COMPLEXITY OF MODEL CHECKING RECURSION SCHEMES FOR FRAGMENTS OF THE MODAL MU-CALCULUS

NAOKI KOBAYASHI AND C.-H. LUKE ONG

Graduate School of Information Sciences, Tohoku University, 6-3-09 Aoba, Aramaki, Aoba-ku
Sendai, 980-8579 Japan
e-mail address: koba@ecei.tohoku.ac.jp

Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, UK
e-mail address: Luke.Ong@comlab.ox.ac.uk

Abstract. Ong has shown that the modal mu-calculus model checking problem (equivalently, the alternating parity tree automaton (APT) acceptance problem) of possibly-infinite ranked trees generated by order-n recursion schemes is n-EXPTIME complete. We consider two subclasses of APT and investigate the complexity of the respective acceptance problems. The main results are that, for APT with a single priority, the problem is still n-EXPTIME complete; whereas, for APT with a disjunctive transition function, the problem is $(n-1)$-EXPTIME complete. This study was motivated by Kobayashi's recent work showing that the resource usage verification of functional programs can be reduced to the model checking of recursion schemes. As an application, we show that the resource usage verification problem is $(n-1)$-EXPTIME complete.

1. Introduction

The model checking problem for higher-order recursion schemes has been a topic of active research in recent years (for motivation as to why the problem is interesting, see e.g. the introduction of Ong's paper [12]). This paper studies the complexity of the problem with respect to certain fragments of the modal μ-calculus. A higher-order recursion scheme (recursion scheme, for short) is a kind of (deterministic) grammar for generating a possibly-infinite ranked tree. The model checking problem for recursion schemes is to decide, given an order-n recursion scheme $\mathcal{G}$ and a specification ψ for infinite trees, whether the tree generated by $\mathcal{G}$ satisfies ψ. Ong [12] has shown that if ψ is a modal μ-calculus formula (or equivalently, an alternating parity tree automaton), then the model checking problem is n-EXPTIME complete.

Following Ong's work, Kobayashi [10] has recently applied the decidability result to the model checking of higher-order functional programs (precisely, programs of the simply-typed λ-calculus with recursion and resource creation/access primitives). He considered the *resource usage verification problem* [6]—the problem of whether programs access dynamically created resources in a valid manner (e.g. whether every opened file will eventually be closed, and thereafter never read from or written to before it is reopened). He showed that the resource usage verification problem reduces to a model checking problem for recursion

LOGICAL METHODS
IN COMPUTER SCIENCE

DOI:10.2168/LMCS-???

schemes by giving a transformation that, given a functional program, constructs a recursion scheme that generates all possible resource access sequences of the program. From Ong's result, it follows that the resource usage verification problem is in n-EXPTIME (where, roughly, n is the highest order of types in the program). This result also implies that various other verification problems, including (the precise verification of) reachability ("Does a closed program reach the fail command?") and flow analysis ("Does a sub-term e evaluate to a value generated at program point l?"), are also in n-EXPTIME, as they can be easily recast as resource usage verification problems.

It was however unknown whether n-EXPTIME is the tightest upper-bound of the resource usage verification problem. Although the model checking of recursion schemes is n-EXPTIME-hard for the full modal μ-calculus, only a certain fragment of the modal μ-calculus is used in Kobayashi's approach to the resource usage verification problem. First, specifications are restricted to safety properties, which can be described by Büchi tree automata with a trivial acceptance condition (the class called "trivial automata" by Aehlig [1]). Secondly, specifications are also restricted to linear-time properties—the branching structure of trees is ignored, and only the path languages of trees are of interest. Thus, one may reasonably hope that there is a more tractable model checking algorithm than the n-EXPTIME algorithm.

The goal of this paper is, therefore, to study the complexity of the model checking of recursion schemes for various fragments of the modal μ-calculus (or, alternating parity tree automata) and to apply the result to obtain tighter bounds of the complexity of the resource usage verification problem.

The main results of this paper are as follows:

(i) The problem of whether a given Büchi tree automaton with a trivial acceptance condition (or, equivalently, alternating parity tree automaton with a single priority 0) accepts the tree generated by an order-n recursion scheme is still n-EXPTIME-hard, both in the size of the recursion scheme and that of the automaton. This follows from the n-EXPTIME-completeness of the word acceptance problem of higher-order alternating pushdown automata[1] [4].

(ii) We introduce a new subclass of alternating parity tree automata (APT) called *disjunctive APT*, and show that its acceptance problem for trees generated by order-n recursion schemes is $(n-1)$-EXPTIME complete. From this general result, it follows that both the linear-time properties (including reachability, which is actually $(n-1)$-EXPTIME-complete) and finiteness of the tree generated by a recursion scheme are $(n-1)$-EXPTIME.

(iii) As an application, we show that the resource usage verification problem [10] is also $(n-1)$-EXPTIME-complete, where n is the highest order of types used in the source program (written in an appropriate language [10]).

