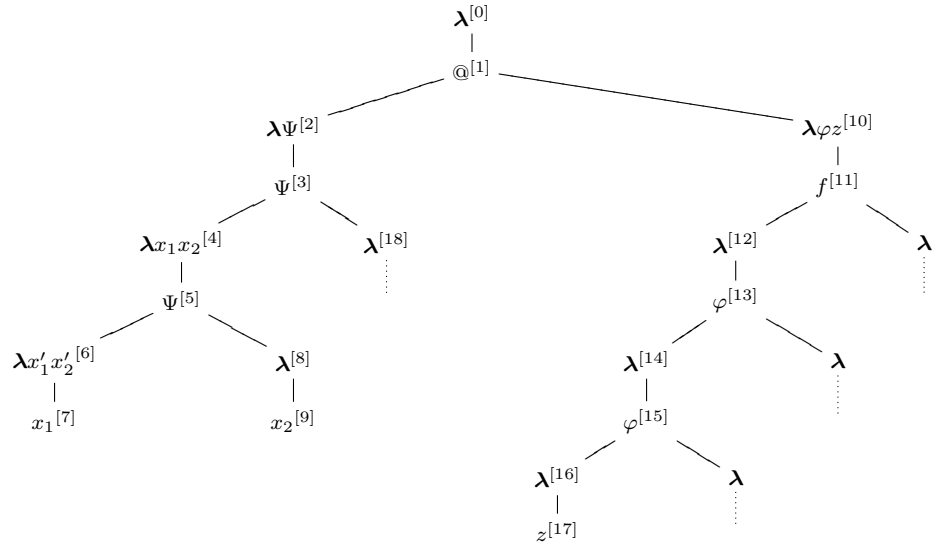


Fig. 3. An example of an order-3 computation tree.

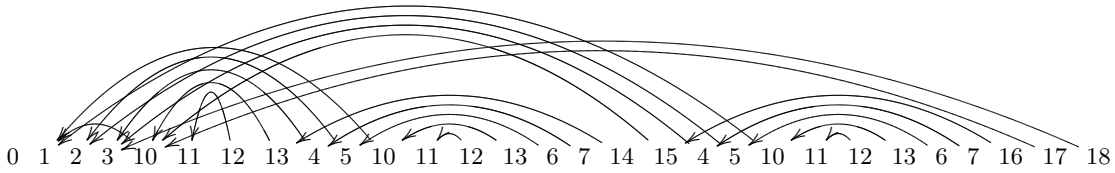
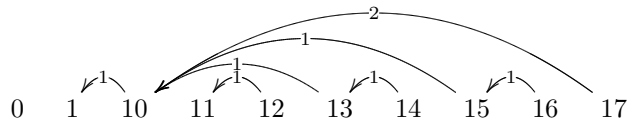


Fig. 4. Order-3 traversals can be quite complex!

is a well-defined node n such that $\ulcorner t\,\boldsymbol{\lambda}\overline{\xi}\urcorner n$ is a path in the tree. Since $\ulcorner t\,\boldsymbol{\lambda}\overline{\xi}\,n\urcorner = \ulcorner t\,\boldsymbol{\lambda}\overline{\xi}\urcorner n$, it is a path in the tree. It follows from Lemma 5 that $\ulcorner t\,\boldsymbol{\lambda}\overline{\xi}\,n\urcorner$ satisfies P-visibility. Thus $t\,\boldsymbol{\lambda}\overline{\xi}\,n$ satisfies P-visibility (and hence Visibility). For (iii), by the induction hypothesis, the odd-length traversal s has the shape $\epsilon\,u_1\cdots u_r$ where each u_i is a basic segment and u_r is an even-length @-block. Suppose the non-lambda node n is justified by a lambda node γ in u_i (say). The case where n is labelled by a long-apply or a Σ-symbol is trivial. We assume that n is labelled by a variable. Then it follows that $t\,\boldsymbol{\lambda}\overline{\xi}\,n$ has the shape $\epsilon\,u_1\cdots u_{i-1}\,\underbrace{u_i\cdots u_r\,n}_{u}$ where u is an odd-length @-block, as required. □

A corollary of the Proposition is that every finite traversal t can be extended (in the sense that $t\,\alpha$ is a traversal, for some node α), except when the last node of t is labelled by a nullary Σ-symbol. Thus a maximal traversal is either infinite or it ends in a node labelled by a nullary Σ-symbol.

Example 2.8 (Traversals). The justified sequence in Example 2.6 is a traversal over an order-2 computation tree. For an order-3 example, consider the computation tree in Figure 3. For ease of reference, we give nodes of the computation tree $\lambda(G)$ numeric names, which are indicated (within square-brackets) as superscripts. See Figure 4 for a traversal over the computation tree. The P-view of the traversal in Figure 4 truncated at node 17, $\ulcorner 0\cdots 17\urcorner$, is the following path from the root to node 17:



□

REMARK 2.9. The pointers in a traversal over any computation tree $\lambda(G)$ are uniquely reconstructible from the underlying sequence of nodes *and their respective labels*; thus pointers are not an additional structure imposed on the underlying sequence. However it is convenient (e.g. in the definition of view) to define traversals as sequences equipped with pointers. Another advantage of pointers is that they help to clarify the correspondence between traversals and *interaction sequences* (that arise in the construction of the game semantics of the recursion scheme in question) - on which, more anon.

2.4 Traversal-trees of a property alternating parity automaton $\mathcal{B}$

We first state an important theorem that underpins our approach.

THEOREM 7 (PATH-TRAVERSAL CORRESPONDENCE). *Let G be a recursion scheme.*

(i) There is a 1-1 correspondence, $p \mapsto t_p$, between maximal paths p in the Σ-labelled (value) tree $[\![G]\!]$ and maximal traversals t_p in the Λ_G-labelled (computation) tree $\lambda(G)$.

(ii) Further for every maximal path p in $[\![G]\!]$, we have

$$t_p \upharpoonright \Sigma^- \quad = \quad p \upharpoonright \Sigma^-$$

where $t_p \upharpoonright \Sigma^-$ denotes the subsequence of t_p consisting of only Σ^--symbols where $\Sigma^- = \Sigma \setminus \{\perp\}$. Note that $\perp \in \Sigma$, but $\perp \notin \Lambda_G$.

Using the language of game semantics, we are claiming, in (ii), that the traversal t_p is the uncovering *of p.*

PROOF. The proof, which is by (innocent) game semantics, is in an accompanying paper [Ong 2006a]. □

We illustrate the Theorem by the following example.

EXAMPLE 2.10. Consider the order-2 recursion scheme G as defined in Example 2.1. The value tree $[\![G]\!]$ and the auxiliary computation tree $\lambda(G)$ are displayed in Figure 5. For ease of reference, we give nodes

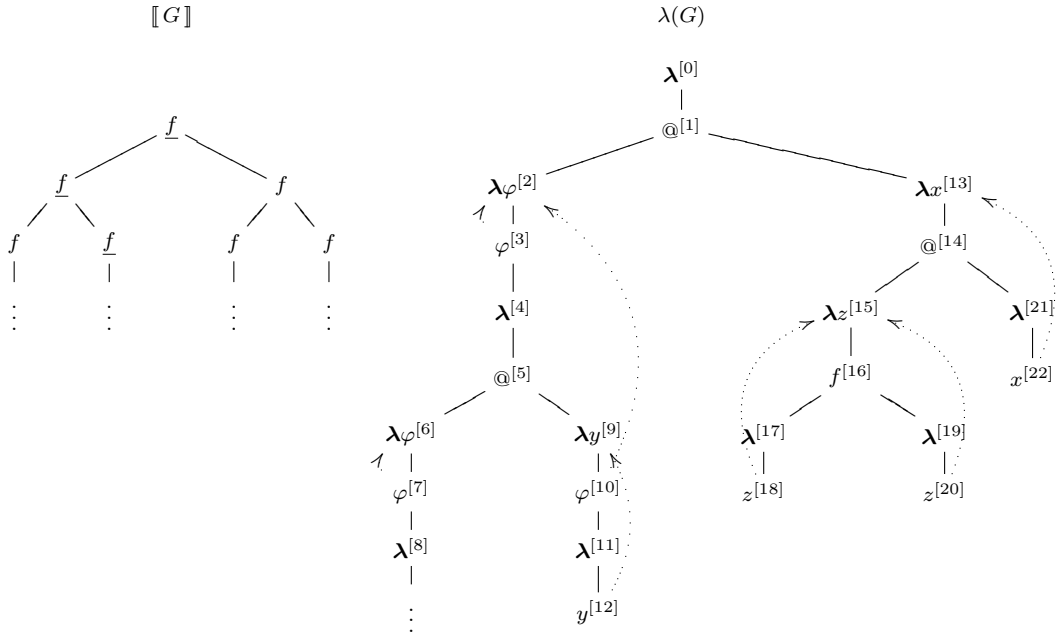


Fig. 5. Correspondence between paths in the value tree and traversals over a computation tree.

of the computation tree $\lambda(G)$ numeric names, which are indicated (within square-brackets) as superscripts. With reference to Figure 5, take the path $\epsilon \cdot 1 \cdot 12 \cdots$ in the value tree $[\![G]\!]$ (the first few nodes in the path are underlined in the figure). The traversal (over the computation tree $\lambda(G)$) that corresponds to it is

$$0 \cdot 1 \cdot 2 \cdot 3 \cdot 13 \cdot 14 \cdot 15 \cdot 16 \cdot 17 \cdot 18 \cdot 21 \cdot 22 \cdot 4 \cdot 5 \cdot 6 \cdot 7 \cdot 9 \cdot 10 \cdot 13 \cdot 14 \cdot 15 \cdot 16 \cdot 19 \cdot 20 \cdot 21 \cdot 22 \cdot 11 \cdot 12 \cdot 8 \cdots$$

Further there is a correspondence between prefixes of the path on the one hand, and prefixes of the traversal that end in a Σ-symbol, on the other:

—ϵ corresponds to $0 \cdot 1 \cdot 2 \cdot 3 \cdot 13 \cdot 14 \cdot 15 \cdot 16$
—$\epsilon\,1$ corresponds to $0 \cdot 1 \cdot 2 \cdot 3 \cdot 13 \cdot 14 \cdot 15 \cdot 16 \cdot 17 \cdot 18 \cdot 21 \cdot 22 \cdot 4 \cdot 5 \cdot 6 \cdot 7 \cdot 9 \cdot 10 \cdot 13 \cdot 14 \cdot 15 \cdot 16$
—$\epsilon \cdot 1 \cdot 12$ corresponds to

$$0 \cdot 1 \cdot 2 \cdot 3 \cdot 13 \cdot 14 \cdot 15 \cdot 16 \cdot 17 \cdot 18 \cdot 21 \cdot 22 \cdot 4 \cdot 5 \cdot 6 \cdot 7 \cdot 9 \cdot 10 \cdot 13 \cdot 14 \cdot 15 \cdot 16 \cdot 19 \cdot 20 \cdot 21 \cdot 22 \cdot 11 \cdot 12 \cdot 8 \cdots 16$$

—etc. □

Let G be a recursion scheme. Take 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 2.11. A ***traversal-tree*** of a property APT $\mathcal{B}$ over a Λ_G-labelled tree $\lambda(G)$ is a (unranked) $(\mathsf{Dom}(\lambda(G)) \times Q)$-labelled $[m]$-tree t, where $m = ar(\Sigma) \times |Q|$, 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 $1 \leq j \leq m$ 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 $t(\beta_1) = (\gamma, q_1)$.

$\cdots$	ψ or @	$\boldsymbol{\lambda}\overline{\xi}$	$\cdots$	ξ	$\boldsymbol{\lambda}\overline{\varphi}$	$\lambda(G)(\alpha)$	$\in$	Λ_G
	(γ, q_1)	(α_1, q_1)		(α, q)	$(\gamma\,i, q)$	$t(\beta) = (\alpha, q)$	$\in$	$\mathsf{Dom}(\lambda(G)) \times Q$
	β_1	$\beta_1\,1$		β	$\beta\,1$	β	$\in$	$\mathsf{Dom}(t)$

Note that $\gamma\,i$ is a lambda node that is i-justified by γ which is labelled by either a long-apply symbol or a variable.
—If $\lambda(G)(\alpha)$ is a lambda, then $t(\beta\,1) = (\alpha\,1, q)$.

A traversal-tree t is *accepting* if all infinite paths $(\alpha_0, q_0)\,(\alpha_1, q_{i_1})\,(\alpha_2, q_{i_2}) \cdots$ through it satisfy the parity acceptance condition, namely, $\limsup \langle\, \Omega(q_{i_j}) : j \geq 0 \,\rangle$ 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 2.12 (Running). Let the input ranked alphabet Σ be $\{\, a : o,\ h : o \to o,\ g : o^2 \to o \,\}$, and let G be the following (unsafe) order-2 recursion scheme:

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

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:

$$\delta\ :\ \begin{cases} (1, g) \ \mapsto\ ((1,1) \wedge (2,1))\ \vee\ ((1,2) \wedge (2,1)) \\ (1, a) \ \mapsto\ \mathsf{true} \\ (2, a) \ \mapsto\ \mathsf{true} \end{cases}$$

In Figure 6, we present the computation tree $\lambda(G)$ on the left, and a traversal-tree of $\mathcal{B}$ over $\lambda(G)$ on the right.

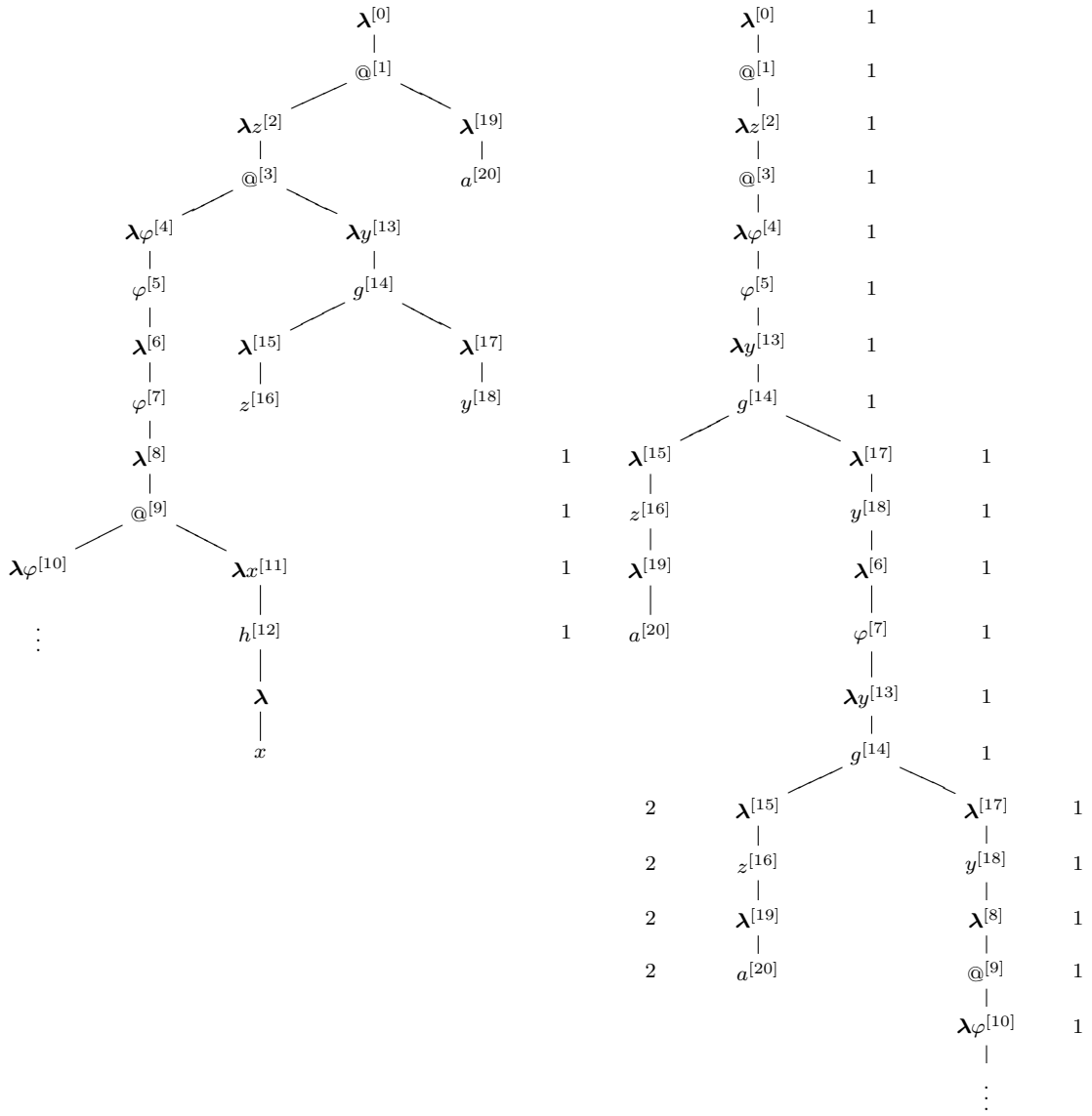


Fig. 6. A computation tree and a traversal-tree of an APT over it.

We state a straightforward consequence of Theorem 7 as follows.

COROLLARY 8. *Let G be a recursion scheme. For every APT $\mathcal{B}$ over Σ-labelled trees, there is a one-one correspondence between*

(1) accepting run-trees of $\mathcal{B}$ over the value tree $[\![G]\!]$

(2) accepting traversal-trees of $\mathcal{B}$ over the computation tree $\lambda(G)$. □

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

3. THE TRAVERSAL-SIMULATING ALTERNATING PARITY AUTOMATON $\mathcal{C}$

In this section, we show how traversals over a computation tree can be simulated by paths in the tree. We first give an informal explanation of the simulation and then present the definition of the traversal-simulating APT.

3.1 How to simulate accepting traversal-trees? An informal explanation

Our task is 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 traversals can jump all over the computation tree and visit some nodes infinitely often. As we have seen, in Example 2.8, traversals can be very complex indeed. Our idea is to exploit the fact that the P-view of a traversal is a path in the computation tree (Proposition 6). 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 shall 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 $\mathcal{B}$.

An order-2 illustration. 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 Figure 7 illustrates:

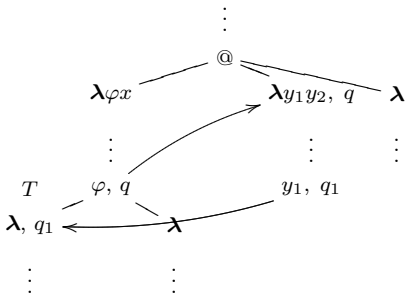


Fig. 7. an order-2 illustration.

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

—When reading the node labelled φ with simulating state q, we *guess* that the detour to $\boldsymbol{\lambda} y_1 y_2$ will return to the 1-child of the φ-labelled node with simulating state q_1; the automaton proceeds by descending in the direction of 1.

—In order to verify the guess, we *spawn* an automaton at the root of the subtree that denotes the actual parameter of φ (i.e. the node labelled by $\boldsymbol{\lambda} y_1 y_2$). The automaton then proceeds by descending the subtree.

Thus, in general, at a node α that is labelled by an @, 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

(i) $q \in Q$ is the state that is being simulated when a ξ_i-labelled node (a descendent of the node $\alpha 0$ labelled $\boldsymbol{\lambda}\xi_1 \cdots \xi_n$) is met by the descending automaton, simulating the traversal

(ii) $m \in [p]$ is the maximal priority that will have been seen at that point, since reading the node labelled by $\boldsymbol{\lambda}\xi_1 \cdots \xi_n$

(iii) 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 with respect to the property APT $\mathcal{B}$, describes the manner (i.e. with what simulating state, and at which child of ξ_i) in which the traversal, which now jumps to a neighbouring subtree denoting the actual parameter of ξ_i, will eventually return to (the original subtree) through some children of the ξ_i-labelled node.

Take, for example, the situation in Figure 7 and consider the profile (φ, q, m, c): The formal parameter is $\varphi : (o, o, o)$ and its actual parameter is denoted by a subtree whose root is labelled by $\boldsymbol{\lambda} y_1\, y_2$, and

$$c \quad = \quad \{\, (y_1, q_1, m_1, \varnothing), (y_1, q_2, m_2, \varnothing)\,\}.$$

The interface c in the profile (φ, q, m, c) for φ says, among other things, that the traversal that enters the subtree denoting the actual parameter of φ with simulating state q will evolve in such a way that one extension – corresponding to $(y_1, q_1, m_1, \varnothing)$ – will exit the subtree through the variable y_1 with simulating state q_1 such that m_1 is the maximal priority seen since reading $\boldsymbol{\lambda} y_1 y_2$, and another – corresponding to $(y_1, q_2, m_2, \varnothing)$ – will exit through y_1 with simulating state q_2 such that m_2 is the maximal priority seen.

3.2 Variable profiles and environment

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

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

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

Definition 3.1. (i) The set $\mathbf{VP}_G^{\mathcal{B}}(A)$ of ***variable profiles*** (or *profiles* for short) of type A for $\lambda(G)$ relative to $\mathcal{B}$ are defined inductively as follows:

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

We say that $(\xi, q, m, c) \in \mathbf{VP}_G^{\mathcal{B}}(A)$ is a *profile for* ξ, and we refer to m as the ***priority*** and c the ***interface*** of the profile respectively. We write $\mathbf{VP}_G^{\mathcal{B}}(\xi : A)$ for the set of profiles for ξ.
We let θ and τ range over variable profiles, and define a binary relation $\lessdot$ over profiles of variables that occur in $\lambda(G)$: we define $\tau \lessdot \theta$ just in case τ is an element of the interface of θ.

(ii) An *active profile* is just a profile $\theta = (\varphi, q, m, c)$ (say) equipped with a 1-bit memory b. The boolean value b is intended to be the answer to the question: "Is the highest priority seen thus far (since the creation of the active profile) equal to m?" Formally an ***active profile*** is a pair θ^b where θ is a profile and $b \in \{\, \mathsf{t}, \mathsf{f} \,\}$ is a Boolean value. 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 $l \leq p$, we define an *update* function of the 1-bit memory as follows:

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

where $[l = m]$ denotes the Boolean value of the equality test "$l = m$". For any profile θ, we define $\theta \uparrow l$ (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 $\{\, \theta^b \uparrow l : \theta^b \in \rho \,\}$.

3.3 Traversal-simulating alternating parity automaton $\mathcal{C}$

Definition 3.2. Given a recursion scheme G and a *property* APT $\mathcal{B} = \langle\, \Sigma, Q, \delta, q_0, \Omega \,\rangle$ over the Σ-labelled value tree $[\![\, G \,]\!]$, we can now define the auxiliary ***traversal-simulating alternating parity automaton*** over the Λ_G-labelled computation tree $\lambda(G)$:

$$\mathcal{C} \;=\; \langle\, \Lambda_G,\ Q_{\mathcal{C}},\ \delta_{\mathcal{C}} : Q_{\mathcal{C}} \times \Lambda_G \longrightarrow \mathsf{B}^+([ar(\Lambda_G)]_0 \times Q_{\mathcal{C}}),\ q_0\, \varnothing,\ \Omega_{\mathcal{C}} : Q_{\mathcal{C}} \longrightarrow \mathbb{N} \,\rangle$$

where:

—The input alphabet is Λ_G.

—A $\mathcal{C}$-*state* (i.e. an element of $Q_{\mathcal{C}}$) is either a pair $q\,\rho$ or a triple $q\,\rho\,\theta$, where $q \in Q$ is the $\mathcal{B}$-state that is 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 is defined by cases:

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

Given a $\mathcal{C}$-state $d = q\,\rho$ or $q\,\rho\,\theta$, we shall refer to $\Omega(q)$ as its ***$\mathcal{B}$-priority***, and $\Omega_{\mathcal{C}}(d)$ as its ***$\mathcal{C}$-priority***.

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

$$\bigcup_{i=1}^{k} \rho_i \ = \ \rho. \tag{3}$$

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, since true is satisfied by every subset of $[0] \times Q = \varnothing$, it follows that equation (3) can only be satisfied provided $\rho = \varnothing$.

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

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

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

for some $r \geq 0$ (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 $1 \leq i_j \leq n$.

Guess ρ' to be either ρ or $\rho \setminus \{\, (\varphi, q, m, c)^{\mathsf{t}} \,\}$.

For each $1 \leq j \leq r$, guess distinct environments $\rho_{j1}, \cdots, \rho_{jr_j}$ with $r_j \geq 1$, such that

$$\bigcup_{j=1}^{r} \bigcup_{k=1}^{r_j} \rho_{jk} \ = \ \rho'. \tag{4}$$

For each $1 \leq j \leq r$ and each $1 \leq k \leq r_j$, 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.

Note. If φ is order 0, the interface c in θ is necessarily empty (i.e. $r = 0$). Thus, for equation (4) to hold, we must have $\rho' = \varnothing$; it follows that $\rho = \{\, (\varphi, q, m, \varnothing) \,\}$.

Case 3: l is @ of type $((A_1, \cdots, A_n, o), A_1, \cdots, A_n, o)$ where $n \geq 1$, 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

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

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

$$\bigcup_{j=1}^{r} \bigcup_{k=1}^{r_j} \rho_{jk} \;=\; \rho. \tag{5}$$

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 $e = q\,\rho$.

We tabulate the shape of the $\mathcal{C}$-state of an automaton at a node according to its label:

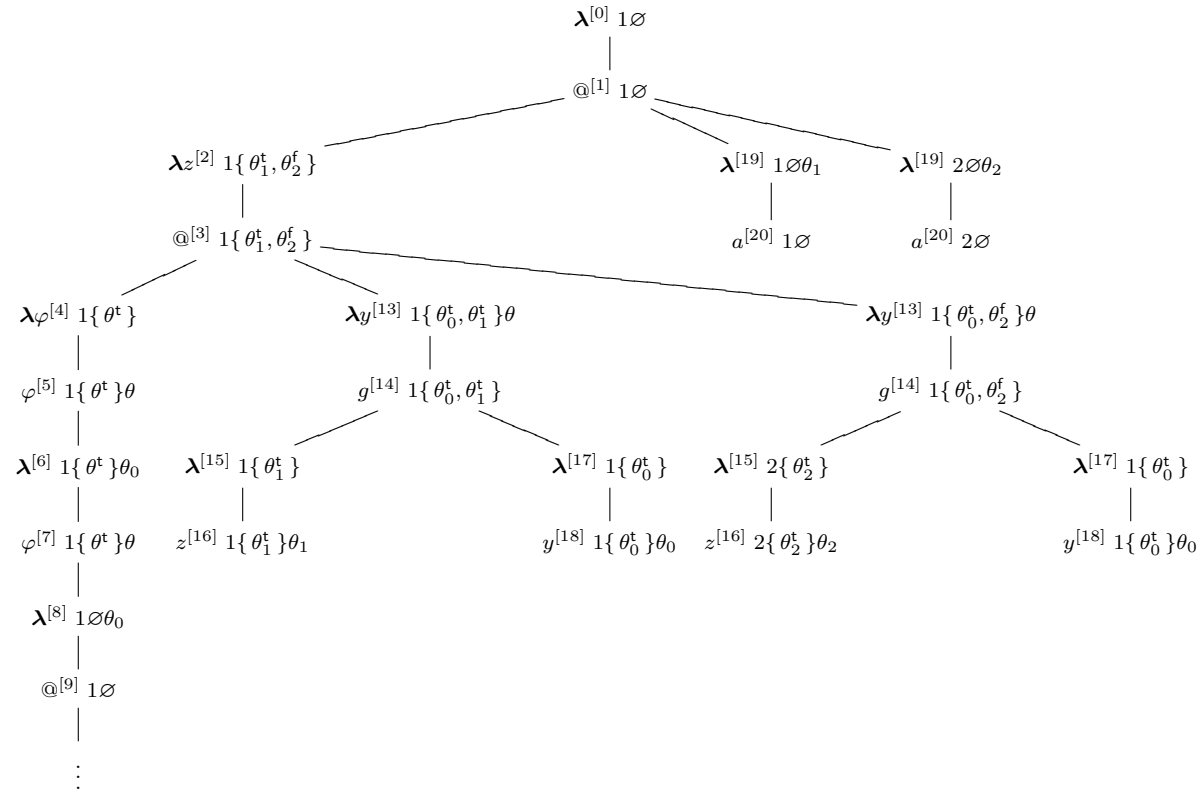
Node labels		*$\mathcal{C}$-states*
Non-Lambda	Apply, @	$q \quad \rho$
	Variable, φ	$q \quad \rho \quad \theta$
	Σ-symbol, f	$q \quad \rho$
Lambda	Leftmost child of an @	$q \quad \rho$
	Nonleftmost child of an @	$q \quad \rho \quad \theta$
	Child of a Σ-symbol	$q \quad \rho$
	Child of a Variable	$q \quad \rho \quad \theta$

REMARK 3.3. Note the similarity between cases 2 (variable label) and 3 (@ label): the interface c of the profile θ for $\varphi : (A_1, \cdots, A_n, o)$ in the former, corresponds to the (guessed) profile-set c in the latter – they are "denotations" of syntactic objects of the same type $(A_1, \cdots, A_n, o)$.

EXAMPLE 3.4. Take the computation tree $\lambda(G)$ and the property APT $\mathcal{B}$ for Σ-labelled trees as defined in Example 2.12. In Figure 8 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 of the traversal-tree in Figure 6.

At the node $@^{[3]}$, two copies of the automaton are spawn in direction 1 (with respect to the computation tree $\lambda(G)$), one with $\mathcal{C}$-state $1\{\, \theta_0^{\mathsf{t}}, \theta_1^{\mathsf{t}} \,\}\,\theta$ and the other with $\mathcal{C}$-state $1\{\, \theta_0^{\mathsf{t}}, \theta_2^{\mathsf{f}} \,\}\,\theta$. This is *not* a consequence of the fact that there are two occurrences of φ (namely $\varphi^{[5]}$ and $\varphi^{[7]}$) in the extent rooted at $\boldsymbol{\lambda}\varphi^{[4]}$ – just consider, for example, another (accepting) run-tree as shown in Figure 9. This run-tree simulates a traversal-tree in which, at all times, it is the second disjunct of $\delta(1, g)$ (where δ is the transition function of the property APT $\mathcal{B}$ in Example 2.12) that is chosen.