The rest of this section is organized as follows. Section 2 reviews definitions of recursion schemes and alternating parity tree automata (APT). Section 3 introduces the class of trivial APT and studies the complexity of model checking recursion schemes. Section 4 introduces the class of disjunctive APT and studies the complexity of model checking recursion schemes. Section 5 applies the result to analyze the complexity of the resource usage verification. Section 6 discusses related work and concludes the paper.

[1] Engelfriet's proof [4] is for a somewhat different (but equivalent) machine which is called *iterated pushdown automaton.*

2. Preliminaries

Let Σ be a ranked alphabet, i.e. a function that maps a terminal symbol to its arity, which is a non-negative integer. Let $\mathbb{N} = \{1, 2, \cdots\}$. A Σ-labeled (unranked) tree T is a partial map from $\mathbb{N}^*$ to $dom(\Sigma)$, such that $s\,k \in dom(T)$ (where $s \in \mathbb{N}^*, k \in \mathbb{N}$) implies $\{s\} \cup \{sj \mid 1 \leq j < k\} \subseteq dom(T)$. A (possibly infinite) sequence π over $\mathbb{N}$ is a *path* of T just if every finite prefix of π is in $dom(T)$. A tree is *ranked* just if $\max\{j \mid s\,j \in dom(T)\}$ is equal to the arity of $T(s)$ for each $s \in dom(T)$.

Higher-Order Recursion Schemes. The set of *types* is defined by:

$$\kappa ::= \mathsf{o} \mid \kappa_1 \to \kappa_2$$

where o is the type of trees. By convention, → associates to the right; thus, for example, $o \to o \to o$ means $o \to (o \to o)$. The *order* of κ, written $order(\kappa)$, is defined by:

$$\begin{array}{rcl} order(\mathsf{o}) & := & 0 \\ order(\kappa_1 \to \kappa_2) & := & \max(order(\kappa_1) + 1, order(\kappa_2)). \end{array}$$

A (deterministic) *higher-order recursion scheme* (recursion scheme, for short) is a quadruple $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, where

(i) Σ is a ranked alphabet of *terminal symbols*.
(ii) $\mathcal{N}$ is a map from a finite set of symbols called *non-terminals* to types.
(iii) $\mathcal{R}$ is a set of rewrite rules $F\ \widetilde{x} \to t$. Here $\widetilde{x} = x_1, \cdots, x_n$ abbreviates a sequence of variables, and t is an applicative term constructed from non-terminals, terminals, and variables $x_1, \cdots, x_n$.
(iv) S is a *start symbol.*

We require that $\mathcal{N}(S) = \mathsf{o}$. The set of (typed) terms is defined in the standard manner: A non-terminal or variable of type κ is a term of type κ. A terminal of arity k is a term of type $\underbrace{\mathsf{o} \to \cdots \to \mathsf{o}}_{k} \to \mathsf{o}$. If terms t_1 and t_2 have types $\kappa_1 \to \kappa_2$ and κ_1 respectively, then $t_1\ t_2$ is a term of type κ_2. By convention, application associates to the left; thus, for example, $s\,t\,u$ means $(s\,t)\,u$. For each rule $F\ \widetilde{x} \to t$, $F\ \widetilde{x}$ and t must be terms of type o. There must be exactly one rewrite rule for each non-terminal. The *order* of a recursion scheme is the highest order of (the types of) its non-terminals.

A rewrite relation on terms is defined inductively by:

(i) If $F\,\widetilde{x} \to t \in \mathcal{R}$, then $F\,\widetilde{s} \longrightarrow_{\mathcal{G}} [\widetilde{s}/\widetilde{x}]t$.
(ii) If $t \longrightarrow_{\mathcal{G}} t'$, then $t\,s \longrightarrow_{\mathcal{G}} t'\,s$ and $s\,t \longrightarrow_{\mathcal{G}} s\,t'$.

The *value tree* of a recursion scheme $\mathcal{G}$, written $[\![\mathcal{G}]\!]$, is the (possibly infinite) tree obtained by infinite rewriting of the start symbol S. More precisely, let us define $t^{\perp}$ by:

$$a^{\perp} := a \qquad F^{\perp} := \perp \qquad (t_1 t_2)^{\perp} := \begin{cases} \perp & \text{if } {t_1}^{\perp} = \perp \\ {t_1}^{\perp}\,{t_2}^{\perp} & \text{otherwise} \end{cases}$$

The value tree $[\![\mathcal{G}]\!]$ is the $\Sigma \cup \{\perp \mapsto 0\}$-ranked tree defined by:

$$[\![\mathcal{G}]\!] := \bigsqcup \{t^{\perp} \mid S \longrightarrow^*_{\mathcal{G}} t\}.$$

Here, $\bigsqcup S$ denotes the least upper bound with respect to the tree order $\sqsubseteq$ defined by

$$T_1 \sqsubseteq T_2 \iff \forall s \in dom(T_1)\,.\,(T_1(s) = T_2(s)\ \vee\ T_1(s) = \perp)$$

Note that $[\![\mathcal{G}]\!]$ is always well-defined, as the rewrite relation $\longrightarrow_{\mathcal{G}}$ is confluent.