A run-tree $\mathbf{r}$ of the APT $\mathcal{C}$ is a certain set of $Q_{\mathcal{C}}$-annotated paths of the computation tree $\lambda(G)$. When technically convenient to do so, we shall present $\mathbf{r}$ in terms of its trace language (see Remark 1.1), which is a (nonnull-prefix closed) set of sequences of pairs of the form $\alpha_{\langle e \rangle}$ where $\alpha \in \mathsf{Dom}(\lambda(G))$ and $e \in Q_{\mathcal{C}}$. Take a node $\omega \in \mathsf{Dom}(\mathbf{r})$ with $\mathbf{r}(\omega) = \alpha_{\langle d \rangle}$. We shall call ω a *variable node* (respectively lambda, @- and prime node) just if α is a variable node (respectively lambda, @- and prime node). We say that ω is ***in***



To save writing, we name the following variable profiles:

$$\theta = (\varphi, 1, 1, \{\,\theta_0\,\}) \qquad \theta_0 = (y, 1, 1, \varnothing) \qquad \theta_1 = (z, 1, 1, \varnothing) \qquad \theta_2 = (z, 2, 2, \varnothing)$$

Fig. 8. A run-tree of the traversal-simulating APT associated with the property APT in Example 2.12.

the scope of a lambda node $\gamma \in \mathsf{Dom}(\mathbf{r})$ just if $\gamma \leq \omega$, such that every node ω' such that $\gamma < \omega' \leq \omega$ is non-prime. The following Proposition characterises the environment part of the $\mathcal{C}$-states that label an accepting run-tree.

PROPOSITION 9 (ENVIRONMENT). *Take a node ω of an accepting run-tree $\mathbf{r}$, with $\mathbf{r}(\omega) = \alpha_{\langle d\rangle}$ and $d = q\,\rho$ or $q\,\rho\,\theta$. For any variable profile $\tau = (\varphi, q_2, m, c)$, we have $\tau^b \in \rho$ if and only if*

(1) ω is in the scope of some (necessarily unique) lambda node $\gamma \in \mathsf{Dom}(\mathbf{r})$ that is labelled $\boldsymbol{\lambda}\overline{\varphi}$ such that φ is in the list $\overline{\varphi}$, with $\mathcal{C}$-state $q_1\,\rho_1\,\theta_1$ and $\tau^{[\Omega(q_1)=m]} \in \rho_1$; and

(2) in the subtree of $\mathbf{r}$ rooted at ω, there is an occurrence of a φ-labelled node which is bound by γ with $\mathcal{C}$-state $q_2\,\rho_2\,\tau$ (where necessarily $\tau^{\mathsf{t}} \in \rho_2$);

further $b = [l = m]$ where l is the maximal $\mathcal{C}$-priority in the path from γ to ω (inclusive).

PROOF. By a straightforward induction on the length of paths in an accepting run-tree. □

Finally we collect a number of useful observations:

PROPOSITION 10. *Let $\mathbf{r}$ be an accepting run-tree of a traversal simulating APT $\mathcal{C}$. Let $\omega \in \mathsf{Dom}(\mathbf{r})$ with $\mathbf{r}(\omega) = \alpha_{\langle d\rangle}$ where $d = q\,\rho$ or $q\,\rho\,\theta$.*

(i) Suppose $\lambda(G)(\alpha) = \boldsymbol{\lambda}\overline{\xi}$ (so that $d = q\,\rho\,\theta$). For any profile τ of a variable ξ in the list $\overline{\xi}$, we have $\tau \lessdot \theta$ iff $\tau \uparrow \Omega(q) \in \rho$.

(ii) For any profile $\tau = (\xi, q_2, m, c)$ such that $\tau^b \in \rho$, we have $\Omega(q) \leq m$; it follows that the $\mathcal{B}$-priority at d is less than or equal to its $\mathcal{C}$-priority $\Omega_{\mathcal{C}}(d)$.

PROOF. Immediate consequences of Definition 3.2. □

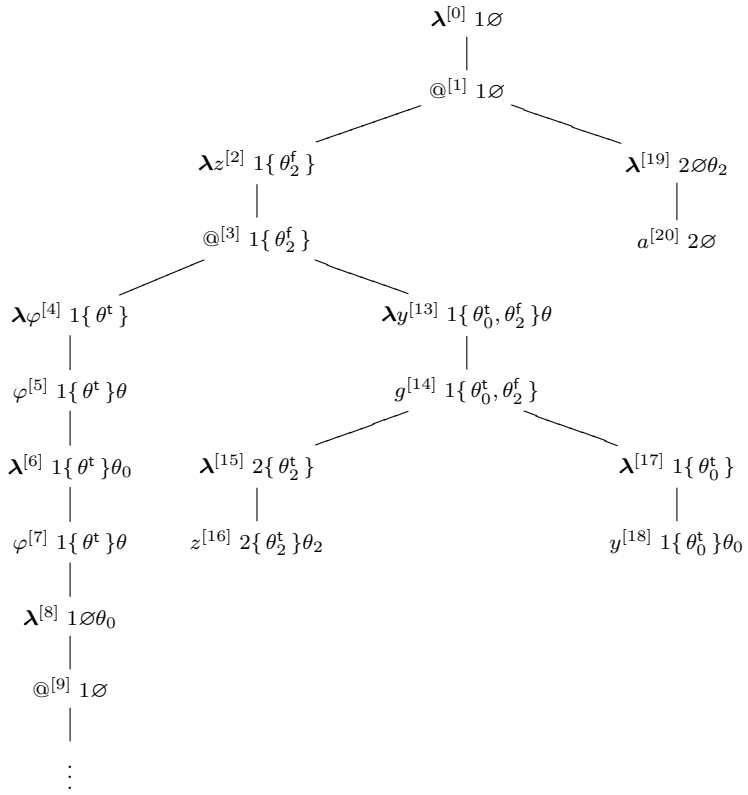


Fig. 9. Another run-tree of the traversal-simulating APT associated with the property APT in Example 2.12.

4. FROM RUN-TREES OF $\mathcal{C}$ TO TRAVERSAL-TREES OF $\mathcal{B}$

For the rest of the paper, we shall fix a recursion scheme G, and an associated computation tree $\lambda(G)$, which is a Λ_G-labelled tree. We shall also fix a property APT $\mathcal{B}$ over Σ-labelled trees, and write $\mathcal{C}$ as the associated traversal-simulating APT over Λ_G-labelled trees.

Take an accepting run-tree $\mathbf{r}$ of $\mathcal{C}$ over the computation tree $\lambda(G)$. We shall construct an annotated traversal-tree – call it $\mathbf{t}$ – in which each node is annotated with a $\mathcal{C}$-state i.e. $\mathbf{t}$ is a $(\mathsf{Dom}(\lambda(G))\times Q_{\mathcal{C}})$-labelled tree. We shall assume that $\mathbf{r}$ and $\mathbf{t}$ are presented in terms of their respective trace languages i.e. each is a (certain nonnull-prefix closed) set of sequences of pairs of the form $\alpha_{\langle e\rangle}$ where $\alpha \in \mathsf{Dom}(\lambda(G))$ and $e \in Q_{\mathcal{C}}$. The difference between the two is that the element-wise first-projection of a trace of $\mathbf{r}$ gives a path in $\lambda(G)$, whereas that of a trace of $\mathbf{t}$ gives a traversal over $\lambda(G)$.

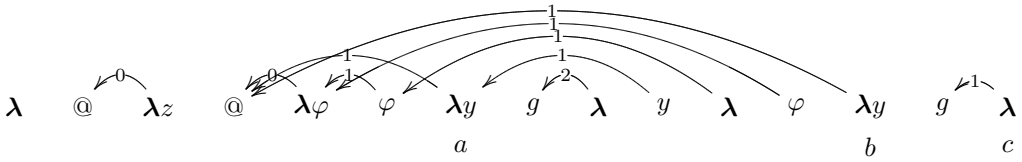
First we introduce a partial order $\preccurlyeq$ on prefixes of a traversal, called ***view order***.

(i) We define (informally) a binary relation $\preccurlyeq$ over prefixes of a traversal w 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, $\ulcorner u \urcorner \leq \ulcorner v \urcorner$.

(ii) Suppose $u \preccurlyeq v$ (which we assume has the force that u is a prefix of v). We shall refer to the segment s such that $\ulcorner u \urcorner\, s = \ulcorner v \urcorner$ as the *view difference* of u and v.

Example 4.1. Consider the following justified sequence $\mathbf{t}$ (over the computation tree in Figure 2):

wherein nodes are referred to by their respective labels



We have $\ulcorner \mathbf{t}_{\leqslant a} \urcorner = \ulcorner \mathbf{t}_{\leqslant b} \urcorner \leq \ulcorner \mathbf{t}_{\leqslant c} \urcorner$; but $\mathbf{t}_{\leqslant a} \not\preccurlyeq \mathbf{t}_{\leqslant b}$, $\mathbf{t}_{\leqslant a} \not\preccurlyeq \mathbf{t}_{\leqslant c}$ and $\mathbf{t}_{\leqslant b} \preccurlyeq \mathbf{t}_{\leqslant c}$.

We state some properties of the binary relation $\preccurlyeq$.

LEMMA 11 (VIEW ORDER). *Let u, u_i, v range over prefixes of a traversal w.*

(i) The binary relation $\preccurlyeq$ is a partial order.

(ii) If $u_1 \preccurlyeq v$ and $u_2 \preccurlyeq v$ then either $u_1 \preccurlyeq u_2$ or $u_2 \preccurlyeq u_1$.

(iii) If $\mathbf{l}(v)$ is hereditarily justified by $\mathbf{l}(u)$ in w, then (by Visibility) $u \preccurlyeq v$; the converse is of course false.

PROOF. A straightforward exercise. □

4.1 The traversal-tree $\mathbf{t}$: construction and some properties

Given an accepting run-tree $\mathbf{r}$ of $\mathcal{C}$, we shall construct the set $\mathbf{t} = \mathbf{t}[\mathbf{r}] \subseteq (\mathsf{Dom}(\lambda(G)) \times Q_{\mathcal{C}})^*$ by induction on the length of sequences in $\mathbf{t}$. We shall define $\mathbf{t}$ and simultaneously construct the proof of Proposition 12, which collects a number of properties about sequences in $\mathbf{t}$. Note that we take $\mathbf{r}$ to be the corresponding trace language i.e. $\mathbf{r} \subseteq (\mathsf{Dom}(\lambda(G)) \times Q_{\mathcal{C}})^*$.

PROPOSITION 12. *Let w be a sequence in $\mathbf{t}$ that ends in $\alpha_{\langle d \rangle}$ with $d = q\,\rho$ or $q\,\rho\,\theta$. Then we have:*

(i) $\ulcorner w \urcorner \in \mathbf{r}$.

(ii) Take a profile $\tau = (\varphi, q', m, c)$ such that $\tau^b \in \rho$. Then w has a prefix w_τ that ends in $\gamma_{\langle d_0 \rangle}$ i.e.

$$w \quad = \quad \underbrace{\cdots \gamma_{\langle d_0 \rangle}}_{w_\tau} \cdots \alpha_{\langle d \rangle}$$

with $d_0 = q''\,\rho''$ or $q''\,\rho''\,\theta''$, satisfying the following:

(1) $w_\tau \preccurlyeq w$

(2) $\lambda(G)(\gamma) = \boldsymbol{\lambda}\overline{\varphi}$ (say) such that φ is in the list $\overline{\varphi}$, and $\tau^{[\Omega(q'') = m]} \in \rho''$

(3) Every node that occurs in the view difference of w_τ and w is non-prime.

(4) Let l be the highest priority simulated by some annotated node in w that occurs at or after $\gamma_{\langle d_0 \rangle}$. Then $l \leq m$, and $b = [l = m]$.

(iii) If the penultimate element of w is labelled by a variable and annotated with $\mathcal{C}$-state $q_1\,\rho_1\,\theta_1$, then α is labelled by a lambda and d has the form $q_1\,\rho_1'\,\theta_1$ for some ρ_1'.

PROOF OF PROPOSITION 12 AND DEFINITION OF $\mathbf{t} = \mathbf{t}[\mathbf{r}]$. Since $\varepsilon_{\langle q_0\, \varnothing \rangle} \in \mathbf{r}$, the singleton sequence $\epsilon_{\langle q_0\, \varnothing \rangle}$ is in $\mathbf{t}$. Now suppose all sequences u in $\mathbf{t}$ of length up to l have been constructed. We analyse the cases of the label (i.e. $\lambda(G)(\alpha)$) of the last element $\alpha_{\langle d \rangle}$ of u, and construct all pairs $\beta_{\langle e \rangle}$ such that $\underbrace{u\,\beta_{\langle e \rangle}}_{w} \in \mathbf{t}$; at the same time, we prove by induction that statements (i), (ii) and (iii) of the Proposition are valid.

Cases of the label of α

Case 1: α is labelled by an @ with $\mathcal{C}$-state $d = q\,\rho$.

The node β is $\alpha\,0$ and $e = q\,\rho'$, for some ρ' such that $\ulcorner u \urcorner\, \beta_{\langle q\,\rho' \rangle} \in \mathbf{r}$. Since $\ulcorner u \urcorner \in \mathbf{r}$ (by the induction hypothesis of (i)) and the run-tree $\mathbf{r}$ is assumed to be accepting, it follows from Definition 3.2 that there is such a ρ', and β is labelled by a lambda, say, $\boldsymbol{\lambda}\overline{\xi}$. Since $\ulcorner u\,\beta_{\langle q\,\rho' \rangle} \urcorner = \ulcorner u \urcorner\, \beta_{\langle q\,\rho' \rangle}$, we have $\ulcorner w \urcorner \in \mathbf{r}$.

To show (ii), take an active profile $\tau^b \in \rho'$ where $\tau = (\xi, q', m, c)$. Since $\mathbf{r}$ is an accepting run-tree, ξ is in the list $\overline{\xi}$, $w_\tau = w$, $l = \Omega(q)$ and $b = [m = \Omega(q)]$ as required.

Case 2: α is labelled by a Σ-symbol f, with $\mathcal{C}$-state $d = q\,\rho$.

If the arity of f is 0, because $\mathbf{r}$ is accepting, we have u is terminal. Suppose f has arity greater than 0. Since $\ulcorner u \urcorner \in \mathbf{r}$ by the induction hypothesis (i), and $\mathbf{r}$ is an accepting run-tree, there is a set $S = \{ (i_j, q_{l_j}) : 1 \leq j \leq r \}$ that satisfies $\delta(q, f)$; further for each j, we have $\ulcorner u \urcorner (\alpha\, i_j)_{\langle q_{l_j}\, \sigma_j \rangle} \in \mathbf{r}$ such that each $\sigma_j = \rho_j \upharpoonright \Omega(q_{l_j})$ and

$$\bigcup_{j=1}^{r} \rho_j \;=\; \rho \tag{6}$$

We define $u\,(\alpha\, i_j)_{\langle q_{l_j}\, \sigma_j \rangle} \in \mathbf{t}$, for each $1 \leq j \leq r$. Note that (i) follows trivially from $\ulcorner u\, \beta_{\langle q_{l_j}\, \sigma_j \rangle} \urcorner = \ulcorner u \urcorner \beta_{\langle q_{l_j}\, \sigma_j \rangle}$.

As for (ii), take any j and set $w = u\,(\alpha\, i_j)_{\langle q_{l_j}\, \sigma_j \rangle}$, and take an active profile $\tau^b = (\xi, q', m, c)^b \in \sigma_j$. By Proposition 10(ii), we have $\Omega(q_{l_j}) \leq m$. From (6), we have $\tau^{b_1} = (\xi, q', m, c)^{b_1} \in \rho$ such that $\tau^b = \tau^{b_1} \upharpoonright \Omega(q_{l_j})$. By the induction hypothesis of (ii) applied to u, there is a prefix u_τ of u such that $u_\tau \preccurlyeq u$ (and hence, $u_\tau \preccurlyeq w$); also, let l be the highest priority seen in u since $\mathbf{l}(u_\tau)$, we have $b_1 = [l = m]$. Take $w_\tau = u_\tau$. Since $\tau^b = \tau^{b_1} \upharpoonright \Omega(q_{l_j})$ we have

$$\begin{aligned} b \;&=\; b_1 \vee [\Omega(q_{l_j}) = m] \\ &=\; [l = m] \vee [\Omega(q_{l_j}) = m] \\ &=\; [l' = m] \end{aligned}$$

where l' is the highest priority seen in w since $\mathbf{l}(w_\tau)$, as desired.

Case 3: α is labelled by a variable $\xi : (A_1, \cdots, A_n, o)$ where $n \geq 0$, with $d = q\,\rho\,\theta$.

Suppose we have

$$\begin{array}{rcl} u \;=\; u_0 \;\; \gamma_{1\langle d_1 \rangle} \;\; \gamma_{2\langle d_2 \rangle} \;\; \cdots \;\; \alpha_{\langle q\,\rho\,\theta \rangle} \\ \qquad\qquad\quad \boldsymbol{\lambda}\overline{\xi} \qquad\quad \xi \end{array}$$

where γ_1 is (necessarily) labelled by a non-lambda. By the induction hypothesis we have $\ulcorner u \urcorner \in \mathbf{r}$. The $\mathcal{C}$-state annotation of the new node β, labelled by $\boldsymbol{\lambda}\overline{\varphi}$ and i-justified by γ_1, depends on γ_1's label. There are two cases:

—*Case 3.1: Suppose γ_1 is labelled by an @ and $d_1 = q_1 \rho_1$.*

It follows from Definition 3.2 that $d_2 = q_1\, \rho_1'$, for some ρ_1'. We define w to be the following sequence:

$$\begin{array}{rcl} w \;=\; u_0 \;\; \gamma_{1\langle q_1\, \rho_1 \rangle} \;\; \gamma_{2\langle q_1\, \rho_1' \rangle} \;\; \cdots \;\; \alpha_{\langle q\,\rho\,\theta \rangle} \;\; \beta_{\langle q\,\rho'\,\theta \rangle} \\ \qquad\quad @ \qquad\quad \boldsymbol{\lambda}\overline{\xi} \qquad\quad \cdots \qquad\quad \xi \qquad\quad \boldsymbol{\lambda}\overline{\varphi} \end{array}$$

where $\beta = \gamma_1\, i$, for some ρ' yet to be determined; note that (iii) is satisfied at once. Since $\mathbf{r}$ is an accepting run-tree, we have $\theta^{\mathsf{t}} \in \rho$ with $\theta = (\xi, q, m_1, c_1)$ say. By the induction hypothesis of (ii), we have $\theta^{[m_1 = \Omega(q_1)]} \in \rho_1'$, and m_1 is the highest priority seen in u since $\gamma_{2\langle q_1\, \rho_1' \rangle}$.

As for ρ', it is set to be that ρ' such that

$$\begin{array}{rcl} \ulcorner u_0 \urcorner \;\; \gamma_{1\langle q_1 \rho_1 \rangle} \;\; \beta_{\langle q\,\rho'\,\theta \rangle} \;\in\; \mathbf{r} \\ \qquad @ \qquad\quad \boldsymbol{\lambda}\overline{\varphi} \end{array} \tag{7}$$

with β set to $\gamma_1\, i$. But we need to show that there is a P-view of the shape (7) in $\mathbf{r}$. By the induction hypothesis of (i), we have $\ulcorner u \urcorner \in \mathbf{r}$; since $\ulcorner u_0\, \gamma_{1\langle d_1 \rangle}\, \gamma_{2\langle d_2 \rangle} \urcorner \leq \ulcorner u \urcorner$ and $\mathbf{r}$ is nonnull-prefix closed, we have $\ulcorner u_0 \urcorner \gamma_{1\langle d_1 \rangle}\, \gamma_{2\langle d_2 \rangle} \in \mathbf{r}$. Since $\mathbf{r}$ is an accepting run-tree, and $\theta^{[m_1 = \Omega(q_1)]} \in \rho_1'$, we have a P-view in $\mathbf{r}$ of shape (7). To show (i), note that $\ulcorner u\, \beta_{\langle q\,\rho'\,\theta \rangle} \urcorner$ is exactly the annotated P-view in (7), and so, is in $\mathbf{r}$.

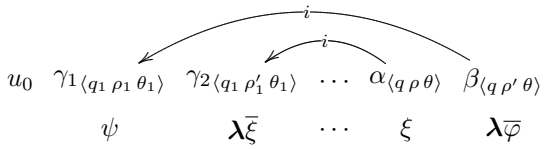
To show (ii), take an active profile $\tau^b = (\chi, q', m, c)^b \in \rho'$. There are two cases: either χ is in the list $\overline{\varphi}$, or for some b_1 the profile τ^{b_1} appears in ρ_1. Suppose the former. Then we have $w_\tau = w$ and $b = [\Omega(q) = m]$ as desired. Now, suppose the latter. Set $v = u_0\,\gamma_{1\langle q_1\,\rho_1\rangle}$. By the induction hypothesis of (ii), there is a prefix v_τ of v such that $v_\tau \preccurlyeq v$ (and so $v_\tau \preccurlyeq w$) and the environment at $\mathbf{l}(v_\tau)$ contains the active profile $(\chi, q', m, c)^{[m=\Omega(q'')]}$ where q'' is the simulated state; further we have $b_1 = [l = m]$ where l is the highest priority seen in v since $\mathbf{l}(v_\tau)$. Now take $w_\tau = v_\tau$. Because ρ' is determined by (7), we have

$$\begin{aligned} b &= b_1 \vee [m_1 = m] \\ &= [l = m] \vee [m_1 = m] \\ &= [l' = m] \end{aligned}$$

where l' is the highest priority seen in w since $v_\tau = w_\tau$, as desired.

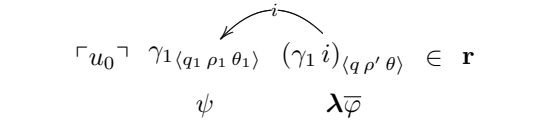
—*Case 3.2: Suppose γ_1 is labelled by a variable ψ and $d_1 = q_1\,\rho_1\,\theta_1$.*

Then w is the following sequence:



where $\beta = \gamma_1\,i$. It follows from the definition of $\mathbf{r}$ that $\theta^{\mathsf{t}} \in \rho$ where $\theta = (\xi, q, m_1, c_1)$ say. By the induction hypothesis of (ii), $\theta^{[m_1=\Omega(q_1)]} \in \rho'_1$, and m_1 is the highest priority seen in u since γ_2. Note that we have statement (iii) at once.

Also, by Proposition 10(i), $\theta \lessdot \theta_1$. Since $\ulcorner u_0 \urcorner \gamma_{1\langle q_1\,\rho_1\,\theta_1\rangle} \leq \ulcorner u \urcorner$, and $\ulcorner u \urcorner \in \mathbf{r}$ (by the induction hypothesis), and $\mathbf{r}$ is nonnull-prefix closed, we have $\ulcorner u_0 \urcorner \gamma_{1\langle q_1\,\rho_1\,\theta_1\rangle} \in \mathbf{r}$. Since $\mathbf{r}$ is accepting and the interface of θ is non-empty, we have



(8)

for some ρ'. Statement (i) then follows from $\ulcorner w \urcorner = \ulcorner u_0 \urcorner \gamma_{1\langle q_1\,\rho_1\,\theta_1\rangle}\,(\gamma_1\,i)_{\langle q\,\rho'\,\theta\rangle}$.

Finally we note that the argument for (ii) is identical to that for (ii) in case (3.1).

Case 4: α is labelled by a lambda.

Since $\ulcorner u \urcorner \in \mathbf{r}$ by the induction hypothesis, and since $\mathbf{r}$ is assumed to be an accepting run-tree, we have $\ulcorner u \urcorner (\alpha\,1)_{\langle e\rangle} \in \mathbf{r}$, for some unique $\mathcal{C}$-state e. We set the new node β to $\alpha\,1$ and annotate it with the $\mathcal{C}$-state e. Note that e simulates the same $\mathcal{B}$-state as the $\mathcal{C}$-state that annotates α. Statements (i) and (ii) are easily seen to be valid. □

Remark 4.2. In the proof of Cases 3.1 and 3.2, the choice of ρ' in (7) and (8) respectively is immaterial.

Let $\mathbf{t}^-$ be the $(\mathsf{Dom}(\lambda(G)) \times Q)$-labelled 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. I.e. we transform each label, which has the form $\alpha_{\langle q\,\rho\rangle}$ or $\alpha_{\langle q\,\rho\,\theta\rangle} \in \mathsf{Dom}(\lambda(G)) \times Q_{\mathcal{C}}$, to the label $\alpha_{\langle q\rangle} \in \mathsf{Dom}(\lambda(G)) \times Q$.

Lemma 13. *The $(\mathsf{Dom}(\lambda(G)) \times Q)$-labelled tree $\mathbf{t}^-$ is a traversal-tree of $\mathcal{B}$ over the computation tree $\lambda(G)$.*

Proof. Straightforward – we check that the rules in Definition 2.11 are followed in all four cases. □

4.2 Spine of a traversal

Our aim is to prove that the traversal tree $\mathbf{t}^{-}$ satisfies the accepting condition, assuming that the run-tree $\mathbf{r}$ is accepting. Take any infinite path w in $\mathbf{t}$. Since the P-view of every finite prefix of w is in $\mathbf{r}$ (Proposition 12), we first construct an appropriate infinite strictly-increasing sequence of prefixes of w (which we shall call a *spinal decomposition* of w)

$$u_1 < u_2 < u_3 < \cdots$$

such that the corresponding sequence of their respective P-views forms a strictly-increasing sequence under prefix ordering i.e.

$$\ulcorner u_1 \urcorner < \ulcorner u_2 \urcorner < \ulcorner u_3 \urcorner < \cdots.$$

The infinite sequence of P-views defines an infinite path in $\mathbf{r}$. Hence the path satisfies the parity condition i.e. the highest priority of the states that occur infinitely often is even. It then remains to reflect this property back to the infinite traversal w. In this subsection, we first make define the notion of spinal decomposition, then we show that every infinite traversal w has a spinal decomposition (Lemma 14); and finally we show that the parity condition can be transferred back to w (Lemma 16).

Definition 4.3. Fix an infinite justified sequence w. An infinite (strictly increasing) 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) $|\ulcorner u_1 \urcorner| < |\ulcorner u_2 \urcorner| < |\ulcorner u_3 \urcorner| < \cdots$

The infinite justified subsequence of w as given by the infinite (strictly increasing) sequence $\ulcorner u_1 \urcorner < \ulcorner u_2 \urcorner < \ulcorner u_3 \urcorner < \cdots$ is called the (associated) ***spine***. Note that neither (i) nor (ii) above is a consequence of the other.

Remark 4.4. The P-view operation $\ulcorner - \urcorner$ is a transformation on finite justified sequences. Because the definition of P-view of a justified sequence is by a case analysis of the *last node* of the sequence (and by recursion on certain prefixes), there is no obvious extension of the definition to infinite sequences. However recall that a finite P-view can be characterised as a justified sequence that is a fixpoint of the P-view operation. It follows from property (i) of Definition 4.3 that the spine (of an infinite traversal) is an infinite justified subsequence such that each finite prefix of it is a fixpoint of the P-view operation. For this reason, we argue that the spine of an infinite justified sequence may be thought of as its P-view.

Lemma 14. *Every infinite traversal over a given computation tree has a spinal decomposition.*

Proof. Take an infinite traversal w represented as an infinite justified sequence of nodes of the computation tree $\lambda(G)$. We shall construct an infinite sequence of prefixes of w of one of the following shapes:

(a) $u_0 \preccurlyeq u_1 \preccurlyeq u_2 \preccurlyeq \cdots \preccurlyeq u_n \preccurlyeq u_{n+1} \preccurlyeq \cdots$
(b) $u_0 \preccurlyeq u_1 \preccurlyeq \cdots \preccurlyeq u_i \preccurlyeq v^1_@ \preccurlyeq v^2_@ \preccurlyeq v^3_@ \preccurlyeq \cdots$, for some $i \geq 0$.

such that for every $j \geq 0$, we have $|\ulcorner u_j \urcorner| = 2j + 1$; and each $v^j_@$ ends in a node labelled by a long-apply symbol. It follows that each u_j ends in a node labelled by a lambda, and u_0 is the root node ϵ.

We begin with some terminology. Let u be a prefix of w.

—We say that u is *hereditarily finite* just if there are only finitely many prefixes v of w such that $\mathbf{l}(v)$ is hereditarily justified by $\mathbf{l}(u)$; we say that u is *hereditarily infinite* otherwise.
—We say that u is *finitely visible* just if there are only finitely many prefixes v of w such that $u \preccurlyeq v$; we say that u is *infinitely visible* otherwise.

Clearly if u ends in a node labelled by a nullary Σ-symbol or order-0 variable, then u is finitely visible.

—We say that u *satisfies property (HF)* just if every prefix s of u such that $s \preccurlyeq u$ is hereditarily finite.

If $\mathbf{l}(v)$ is hereditarily justified by $\mathbf{l}(u)$, then $u \preccurlyeq v$. Hence, if u is hereditarily infinite, it is infinitely visible; the converse, however, is not true.

First observe that $u_0 = \epsilon$ is hereditarily finite, infinitely visible, and satisfies (HF). Suppose for some $i \geq 0$ we have constructed $u_0, \cdots, u_i$, and all of them are hereditarily finite, infinitely visible, and satisfies

(HF). Set $\beta_i \in \mathsf{Dom}(\lambda(G))$ to be the (necessarily non-lambda) node following u_i in w (so that $u_i\,\beta_i \leq w$). It follows that $u_i\,\beta_i$ also satisfies the same three properties. Consider now the cases of the label of β_i:

Case 1: β_i is labelled by a Σ-symbol f of arity at least one.

Because u_i is infinitely visible, the symbol f has arity greater than 0. Set $u_{i+1} = u_i\,\beta_i\,\beta$ where β is the node following β_i in w. Observe that u_{i+1} remains hereditarily finite, infinitely visible, and satisfies (HF).

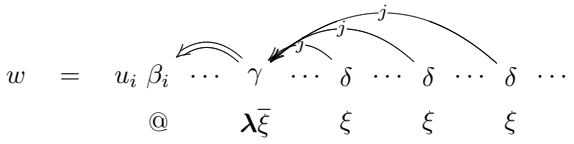
Case 2: β_i is labelled by a variable φ of order at least one.

Because u_i satisfies (HF), and β_i is justified by some node in u_i that appears in $\ulcorner u_i \urcorner$, we have β_i is hereditarily finite. Because u_i is infinitely visible and satisfies (HF), there exists some prefix v of w such that $\mathbf{l}(v)$ (necessarily a lambda node) points to β_i and v is infinitely visible. Because of (HF), there is a largest (with respect to prefix ordering) such v, and we set u_{i+1} to be it.

(In general it is possible for a prefix that ends in a variable node to be hereditarily infinite, but only if the variable has order at least 2; see e.g. Example 2.8.)

Case 3: β_i is labelled by an @.

The case of β_i being hereditarily finite is exactly the same as the preceding. Now suppose β_i is hereditarily infinite. I.e. for some $\gamma, \delta \in \mathsf{Dom}(\lambda(G))$ that are hereditarily justified by β_i, there are infinitely many occurrences of δ in w such that each points to γ, as depicted in the following:

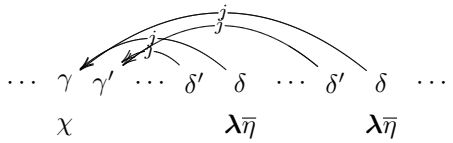


Notation. In the above diagram (and those that follow) we use:

—double-shaft arrows $\cdots \alpha_1 \cdots \alpha_2$ to mean "the node α_2 is hereditarily justified by the node α_1"

—dotted-shaft arrows $\underbrace{u\,\alpha_1 \cdots \alpha_2}_{v}$ to mean "the node α_1 appears in the P-view of v"; or equivalently "$u\,\alpha_1 \preccurlyeq v$".

Further we can pick a γ that is labelled by a lambda $\boldsymbol{\lambda}\overline{\xi}$ (say), so that δ is labelled by a variable ξ (which is in the list $\overline{\xi}$). (For if γ is labelled by a variable χ (say) and δ by a lambda $\boldsymbol{\lambda}\overline{\eta}$ (say) which j-points to γ, as shown in the following:



Since w is a traversal, the node δ', which immediately precedes δ, necessarily j-points to the node γ', which immediately follows γ, and γ' is labelled by a lambda. For the same reason, the node that immediately proceeds every subsequent occurrence of δ (as shown above) is δ' which j-points to the same occurrence of γ'. We can then take γ' and δ' to be our γ and δ respectively.)

Now, writing $v^1_@ = u_i\,\beta_i$, let v^1_0 be the least prefix of w such that

(I1) $\mathbf{l}(v^1_0)$ is hereditarily justified by $\mathbf{l}(v^0_@)$, and $\mathbf{l}(v^1_0)$ is such a γ (labelled by the lambda $\boldsymbol{\lambda}\overline{\xi}$, say); and

(I2) set $v^1_1 \leq v^1_2 \leq v^1_3 \leq \cdots$ such that for each $i \geq 1$, we have $\mathbf{l}(v^1_i) = \delta$ (labelled by the variable ξ) points to $\mathbf{l}(v^1_0)$.

Note that we have $v^1_0 \preccurlyeq v^1_i$ for each i; and $\ulcorner v^1_1 \urcorner = \ulcorner v^1_2 \urcorner = \ulcorner v^1_3 \urcorner = \cdots$ because the P-view of a traversal is a path in the computation tree (Proposition 6(ii)).

Next we show that we can in fact construct another infinite sequence of prefixes

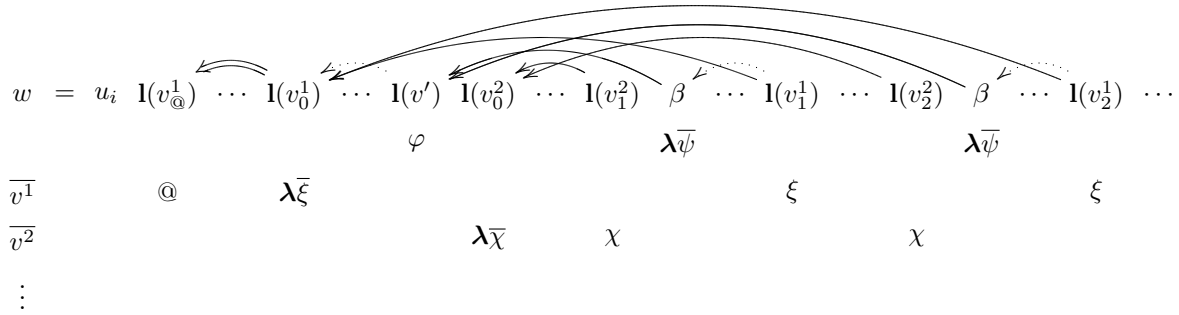
$$\overline{v^2} \quad = \quad v^2_0 \;\leq\; v^2_1 \;\leq\; v^2_2 \;\leq \cdots$$

that has the same properties – namely (I1) and (I2) – as $\overline{v^1}$. (Indeed, we shall see that we can construct an infinite sequence of such sequences $\overline{v^1}, \overline{v^2}, \overline{v^3}, \cdots$.)

Let v' be the largest prefix of w such that

(i) $v_0^1 \preccurlyeq v'$, and
(ii) for each $i \geq 1$, we have $v' \preccurlyeq v_i^1$.

Plainly $\mathbf{l}(v')$ is labelled by a variable φ (say), and it follows that there are infinitely many occurrences of the node β, labelled by a lambda $\boldsymbol{\lambda}\overline{\psi}$ (say), that points to $\mathbf{l}(v')$. Set v_0^2 to be the one-node extension of v' (i.e. $v_0^2 = v'\,\alpha'$ for some $\alpha' \in \mathsf{Dom}(\lambda(G))$). Thus, for the same reason as before, we have infinitely many prefixes of w, namely $v_1^2 \leq v_2^2 \leq v_3^2 \leq \cdots$, such that for each $i \geq 1$, we have $\mathbf{l}(v_i^2)$ (labelled by variable χ say) points to $\mathbf{l}(v_0^2)$ (labelled by $\boldsymbol{\lambda}\overline{\chi}$), as depicted in the following diagram:



We observe that $\mathbf{l}(v_1^2)$ (labelled by χ) and β (labelled by $\boldsymbol{\lambda}\overline{\psi}$) are hereditarily justified by $\mathbf{l}(v_@^2)$ which is labelled by @, for some prefix $v_@^2$; further the 2-element block, $\mathbf{l}(v_i^2)$ followed by β, occurs infinitely often in w, as i ranges over positive integers.

Clearly we have $v_@^2 \preccurlyeq v'$. Since $u_i\,\beta_i = v_@^1 \preccurlyeq v'$ as well, we have either $v_@^2 \preccurlyeq v_@^1$ or $v_@^1 \preccurlyeq v_@^2$. But the former would contradict the assumption that u_i satisfies (HF), thus we have $v_@^1 \preccurlyeq v_@^2$.

Now we note that $\mathbf{l}(v_1^1)$ is hereditarily justified by $\mathbf{l}(v_@^1)$; further

$$\cdots \mathbf{l}(v_@^1) \;\preccurlyeq\; v_@^2 \;\preccurlyeq\; \cdots \beta \;\preccurlyeq\; \cdots \mathbf{l}(v_1^1)$$

Thus it follows from Remark 2.4(4) that the node β is a descendant – in the computation tree – of some i-child of $\mathbf{l}(v_@^2)$ where $i > 0$. Hence, by Proposition 6, we have $\mathbf{l}(v_1^2)$ (labelled by χ) is a descendant – in the computation tree – of the 0-child of $\mathbf{l}(v_@^2)$. We can repeat the same argument as before and construct $v_1^3 \leq v_2^3 \leq \cdots$, and we must therefore have $v_@^2 \preccurlyeq v_@^3$; since $v_@^3 \preccurlyeq v_@^2$ would contradict Remark 2.4(4). By repeating the same argument as before, we obtain the following spinal decomposition of w:

$$u_1 \;\preccurlyeq\; \cdots \;\preccurlyeq\; u_i \;\preccurlyeq\; v_@^1 \;\preccurlyeq\; v_@^2 \;\preccurlyeq\; v_@^3 \;\preccurlyeq\; \cdots$$

and we are done. □

Remark 4.5. The proof relies on the the computation tree $\lambda(G)$ satisfying the property as given by Remark 2.4(4).

Question 15. *Does w always have a unique spine?*

Lemma 16. *Let w be an infinite path in $\mathbf{t}^-$ and w^+ be the corresponding path in $\mathbf{t}$; and let p be a spine of w. The highest $\mathcal{B}$-priority that occurs infinitely often in w coincides with the highest $\mathcal{C}$-priority that occurs infinitely often in the infinite sequence p.*

Proof. Since w has a spinal decomposition, we can write w as $u_1\,u_2\,u_3\,\cdots$ such that each u_i is a segment of the shape

$$u_i \;=\; \overset{j_i}{\alpha_{\langle d\rangle}}\; v_i\; \beta_{\langle e\rangle}$$

where α is labelled by a non-lambda, and β by a lambda, and β is j_i-justified by α. It suffices to prove:

CLAIM. *The highest $\mathcal{C}$-priority that occurs in the two-element segment $\alpha_{\langle d\rangle}\ \beta_{\langle e\rangle}$, which is $\max(\Omega_{\mathcal{C}}(d), \Omega_{\mathcal{C}}(e))$, coincides with the highest $\mathcal{B}$-priority that is simulated by some annotated node in u_i.*

We analyse the cases of the non-lambda label of α:

Case 1: α's label is a Σ-symbol. The $\mathcal{C}$-state e is a pair of the shape $q\,\rho$, and v_i is the null sequence.

Case 2: α's label is an @. There are two subcases.

—*Case 2.1.* If j_i is 0, then e is a pair of the shape $q\,\rho$ and v_i is the null sequence.

—*Case 2.2.* If $j_i > 0$, then we have

$$v_i \;=\; \overset{\overset{\overset{\overset{\overset{\overset{j_i}{\curvearrowleft}}{\curvearrowright}}{\alpha'_{\langle d_0\rangle}}}{\boldsymbol{\lambda}\overline{\varphi}}}{\cdots}}{\varphi_{j_i}}\ \beta'_{\langle q\,\rho\,\theta\rangle}$$

where β' is labelled by the variable φ_{j_i}, $\theta = (\varphi_{j_i}, q, m, c)$, and α' is labelled by $\boldsymbol{\lambda}\overline{\varphi}$ (such that φ_{j_i} occurs as the j_i-th element of the list $\overline{\varphi}$); and it follows from Proposition 12 that e is the triple $q\,\rho_1\,\theta$ for some ρ_1, where m is the highest priority simulated by the annotated nodes in v_i. Since the annotated nodes $\alpha_{\langle d\rangle}$ and ${\alpha'}_{\langle d_0\rangle}$ simulate the same $\mathcal{B}$-state, it follows that m is also the highest priority simulated by the annotated nodes in u_i, as required.

Case 3: α's label is a variable η. Omitted as it is similar to Case 2.2 in the preceding. □

Thus we can conclude:

PROPOSITION 17. *Every infinite path in the traversal-tree $\mathbf{t}^-$ determines an infinite path in the accepting run-tree $\mathbf{r}$ such that the highest $\mathcal{B}$-priority that occurs infinitely often in the former path coincides with the $\mathcal{C}$-priority that occurs infinitely often in the latter path. Hence $\mathbf{t}^-$ is an accepting traversal-tree of $\mathcal{B}$ over the computation tree $\lambda(G)$.* □

5. 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 $\mathcal{B} = \langle\, \Sigma, Q, \delta, q_0, \Omega \,\rangle$ over the computation tree $\lambda(G)$. Recall that $\mathbf{t}$ is an unranked $(\mathsf{Dom}(\lambda(G)) \times Q)$-labelled tree. We shall perform a succession of annotation operations on the traversal-tree $\mathbf{t}$, transforming it eventually to a $(\mathsf{Dom}(\lambda(G)) \times Q_{\mathcal{C}})$-labelled tree $\widehat{\mathbf{t}}$. Note that $\widehat{\mathbf{t}}$ has the same underlying tree as $\mathbf{t}$ (i.e. $\mathsf{Dom}(\widehat{\mathbf{t}}) = \mathsf{Dom}(\mathbf{t})$) but its nodes are labelled by pairs of the form $\alpha_{\langle d\rangle}$ where $\alpha \in \mathsf{Dom}(\lambda(G))$ and $d \in Q_{\mathcal{C}}$. We shall show that the set of P-views of traces of $\widehat{\mathbf{t}}$ gives an accepting run-tree of the traversal-simulating APT $\mathcal{C}$.

5.1 Annotating traversal-trees with $\mathcal{C}$-states

First some terminology. Let t be a Γ-labelled tree in which children of the same node are not ordered (e.g. $\mathbf{t}$). Take any $\omega \in \mathsf{Dom}(t)$. We define ω's *history* to be the trace $t(\alpha_1)\cdots t(\alpha_n) \in \Gamma^*$ where $\epsilon = \alpha_1, \cdots, \alpha_n = \omega$ is the unique path to ω in $\mathsf{Dom}(t)$.

DEFINITION 5.1 (TRANSFORMING $\mathbf{t}$ TO $\widehat{\mathbf{t}}$). We begin with some preprocessing of the traversal-tree $\mathbf{t}$. For each node $\omega \in \mathsf{Dom}(\mathbf{t})$ that is labelled $\alpha_{\langle q\rangle}$ such that $\lambda(G)(\alpha)$ is a variable ξ (say), we transform the label to $\alpha_{\langle q\,(\xi,q,m,\varnothing)\rangle}$ where m is the highest priority seen in ω's history, starting from its justifier node (i.e. binder of ξ). Call $\mathbf{t}_0$ the traversal-tree thus annotated.

Stage A: Annotating variable profiles.

Starting from $\mathbf{t}_0$, we shall perform N successive operations (where $N+1$ is the order of the recursion scheme G), each transforming $\mathbf{t}_l$ to $\mathbf{t}_{l+1}$, yielding $\mathbf{t}_N$ in the end.

The invariant for each l is:

> For every node in $\mathbf{t}_l$ that is labelled $\alpha_{\langle d\rangle}$ such that $\lambda(G)(\alpha)$ is a variable $\xi : A$ of order up to l, we have $d = q\,\theta$ where θ is a profile for ξ.

The tree $\mathbf{t}_{l+1}$ is defined as follows: For each node $\omega \in \mathsf{Dom}(\mathbf{t}_l)$ that is labelled $\alpha_{\langle q\,(\xi,q,m,\varnothing)\rangle}$ where the variable $\xi : A = (A_1, \cdots, A_n, o)$ is of order $l+1$, replace the quadruple $(\xi, q, m, \varnothing)$ in the label by the variable profile $(\xi, q, m, c) \in \mathbf{VP}^{\mathcal{B}}_G(\xi : A)$, where c is defined to be the set consisting of profiles $\theta \in \bigcup_{j=1}^{n} \mathbf{VP}^{\mathcal{B}}_G(\varphi_j : A_j)$ such that for some segment v and some $1 \le j \le m$, we have

$$\cdots \quad \underset{\xi}{\alpha_{\langle q\,(\xi,q,m,\varnothing)\rangle}} \quad \overset{j}{\underset{\boldsymbol{\lambda}\overline{\varphi}}{\gamma_{\langle q\rangle}}} \quad v \quad \underset{\varphi_j}{\beta_{\langle q_1\,\theta\rangle}} \quad \in \quad \mathsf{Traces}(\mathbf{t}_l)$$

where $\cdots \alpha_{\langle q\,(\xi,q,m,\varnothing)\rangle}$ is the history of ω. Note that (it follows from the definition of traversal that) the order of φ_j is at most l. We perform the replacement operation for all such variable nodes in $\mathbf{t}_l$ by induction on the length of its history.

Set $\mathbf{t}_{\mathrm{vp}} = \mathbf{t}_N$ where $N+1$ is the order of G.

Stage B: Annotating environments.

For each node $\omega \in \mathsf{Dom}(\mathbf{t}_{\mathrm{vp}})$, we add an environment to its label as follows. Suppose u is ω's history in $\mathsf{Traces}(\mathbf{t}_{\mathrm{vp}})$ and $\mathbf{t}_{\mathrm{vp}}(\omega) = \alpha_{\langle q\,\theta\rangle}$ or $\alpha_{\langle q\rangle}$, depending on whether $\lambda(G)(\alpha)$ is a variable or not. We transform the label to $\alpha_{\langle q\,\rho\,\theta\rangle}$ or $\alpha_{\langle q\,\rho\rangle}$ respectively, where ρ is defined to be the set of active profiles τ^b where $\tau = (\xi, q', m, c)$ such that there is a sequence $w = u\,v\,\beta_{\langle q'\,\tau\rangle}$ in $\mathsf{Traces}(\mathbf{t}_{\mathrm{vp}})$ – note that $\lambda(G)(\beta) = \xi$ satisfying

(i) $u \preccurlyeq w$, and

(ii) the last element $\beta_{\langle q'\,\tau\rangle}$ of w is bound by (i.e. points to) a lambda node β_0 (say) that appears in $\ulcorner u \urcorner$, and

(iii) $b = [\![l = m]\!]$ where l is the highest priority seen in u since β_0.

Note that it follows from Remark 2.4(4) that there is no occurrence of any 2-node segment of the shape $\overset{0}{@}\ \underset{\boldsymbol{\lambda}\overline{\psi}}{\ }$ in the view difference of u and w. We add the environment annotation to all nodes in $\mathbf{t}_{\mathrm{vp}}$ by induction on the length of u.

Let $\mathbf{t}_{\mathrm{env}}$ be the annotated tree obtained at the end of the transformation. Note that at this stage, every non-variable node in $\mathbf{t}_{\mathrm{env}}$ is annotated by a pair $q\,\rho$, and every variable node by a triple $q\,\rho\,\theta$.

Stage C: Annotating $\mathcal{C}$-states.

For each node $\omega \in \mathsf{Dom}(\mathbf{t}_{\mathrm{env}})$ with label of the shape $\alpha_{\langle q\,\rho\,\theta\rangle}$ such that $\lambda(G)(\alpha)$ is a variable, we transform the label of the node $\omega\,1$ from (say) $\beta_{\langle q\,\rho_1\rangle}$ to $\beta_{\langle q\,\rho_1\,\theta\rangle}$ (note that $\lambda(G)(\beta)$ is necessarily a lambda). We perform the annotation of all lambda-nodes in $\mathbf{t}_{\mathrm{env}}$ by induction on the length of u.

Let $\widehat{\mathbf{t}}$ be the annotated traversal-tree obtained at the end of the three-stage transformation. We have $\widehat{\mathbf{t}}$ is a $(\mathsf{Dom}(\lambda(G)) \times Q_{\mathcal{C}})$-labelled $[m]$-tree where $m = ar(\Sigma) \times |Q|$.

Example 5.2. Take the order-2 computation tree $\lambda(G)$ over the input alphabet $\Sigma = \{\, g, a \,\}$ as defined in Example 2.12. Let $\mathbf{t}$ be the traversal-tree in the Example. We present the $\mathcal{C}$-state annotated traversal-tree (as defined in Definition 5.1), $\widehat{\mathbf{t}}$, in Figure 10.

5.2 P-views of annotated traversal-trees

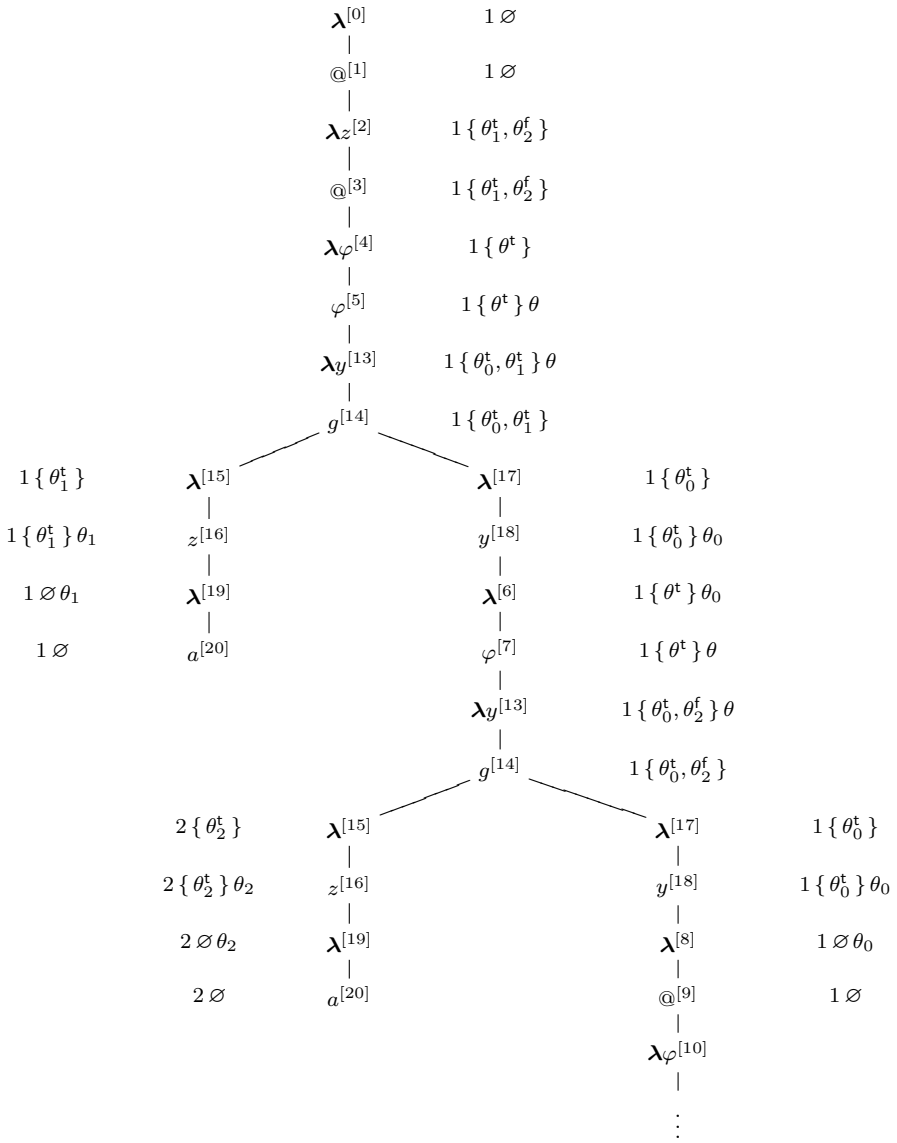
We define $\mathcal{P} = \mathcal{P}[\widehat{\mathbf{t}}]$ to be the set of P-views $\ulcorner u \urcorner$ as u ranges over $\mathsf{Traces}(\widehat{\mathbf{t}})$. Thus $\mathcal{P}$ is a (nonnull-prefix closed) set of words over the alphabet $\mathsf{Dom}(\lambda(G)) \times Q_{\mathcal{C}}$. It follows from Proposition 6(ii) that $\mathcal{P}$ is a set of $Q_{\mathcal{C}}$-annotated paths in the computation tree $\lambda(G)$.

Example 5.3. As a preparation for the following Proposition, the reader may wish to check that the set of P-views of traces of $\widehat{\mathbf{t}}$ in Example 5.2 are paths in the run-tree which is given in Figure 8.

Proposition 18. *Suppose $\mathbf{t}$ is an accepting traversal-tree of the property APT $\mathcal{B}$. The set $\mathcal{P}$ is an accepting run-tree of the traversal-simulating APT $\mathcal{C}$ over the computation tree $\lambda(G)$.*

Proof. We shall show that for each sequence $p \in \mathcal{P}$ with $\alpha_{\langle d\rangle}$ as the last element (where $\alpha \in \mathsf{Dom}(\lambda(G))$ and d is a $\mathcal{C}$-state), the set

$$\{\, (i, e) \in \mathsf{Dir}(\lambda(G)(\alpha)) \times Q_{\mathcal{C}} : p\,(\alpha\, i)_{\langle e\rangle} \in \mathcal{P} \,\}$$



We use the following shorthand for variable profiles:

$$\theta = (\varphi, 1, 1, \{\,\theta_0\,\}) \qquad \theta_0 = (y, 1, 1, \varnothing) \qquad \theta_1 = (z, 1, 1, \varnothing) \qquad \theta_2 = (z, 2, 2, \varnothing)$$

Fig. 10. A $\mathcal{C}$-state annotated traversal-tree $\widehat{\mathbf{t}}$ (transformed from the traversal-tree in Figure 6).

satisfies the formula $\delta_{\mathcal{C}}(d, \lambda(G)(\alpha))$, where $\delta_{\mathcal{C}}$ is the transition function of the traversal-simulating automaton $\mathcal{C}$. That $\varepsilon_{\langle q_0 \varnothing \rangle}$ is in $\mathcal{P}$ is obvious. Take a sequence $p \in \mathcal{P}$, and take a trace u of $\widehat{\mathbf{t}}$ such that $\ulcorner u \urcorner = p$. We analyse the cases of the label $\lambda(G)(\alpha)$ of the last node $\alpha_{\langle d \rangle}$ of p where $d = q\,\rho$ or $q\,\rho\,\theta$.

Cases of the label of α:

Case 1: α's label is a Σ-symbol f with arity $r \geq 0$, and $d = q\,\rho$.

Since $\mathbf{t}$ is accepting (by assumption), there is a minimal set $S = \{\,(i_1, q_{j_1}), \cdots, (i_k, q_{j_k})\,\}$ that satisfies $\delta(q, f)$, where each $1 \leq i_l \leq ar(f)$. If $S = \varnothing$, then p is terminal. Otherwise for each $1 \leq l \leq k$, we have $u\,(\alpha\,i_l)_{\langle q_{j_l}\,\sigma_l \rangle}$ is a trace of $\widehat{\mathbf{t}}$. We observe that for any trace w of $\widehat{\mathbf{t}}$ such that $u \leq w$, we have $u \preccurlyeq w$ iff $u\,(\alpha\,i_l)_{\langle q_{j_l}\,\sigma_l \rangle} \preccurlyeq w$ for some $1 \leq l \leq k$. Thus, a profile $\theta^b \in \rho$ iff for some $1 \leq l \leq k$, we have $\theta^b \uparrow \Omega(q_{j_l}) \in \sigma_l$. It follows that we can construct environments $\rho_1, \cdots, \rho_k$ such that $\bigcup_{l=1}^{k} \rho_l = \rho$ and $\sigma_l = \rho_l \uparrow \Omega(q_{j_l})$ for each

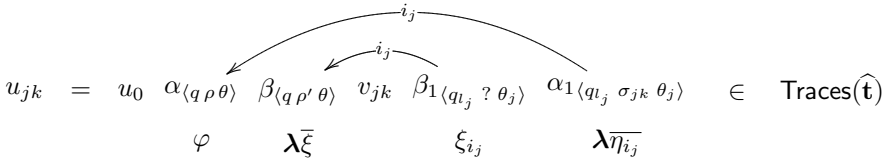
l. Now, for each l, we have $\ulcorner u\,(\alpha\, i_l)_{\langle q_{j_l}\,\sigma_l\rangle}\urcorner = p\,(\alpha\, i_l)_{\langle q_{j_l}\,\sigma_l\rangle} \in \mathcal{P}$; and the set $\{\,(i_1, q_{j_1}\,\sigma_1), \cdots, (i_k, q_{j_k}\,\sigma_k)\,\}$ satisfies $\delta_{\mathcal{C}}(q\,\rho, f)$ as required.

Case 2: α's label is a variable $\varphi : (A_1, \cdots, A_n, o)$ with $n \geq 0$, and $d = q\,\rho\,\theta$. By construction, the profile θ is of the form (φ, q, m, c), where m is, by construction, the largest priority seen in u since the node that is the binder of α. By definition of $\widehat{\mathbf{t}}$ we have $\theta^{\mathsf{t}} \in \rho$. Now suppose

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

for some $r \geq 0$.

Take a j. In the annotated tree $\widehat{\mathbf{t}}$, consider extensions of $u = u_0\,\alpha_{\langle q\,\rho\,\theta\rangle}$ ending in a lambda node that points to α as follows:



It is possible that there are infinitely many such segments v_{jk} for a given j, but there can only be finitely many distinct σ_{jk}, as k varies. W.l.og. suppose $\sigma_{j1}, \cdots, \sigma_{jr_j}$ is a pairwise distinct list, such that for any $k > r_j$, we have σ_{jk} coincides with one in the list. We analyse σ_{jk} in relation to c_j and ρ. By the Stage B annotation, we have an active profile $\tau^b \in \sigma_{jk}$ where $\tau = (\psi, q', m', c')$ iff (ψ is in the list $\overline{\eta_{i_j}}$, and so) $\tau \in c_j$ and $b = [m' {=} \Omega(q_{l_j})]$, or $\tau^{b_1} \in \rho$ (because $\boldsymbol{\lambda}\overline{\eta_{i_j}}$ points to φ) where $b_1 = [l {=} m']$ with l is the highest priority seen in u since the binder node of ψ. In the latter, we have $b = b_1 \vee [m' = m_j]$ where m_j is the highest priority seen in the segment between β and β_1, which is precisely the priority of the profile θ_j by Stage B of the annotation. Thus we can construct environments ρ_{jk} such that $\sigma_{jk} = c_j \uparrow \Omega(q_{l_j}) \cup \rho_{jk} \uparrow m_j$ and $\bigcup_{j=1}^{r} \bigcup_{k=1}^{r_j} \rho_{jk} \subseteq \rho$; inclusion in the other direction follows from the observation that for any trace w of $\widehat{\mathbf{t}}$ such that $u \leq w$, we have $u \preccurlyeq w$ implies that $u_{jk} \preccurlyeq w$ for some k (possibly greater than r_j). Hence we have $\bigcup_{j=1}^{r} \bigcup_{k=1}^{r_j} \rho_{jk} \;=\; \rho$.

By taking the respective P-views of the sequences u_{jk}, we note that $p\,(\alpha\, x)_{\langle d_x\rangle} \in \mathcal{P}$, for each pair (x, d_x) taken from the following set

$$\{\,(i_j,\; q_{l_j}\,\sigma_{jk}\,\theta_j) \;:\; 1 \leq j \leq r, 1 \leq k \leq r_j\,\}$$

which, it is easy to see, satisfies $\delta_{\mathcal{C}}(q\,\rho\,\theta,\; \varphi)$ as required.

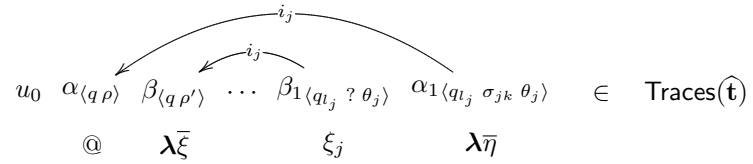
Case 3: α's label is @, and $d = q\,\rho$.

We consider traces v (of $\widehat{\mathbf{t}}$) that extend u and end in a lambda node which points to α. There are two types of such extensions. First u has a (unique) one-node extension, $u\,\boldsymbol{\lambda}\overline{\xi}_{\langle q\,\rho'\rangle}$, for some ρ'. Because of Remark 2.4, all profiles in ρ' are of variables in $\overline{\xi}$ (to be bound by $\boldsymbol{\lambda}\overline{\xi}$); say we have

$$\rho' \;=\; \{\, \theta_j^{[m_j = \Omega(q)]} : 1 \leq j \leq r \,\}$$

where $\theta_j = (\xi_{i_j}, q_{l_j}, m_j, c_j)$.

Next we consider the second type of lambda nodes that point to α. Fix a j; writing $u = u_0\,\alpha_{\langle q\,\rho\rangle}$, consider traversals of the following shape



where $1 \leq k \leq r_j$ (note that there are only finitely many such distinct σ_{jk}). Since the node $\alpha 1_{\langle q_{l_j}\,\sigma_{jk}\,\theta_j\rangle}$ points to $\alpha_{\langle q\,\rho\rangle}$, the profiles in σ_{jk} are either of variables from $\overline{\eta}$, or of variables that have profiles also in ρ. Thus, by the same reasoning as Case 2, we can construct environments ρ_{jk} such that

$$\sigma_{jk} \;=\; c_j \uparrow \Omega(q_{l_j}) \;\cup\; \rho_{jk} \uparrow m_j$$

where m_j is the third component of θ_j and $\bigcup_{k=1}^{r_j} \rho_{jk} = \rho$. By taking the respective P-views of the traversals of both kinds, we note that $p\,(\alpha\, j)_{\langle d_j \rangle} \in \mathcal{P}$, for each pair (j, d_j) taken from the following set

$$\{\,(0, q\,\rho')\,\} \cup \{\,(i_j, q_{l_j}\,\sigma_{jk}\,\theta_j) \;:\; 1 \le j \le r, 1 \le k \le r_j\,\}$$

which, it is easy to see, satisfies $\delta_{\mathcal{C}}(q\,\rho, @)$ as required.

Case 4: α's label is a lambda, and $d = q\,\rho$ or $q\,\rho\,\theta$.

By definition of $\widehat{\mathbf{t}}$, the trace u has a unique one-node extension to $u\ (\alpha\, 1)_{\langle e \rangle}$ where $e = q\,\rho\,\theta'$ for some $\theta' \in \rho$ if the label at $\alpha\, 1$ is a variable, and $e = q\,\rho$ otherwise. We have $\ulcorner u\ (\alpha\, 1)_{\langle e \rangle} \urcorner = p\ (\alpha\, 1)_{\langle e \rangle} \in \mathcal{P}$; and we check that $\{\,(1, e)\,\}$ satisfies $\delta_{\mathcal{C}}(d, \lambda(G)(\alpha))$ as required.

Finally it remains to show that every infinite trace of $\mathcal{P}$ satisfies the parity condition. Take an infinite increasing (with respect to prefix ordering) sequence of traces in $\mathcal{P}$, say, $p_0 < p_1 < p_2 < \cdots$. For each $i \ge 0$, we define P_i to be the set of infinite traces w in the annotated traversal-tree $\widehat{\mathbf{t}}$ such that w has a finite prefix v with $\ulcorner v \urcorner = p_i$. It follows from the definition that P_i is non-empty for every $i \ge 0$; further we have $P_0 \supseteq P_1 \supseteq P_2 \supseteq \cdots$. Thus $\bigcap_{i \ge 0} P_i$ is non-empty. It follows that that the spine of every infinite trace $w \in \bigcap_{i \ge 0} P_i$ is the infinite justified sequence defined by the sequence of P-views $p_0 < p_1 < p_2 < \cdots$. □

5.3 Narrow run-trees of the traversal-simulating alternating parity automaton $\mathcal{C}$

Run-trees of a traversal-simulating APT can have a rather large (though necessarily bounded) branching factor. Take a run-tree $\mathbf{r}$ and consider a node $\gamma \in \mathsf{Dom}(\mathbf{r})$ labelled $\alpha_{\langle q\,\rho_0 \rangle}$ such that $\lambda(G)(\alpha)$ is an @-symbol. Take an environment of the form $\rho = \{\,\theta_1, \cdots, \theta_r\,\} \uparrow \Omega(q)$ where each $\theta_j = (\xi_{i_j}, q_{l_j}, m_j, c_j)$. Then the set of γ's children has the following form: (we only give the $\mathcal{C}$-state part of their labels)

$$\{\,q\,\rho\,\} \quad \cup \quad \underbrace{\{\,q_{l_1}\,\tau_{11}\,\theta_1, \quad \cdots, \quad q_{l_1}\,\tau_{1r_1}\,\theta_1\,\}}_{S_{\theta_1}} \quad \cup \quad \cdots \quad \cup \quad \underbrace{\{\,q_{l_r}\,\tau_{r1}\,\theta_r, \quad \cdots, \quad q_{l_r}\,\tau_{rr_r}\,\theta_r\,\}}_{S_{\theta_r}}$$

Observe that for each j, all triples in S_{θ_j} have the same $\mathcal{B}$-state q_{l_j} and profile θ_j.

In this subsection we prove a certain *succinctness property* 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 in the sense that each S_{θ_j} above is a singleton set (similarly for children of nodes γ such that $\lambda(G)(\alpha)$ is a variable of order at least 1). The result will prove useful when analysing the complexity of the modal mu-calculus model checking problem in the sequel.

Definition 5.4. A ***narrow run-tree*** of a traversal-simulating APT $\mathcal{C}$ is a run-tree with a transition map whose definition is obtained from Definition 3.2 except that in (4) of Case 2, for each $1 \le j \le r$, 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 (5) 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 3.2.)

The last result of this section is the following:

Proposition 19 (Succinctness). *If the traversal-simulating APT $\mathcal{C}$ has an accepting run-tree then it has an accepting run-tree $\mathbf{r}_*$ that is narrow. The branching factor of a narrow run-tree is bounded above by the number of distinct variable profiles.*

Narrowing transform. We devote the rest of this section to a description of the narrowing transform and a proof of the Proposition.

Let $\mathbf{r}$ be an accepting run-tree of $\mathcal{C}$. We shall construct a *narrow* accepting run-tree, written $\mathbf{r}^*$. Recall that $\mathbf{r}$ is an unranked $(\mathsf{Dom}(\lambda(G)) \times Q_{\mathcal{C}})$-labelled tree. We define a binary relation $\curvearrowright$ between nodes of $\mathsf{Dom}(\mathbf{r})$ as follows: we say that $\gamma_1 \curvearrowright \gamma_2$ (read "γ_1 calls γ_2") just if

(i) $\mathbf{r}(\gamma_i) = \alpha_{i\,\langle q\,\rho_i\,\theta \rangle}$ i.e. γ_1 and γ_2 have the same $\mathcal{B}$-state q and profile θ in their labels.

(ii) $\lambda(G)(\alpha_1)$ is a variable φ (say, and so, $\theta = (\varphi, q, m, c)$ for some m and c) and $\lambda(G)(\alpha_2)$ is a lambda symbol $\boldsymbol{\lambda}\overline{\xi}$ (say).

(iii) α_1 and α_2 are hereditarily justified by the same @-node.

It follows that φ and $\boldsymbol{\lambda}\overline{\xi}$ have the same type. In general $\curvearrowright$ is a many-to-many relation; further for any $\omega \in \mathsf{Dom}(\mathbf{r})$, its image-set $\curvearrowright(\omega) = \{\,\gamma : \omega \curvearrowright \gamma\,\}$ and preimage-set $\curvearrowright^{-1}(\omega) = \{\,\gamma : \gamma \curvearrowright \omega\,\}$ are both non-empty.

We illustrate the two cases of γ_1 calls γ_2 in the figure below:

A. Pruning $\mathbf{r}$.

(1) Take each node $\omega \in \mathsf{Dom}(\mathbf{r})$ with $\mathbf{r}(\omega) = \alpha_{\langle d \rangle}$ such that $\lambda(G)(\alpha)$ is either an @-symbol or a variable of order at least 1. For each profile θ, consider the set S_θ of ω's children (in $\mathbf{r}$) such that the $\mathcal{C}$-state part of whose labels are of the shape $q\,?\,\theta$ i.e. they have the same profile θ. If S_θ is non-empty, we remove from the tree all nodes in S_θ and their respective descendants, except a single (the choice of which is immaterial) element (and its descendants) in S_θ whose label $q\,\rho\,\theta$ has the least ρ.
(2) Next we remove all (if any) nodes γ with label of the shape $\alpha_{\langle q\,\rho\,\theta \rangle}$ such that $\lambda(G)(\alpha)$ is a lambda and the set $\curvearrowright^{-1}(\gamma) = \{\, \omega : \omega \curvearrowright \gamma \,\}$ is empty.
(3) For any node with label of the shape $\alpha_{\langle q\,\rho\,\theta \rangle}$, we replace it by $\alpha_{\langle q\,m \rangle}$ where m is the priority of θ.

The resultant tree – call it $\mathbf{r}_0$ – has the following property: Let ω be a node in $\mathsf{Dom}(\mathbf{r}_0)$ with $\mathbf{r}_0(\omega) = \alpha_{\langle q\,m \rangle}$.

(i) If $\lambda(G)(\alpha)$ is a variable, then $\curvearrowright(\omega) = \{\, \gamma \in \mathsf{Dom}(\mathbf{r}_0) : \omega \curvearrowright \gamma \,\}$ is a singleton set.
(ii) If $\lambda(G)(\alpha)$ is a lambda and α is justified by either a variable or an @-symbol, we have $\curvearrowright^{-1}(\omega) = \{\, \gamma \in \mathsf{Dom}(\mathbf{r}_0) : \gamma \curvearrowright \omega \,\}$ is non-empty.

B. Reconstituting variable profile.

Starting from $\mathbf{r}_0$, we shall perform N successive operations (where $N+1$ is the order of the recursion scheme G), each transforming $\mathbf{r}_l$ to $\mathbf{r}_{l+1}$, yielding $\mathbf{r}_N$ in the end. The invariant for each l is:

> For every node in $\mathbf{r}_l$ that is labelled $\alpha_{\langle d \rangle}$ such that $\lambda(G)(\alpha)$ is a variable $\xi : A$ of order up to l, we have $d = q\,\theta$ where θ is a profile for ξ.

First some preprocessing for $\mathbf{r}_0$: For each node $\omega \in \mathsf{Dom}(\mathbf{r}_0)$ that is labelled $\alpha_{\langle q\,m \rangle}$ such that $\lambda(G)(\alpha)$ is an order-0 variable x (say), we transform the label to $\alpha_{\langle q\,\theta \rangle}$ where $\theta = (x, q, m, \varnothing)$. The tree $\mathbf{r}_{l+1}$ is defined as follows: For each node $\omega \in \mathsf{Dom}(\mathbf{r}_l)$ that is labelled $\alpha_{\langle q\,m \rangle}$ where $\lambda(G)(\alpha)$ is a variable $\xi : A = (A_1, \cdots, A_n, o)$ of order $l+1$, and suppose $\curvearrowright(\omega) = \omega'$ has label $\beta_{\langle q\,m \rangle}$ and $\lambda(G)(\beta) = \boldsymbol{\lambda}\overline{\varphi}$ (say); we transform ω's label to $\alpha_{\langle q\,\tau \rangle}$ where $\tau = (\xi, q, m, c) \in \mathbf{VP}^{\mathcal{B}}_G(\xi : A)$, and c is defined to be the set of profiles $\theta \in \bigcup_{j=1}^{n} \mathbf{VP}^{\mathcal{B}}_G(\varphi_j : A_j)$ such that there is some node in $\mathbf{r}_l$ with label $\beta'_{\langle q'\,\theta \rangle}$ such that β' is bound by β. We also replace the label of ω' by $\beta_{\langle q\,\tau \rangle}$.

C. Reconstituting environment.

Finally for each node $\omega \in \mathsf{Dom}(\mathbf{r}_N)$ with label of the shape $\alpha_{\langle q \rangle}$ or $\alpha_{\langle q\,\theta \rangle}$, we transform it to $\alpha_{\langle q\,\rho \rangle}$ or $\alpha_{\langle q\,\rho\,\theta \rangle}$ respectively, where ρ consists of all profiles τ such that there is some descendent ω' of ω (in $\mathbf{r}_N$) with label $\beta_{\langle q\,\tau \rangle}$ whereby $\lambda(G)(\beta)$ is a variable that is bound by ω or some ancestor node of ω. We perform the replacement operation simultaneously on all nodes in $\mathbf{r}_N$.

Call the resultant tree $\mathbf{r}^*$.

We claim that $\mathbf{r}_*$ is an accepting narrow run-tree of $\mathcal{C}$. That every infinite path in $\mathbf{r}_*$ satisfies the parity condition follows from the assumption that $\mathbf{r}$ is an accepting run-tree (because each infinite sequence of

parities in $\mathbf{r}_*$ is a sequence of parities in $\mathbf{r}$ by construction). For this reason, there is no harm in regarding environments in $\mathbf{r}_*$ as a set of profiles, as opposed to active profiles. Finally, using exactly the same arguments in the proof of Proposition 18, it is straightforward to verify that $\mathbf{r}_*$ satisfies the transition rules of $\mathcal{C}$.

EXAMPLE 5.5. The run-tree in Figure 9 is obtained from the run-tree in Figure 8 by an application of the narrowing transform.

6. DECIDABILITY AND COMPLEXITY OF MODEL CHECKING

6.1 Correctness of the simulation

We can now prove that our notion of traversal-simulating APT is correct, in the following sense:

THEOREM 20 (SIMULATION). *Let G be a recursion scheme. Take any property APT $\mathcal{B}$ over Σ-labelled trees, and let $\mathcal{C}$ be the associated traversal-simulating APT over Λ_G-labelled trees. The following are equivalent:*

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

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

PROOF. See Section 4 for (i) implies (ii), and Section 5 for the other direction. □

6.2 Λ-labelled deterministic digraph and parity acceptance game

Let Λ be a ranked alphabet. A ***Λ-labelled deterministic digraph*** (or DDG, for short) is a quadruple

$$\mathcal{K} \quad = \quad \langle\, V, \quad \to \subseteq V \times V, \quad l : V \longrightarrow \Lambda, \quad v_0 \in V \,\rangle$$

wherein the underlying digraph $\langle V, \to \rangle$ is vertex-labelled by the function $l : V \longrightarrow \Lambda$, and edge-labelled by $\mathsf{Dir}(\Gamma)$ in that $\to = \bigcup_{i \in \mathsf{Dir}(\Lambda)} \to_i$, such that

(i) for each $i \in \mathsf{Dir}(\Lambda)$, we have $\to_i \subseteq V \times V$ is a partial function

(ii) for each $v \in V$, and each $i \in \mathsf{Dir}(l(v))$, we have $\to_i(v)$ is well-defined i.e. the set $\{\, v' : (v, v') \in \to_i \,\}$ is singleton.

In the following we shall assume that V is finite. We observe that that every finite (ranked and ordered) Λ-labelled tree can be presented as a DDG, and the unfolding of a Λ-labelled DDG is a (ranked and ordered) Λ-labelled tree.

Let $\mathcal{C} = \langle\, \Lambda, Q_{\mathcal{C}}, \delta_{\mathcal{C}}, q_{\mathsf{ini}}, \Omega_{\mathcal{C}} \,\rangle$ be an APT (see Section 1.3 for a definition). We can now define the *acceptance parity game* with respect to $\mathcal{C}$ over a Λ-labelled DDG $\mathcal{K}$, $\mathbf{G}(\mathcal{K}, \mathcal{C})$. First, a notation: For each $v \in V$ and $P \subseteq \mathsf{Dir}(l(v)) \times Q_{\mathcal{C}}$, we write

$$[P]_v \;=\; \{\, (u, q) : (i, q) \in P \;\wedge\; \to_i(v) = u \,\}.$$

DEFINITION 6.1. The ***acceptance parity game*** $\mathbf{G}(\mathcal{K}, \mathcal{C})$ is a digraph. There are two kinds of vertices.

—*A-Vertices* (A for Abelard) are sets of the form $[P]_v$, with $v \in V$ and $P \subseteq \mathsf{Dir}(l(v)) \times Q_{\mathcal{C}}$

—*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_{ini}). 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 $P \subseteq \mathsf{Dir}(l(v)) \times Q_{\mathcal{C}}$ such that P satisfies $\delta_{\mathcal{C}}(q, l(v))$, there is an edge from (v, q) to $[P]_v$.

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

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

A *play* is a (possibly infinite) path in $\mathbf{G}(\mathcal{K},\mathcal{C})$ of the form

$$(v_0, q_{\mathsf{ini}}) \cdot [P_0]_{v_0} \cdot (v_1, q_1) \cdot [P_1]_{v_1} \cdots$$

By convention Eloise resolves the E-vertices, and Abelard the A-vertices. The *winning conditions* are defined as follows:

—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 priority that occurs infinitely often in the following numeric sequence

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

is even.

PROPOSITION 21. *The following are equivalent:*

(i) Eloise has a (history-free) winning strategy in the acceptance parity game $\mathbf{G}(\mathcal{K},\mathcal{C})$.

(ii) The APT $\mathcal{C}$ *accepts the* Λ*-labelled DDG* $\mathcal{K}$.

(iii) The APT $\mathcal{C}$ *accepts the* Γ*-labelled tree given by the unfolding of* $\mathcal{K}$.

PROOF. Equivalence of (i) and (ii) is standard; (ii) and (iii) are equivalent because $\mathcal{K}$ is bisimilar to its unfolding. □

Λ_G-labelled deterministic digraph determined by a recursion scheme G. Given a recursion scheme G, we shall construct a Λ_G-labelled DDG that unfolds to the computation tree $\lambda(G)$. The DGG is constructed by first forming the disjoint union of the DDGs that are determined by the respective RHSs of the rewrite rules of $\overline{G}$, and then identifying all nodes labelled by the same non-terminal F (say) with the root of the component DDG corresponding to F. Formally we define the ***Λ_G-labelled DDG determined by*** G, $\mathrm{Gr}(G)$, as follows:

$$\mathrm{Gr}(G) \quad = \quad \langle\, V, \quad \to \subseteq V \times V, \quad \lambda_G : V \longrightarrow \Lambda_G, \quad v_0 \in V \,\rangle$$

by the following procedure:

(1) First we define the ranked alphabet $\Lambda_G^+ = \Lambda_G \cup \mathcal{N}_{\overline{G}}$ where $\mathcal{N}_{\overline{G}}$ is the set of non-terminals of $\overline{G}$; each symbol in $\mathcal{N}_{\overline{G}}$ is defined to have arity 0.

(2) For each $\overline{G}$-rule (say) $F \to \boldsymbol{\lambda}\varphi_1 \cdots \varphi_n.e$, we define the corresponding Λ_G^+-labelled DDG

$$\mathcal{D}_F \quad = \quad \langle\, V_F, \quad \to^F \subseteq V_F \times V_F, \quad l_F : V_F \longrightarrow \Lambda_G^+, \quad \mathsf{rt}_F \,\rangle$$

to be the appropriate representation of the Λ_G^+-labelled tree that is determined by $\boldsymbol{\lambda}\varphi_1 \cdots \varphi_n.e$, the RHS of the rule. Note that $l_F(\mathsf{rt}_F) = \boldsymbol{\lambda}\varphi_1 \cdots \varphi_n$ with reference to the rule F given above.

(3) Set the digraph $\mathcal{D}$ to be the disjoint union of the underlying digraph of $\mathcal{D}_F$, as F ranges over $\mathcal{N}_{\overline{G}}$. We then define the underlying digraph of $\mathrm{Gr}(G)$ to be $\mathcal{D}$ quotiented by an equivalence relation between $\mathcal{D}$-vertices such that for each $F \in \mathcal{N}_{\overline{G}}$, the equivalence class $[\mathsf{rt}_F]$ consists of rt_F and all nodes (from $\mathcal{D}$) that are labelled by F i.e.

$$[\mathsf{rt}_F] \quad = \quad \left(\bigcup_{H \in \mathcal{N}_{\overline{G}}} l_H^{-1}(F) \right) \cup \{\, \mathsf{rt}_F \,\}$$

and the equivalence classes for all other nodes are singleton sets. The edge-labels of $\mathrm{Gr}(G)$ are inherited from the edge-labels of the component DDGs $\mathcal{D}_F$: we define $\to_i[\mathsf{rt}_F] = [\to_i^F(\mathsf{rt}_F)]$ for each $F \in \mathcal{N}_{\overline{G}}$). We define the vertex-labels by

$$\lambda_G([v]) = \begin{cases} l_F(\mathsf{rt}_F) & \text{if } [v] = [\mathsf{rt}_F] \text{ for some } F \in \mathcal{N}_{\overline{G}} \\ l_H(v) & \text{else if } v \text{ is a vertex in } V_H \end{cases}$$

The root of $\mathrm{Gr}(G)$ is $[\mathsf{rt}_S]$, where S is the start symbol of $\overline{G}$.

It is straightforward to check that

PROPOSITION 22. *Unfolding* $\mathrm{Gr}(G)$ *gives the computation tree* $\lambda(G)$.

Indeed we have:

PROPOSITION 23. *Using the preceding notations, the following are equivalent:*

(i) The property APT $\mathcal{B}$ *has an accepting run-tree over the* Σ*-labelled (value) tree* $[\![G]\!]$.

(ii) The traversal-simulating APT $\mathcal{C}$ *has an accepting run-tree over the* Λ_G*-labelled (computation) tree* $\lambda(G)$*, which is (isomorphic to) the unfolding of* $\mathrm{Gr}(G)$.

(iii) Eloise has a (history-free) winning strategy in the acceptance parity game $\mathbf{G}(\mathrm{Gr}(G), \mathcal{C})$. □

PROOF. The equivalence of (i) and (ii) is just Theorem 20, and the equivalence of (ii) and (iii) follows from Propositions 21 and 22. □

6.3 Complexity of modal mu-calculus model checking

Knapik *et al.* [Knapik et al. 2005] have shown that the modal mu-calculus model checking problem for order-2 trees is 2-EXPTIME-complete. Our result extends theirs to all finite orders. First we note that n-EXPTIME hardness for order-n instances of the problem follows from Cachat's result [Cachat 2003] that the modal mu-calculus model checking problem for trees generated by order-n *safe* recursion scheme is n-EXPTIME complete. Thus it remains to prove that the modal mu-calculus model checking problem for trees generated by order-n recursion scheme, whether safe or not, are n-EXPTIME decidable. We shall do so (thanks to Proposition 23) by analysing the complexity of solving the associated acceptance parity game.

Complexity of solving the acceptance parity game $\boldsymbol{G}(\mathrm{Gr}(G), \mathcal{C})$. Let G be an order-n recursion scheme. Fix a property APT $\mathcal{B} = \langle\, \Sigma, Q, \delta, q_0, \Omega \,\rangle$ with p priorities, and let $\mathcal{C} = \langle\, \Lambda_G, Q_{\mathcal{C}}, \delta_{\mathcal{C}}, q_0 \varnothing, \Omega_{\mathcal{C}} \,\rangle$ be the associated traversal-simulating APT. Our aim is to show that the acceptance parity game $\mathbf{G}(\mathrm{Gr}(G), \mathcal{C})$ is solvable in n-EXPTIME. A recent advance in the complexity of solving parity games (in general) is the following result due to Jurdziński:

THEOREM 24 [JURDZIŃSKI 2000]. *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*

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

and space $O(p \cdot |V| \cdot \log |V|)$. □

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 the early result by Emerson and Lei[8] [Emerson and Lei 1986]) is all that we need for the analysis that follows.

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 $\overline{G}$. Hence $\mathbf{VP}^{\mathcal{B}}_G$, the set of profiles for variables that occur in G with respect to the property APT $\mathcal{B}$, is $\bigcup_{i=0}^{n-1} \mathbf{VP}^{\mathcal{B}}_G(i)$. It follows from the definition of variable profiles that

$$\begin{aligned} |\mathbf{VP}^{\mathcal{B}}_G(i)| &= \mathsf{exp}_i O(|G| \cdot |Q| \cdot p) \\ |\mathbf{VP}^{\mathcal{B}}_G| &= \mathsf{exp}_{n-1} O(|G| \cdot |Q| \cdot p) \end{aligned}$$

where $|G|$ is the size of the long transform $\overline{G}$, $|Q|$ is the number of elements of Q, and $\mathsf{exp}_n m$ is the tower-of-exponentials function defined by

$$\begin{cases} \mathsf{exp}_0 m &= m \\ \mathsf{exp}_{n+1} m &= 2^{\mathsf{exp}_n m} \end{cases}$$

[8] Emerson and Lei showed that the modal mu-calculus model checking problem for a formula of size m and alternation depth d on a system of size n is $O(m \cdot n^{d+1})$.

Recall that an environment is a set of active profiles. So the set of environments is $Env_G = \mathcal{P}(\mathbf{VP}_G^{\mathcal{B}} \times \{\mathsf{t}, \mathsf{f}\})$, and we have $|Env_G| = \mathsf{exp}_n O(|G| \cdot |Q| \cdot p)$. Finally, as $Q_{\mathcal{C}} = (Q \times Env_G \times \mathbf{VP}_G^{\mathcal{B}}) \cup (Q \times Env_G)$, we have

$$|Q_{\mathcal{C}}| = \mathsf{exp}_n O(|G| \cdot |Q| \cdot p).$$

Suppose the parity acceptance game $\mathbf{G}(\mathrm{Gr}(G), \mathcal{C})$ 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 the DDG $\mathrm{Gr}(G)$. Thanks to the narrowing transform (see Proposition 19), it is enough to restrict P to subsets of $\mathsf{Dir}(l(v)) \times Q_{\mathcal{C}}$ that have size at most $|\mathbf{VP}_G^{\mathcal{B}}|$. This gives a tighter upper bound on the number of A-vertices of the game $\mathbf{G}(\mathrm{Gr}(G), \mathcal{C})$: $(d \times |Q_{\mathcal{C}}|)^{|\mathbf{VP}_G^{\mathcal{B}}|} = \mathsf{exp}_n O(|G| \cdot |Q| \cdot p)$. It follows that

$$|V| = \mathsf{exp}_n O(|G| \cdot |Q| \cdot p).$$

Since $|E|$ is at most $|V|^2$, the time complexity for solving $\mathbf{G}(\mathrm{Gr}(G), \mathcal{C})$ is

$$\begin{aligned} O\left(p \cdot (|V|)^{\lfloor p/2 \rfloor + 2}\right) &= O\left(p \cdot (\mathsf{exp}_n O(|G| \cdot |Q| \cdot p))^{\lfloor p/2 \rfloor + 2}\right) \\ &= \mathsf{exp}_n O(|G| \cdot |Q| \cdot p) \end{aligned}$$

Thus we have:

THEOREM 25. *The acceptance parity game $\mathbf{G}(\mathrm{Gr}(G), \mathcal{C})$ can be solved in time $\mathsf{exp}_n O(|G| \cdot |Q| \cdot p)$.* □

Thus, and by Proposition 21, we obtain the main result of the paper (Theorem 3(i)): The modal mu-calculus model checking problem for trees generated by order-n recursion schemes is n-EXPTIME complete. Part (ii) of the Theorem – decidability of MSO model checking – follows immediately from the well-known result that modal mu-calculus and MSO logic are equi-expressive over trees.

7. FURTHER DIRECTIONS

There are many further directions.

Does safety constrain expressiveness?

This is the most pressing open problem. In a FOSSACS 2005 paper [Aehlig et al. 2005b], we have shown that there is no inherently unsafe word language at order 2. More precisely, for every word language that is generated by an order-2 unsafe recursion scheme, there is a safe (but in general non-deterministic) recursion scheme that generates the same language. However it is conjectured that the result does not hold at order 3. Further, for trees, we conjecture that there are already *inherently* unsafe trees at order 2 i.e.

CONJECTURE 26. *There is an unsafe order-2 recursion scheme whose value tree is not the value tree of any safe recursion scheme.*

The Conjecture is closely related to a word language, which we call *Urzyczyn's language* [Aehlig et al. 2005b]. The language can be generated by a *deterministic, unsafe* order-2 recursion scheme (and hence, by a *non-deterministic, safe* order-2 recursion scheme). The Conjecture is equivalent to the statement: Urzyczyn's language cannot be generated by any *deterministic*, safe order-2 recursion scheme (or equivalently any order-2 deterministic pushdown automaton).

Collapsible pushdown automata and recursion schemes

Knapik *et al.* [Knapik et al. 2002] have shown that for every $n \geq 0$, the Σ-labelled trees generated by order-n *safe* recursion scheme are exactly those generated by order-n pushdown automata. A variant class of higher-order pushdown automata called *collapsible pushdown automata* characterise Σ-labelled trees generated by recursion schemes without the safety constraint, extending the result for *panic automata* (due to Knapik *et. al* [Knapik et al. 2005]) to all finite orders. The result will be presented elsewhere.

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

Mixing semantic and verification games

We would like to develop further the pleasing mix of Semantics (games) and Verification (games) in the paper. A specific project, *pace* [Aehlig 2006], is to give a denotational semantics of the lambda calculus "relative to an APT". More generally, it would be interesting to construct a cartesian closed category, suitably parameterized by APTs, whose maps are witnessed by variable profiles.

REFERENCES

Aehlig, K. 2006. A finite semantics for simply-typed lambda terms for infinite runs of automata. preprint.

Aehlig, K., de Miranda, J. G., and Ong, C.-H. L. 2005a. The monadic second order theory of trees given by arbitrary level two recursion schemes is decidable. In *Proceedings of the 7th International Conference on Typed Lambda Calculi and Applications (TLCA'05)*. 39–54. LNCS 3461.

Aehlig, K., de Miranda, J. G., and Ong, C.-H. L. 2005b. Safety is not a restriction at level 2 for string languages. In *Proceedings of the 8th International Conference on Foundations of Software Science and Computational Structures (FOSSACS'05)*. 490–501. LNCS 3411.

Cachat, T. 2003. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In *Proceedings of ICALP 2003*. 556–569. LNCS 2719.

Caucal, D. 1996. On infinite transition graphs having a decidable monadic theory. In *Proceedings 23rd ICALP*. Springer, 194–205. LNCS Vol. 1099.

Caucal, D. 2002. On infinite terms having a decidable monadic theory. In *Proc. MFCS'02*. Lecture Notes in Computer Science, vol. 2420. 165–176.

Damm, W. 1982. The IO- and OI-hierarchy. *Theoretical Computer Science 20*, 95–207.

de Miranda, J. 2006. Structures generated by higher-order grammars and the safety constraint. Ph.D. thesis, University of Oxford. Forthcoming.

Emerson, E. A. and Jutla, C. S. 1991. Tree automata, mu-calculus and determinacy. In *Proceedings of FOCS'91*. 368–377.

Emerson, E. A. and Lei, C. 1986. Efficient model checking in fragments of propositional mu-calculus. In *First IEEE Symp. on Logic in Computer Science*. 267–278.

Hyland, J. M. E. and Ong, C.-H. L. 2000. On Full Abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model. *Information and Computation 163*, 285–408.

Janin, D. and Walukiewicz, I. 1996. On the expressive completeness of the propositional mu-calculus with respect to msol. In *Proceedings of CONCUR'96*. 263–277. LNCS 1119.

Jurdziński, M. 2000. Small progress measures for solving parity games. In *Proc. STACS*. Lecture Notes in Computer Science, vol. 1770. 290–301.

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

Knapik, T., Niwiński, D., Urzyczyn, P., and Walukiewicz, I. 2005. Unsafe grammars and panic automata. In *ICALP'05*. Lecture Notes in Computer Science, vol. 3580. Springer-Verlag, 1450–1461.

Moggi, E. 1989. Notions of computation and monads. *Information and Computation 93*, 55–92.

Muller, D. E. and Schupp, P. E. 1985. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science 37*, 51–75.

Muller, D. E. and Schupp, P. E. 1987. Alternating automata on infinite trees. *Theoretical Computer Science 54*, 267–276.

Ong, C.-H. L. 2006a. Local computation of beta-reduction by game semantics. Preprint.

Ong, C.-H. L. 2006b. On model-checking trees generated by higher-order recursion schemes. In *Proceedings of IEEE Symposium on Logic in Computer Science*. Computer Society Press.

Stirling, C. 2000. Decidability of bisimulation equivalence for pushdown processes. Submitted for publication.

Stirling, C. 2005. Higher-order matching and games. In *Proc. CSL 2005*. 119–134. LNCS 3634.

Stirling, C. 2006. Decidability of higher-order matching. In *Proc. ICALP'06*. Springer.