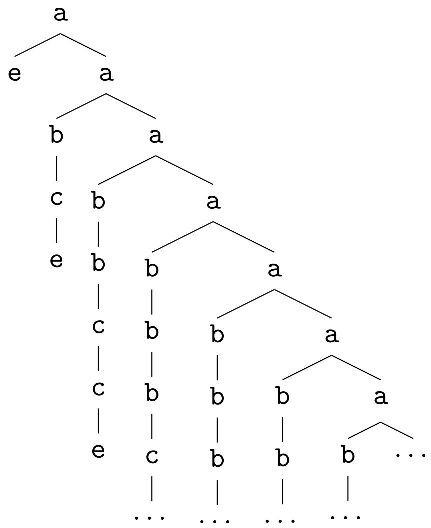

Figure 1: The tree generated by the recursion scheme of Example 1

Example 1. Consider the recursion scheme $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ where

$$\begin{array}{l}
\Sigma = \{\mathtt{a} \mapsto 2,\ \mathtt{b} \mapsto 1,\ \mathtt{c} \mapsto 1,\ \mathtt{e} \mapsto 0\} \\
\mathcal{N} = \{S \mapsto \mathtt{o}, F \mapsto (\mathtt{o} \to \mathtt{o}) \to \mathtt{o} \to \mathtt{o}, I \mapsto \mathtt{o} \to \mathtt{o}, C \mapsto (\mathtt{o} \to \mathtt{o}) \to (\mathtt{o} \to \mathtt{o}) \to (\mathtt{o} \to \mathtt{o})\} \\
\mathcal{R} = \{ \\
\quad S \to F\, I\, \mathtt{e}, \\
\quad F\, f\, x \to \mathtt{a}\,(f\, x)\,(F\,(C\,\mathtt{b}\, f)\,(\mathtt{c}\, x)), \\
\quad I\, x \to x, \\
\quad C\, f\, g\, x \to f(g\, x) \\
\}
\end{array}$$

S is reduced as follows.

$$\begin{array}{rcl}
S & \longrightarrow & F\, I\, \mathtt{e} \\
 & \longrightarrow & \mathtt{a}\,(I\,\mathtt{e})\,(F\,(C\,\mathtt{b}\, I)\,(\mathtt{c}\,\mathtt{e})) \\
 & \longrightarrow & \mathtt{a}\,\mathtt{e}\,(\mathtt{a}\,(C\,\mathtt{b}\, I\,(\mathtt{c}\,\mathtt{e}))\,(F\,(C\,\mathtt{b}\,(C\,\mathtt{b}\, I)))\,(\mathtt{c}\,(\mathtt{c}\,\mathtt{e}))) \\
 & \longrightarrow^* & \mathtt{a}\,\mathtt{e}\,(\mathtt{a}\,(\mathtt{b}\,(\mathtt{c}\,\mathtt{e}))\,(F\,(C\,\mathtt{b}\,(C\,\mathtt{b}\, I)))\,(\mathtt{c}\,(\mathtt{c}\,\mathtt{e}))) \\
 & \longrightarrow^* & \mathtt{a}\,\mathtt{e}\,(\mathtt{a}\,(\mathtt{b}\,(\mathtt{c}\,\mathtt{e}))(\mathtt{a}\,(\mathtt{b}^2(\mathtt{c}^2\,\mathtt{e}))(\mathtt{a}\,(\mathtt{b}^3(\mathtt{c}^3\,\mathtt{e}))\ \cdots)))
\end{array}$$

The value tree is shown in Figure 2. Each path of the tree is labelled by $\mathtt{a}^{m+1}\mathtt{b}^m\mathtt{c}^m\mathtt{e}$. □

Alternating parity tree automata. Given a finite set X, the set $\mathsf{B}^+(X)$ of *positive Boolean formulas* over X is defined as follows:

$$\mathsf{B}^+(X) \ni \theta \ ::= \ \mathtt{t} \mid \mathtt{f} \mid x \mid \theta \wedge \theta \mid \theta \vee \theta$$

where x ranges over X. We say that a subset Y of X *satisfies* θ just if assigning true to elements in Y and false to elements in $X \setminus Y$ makes θ true.

An *alternating parity tree automaton* (or APT for short) over Σ-labelled trees is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_I, \Omega)$ where

(i) Σ is a ranked alphabet; let m be the largest arity of the terminal symbols;
(ii) Q is a finite set of states, and $q_I \in Q$ is the initial state;
(iii) $\delta : Q \times \Sigma \longrightarrow \mathsf{B}^+(\{1, \cdots, m\} \times Q)$ is the transition function where, for each $f \in \Sigma$ and $q \in Q$, we have $\delta(q, f) \in \mathsf{B}^+(\{1, \cdots, \mathit{arity}(f)\} \times Q)$; and
(iv) $\Omega : Q \longrightarrow \{0, \cdots, M - 1\}$ is the priority function.

A *run-tree* of an APT $\mathcal{A}$ over a Σ-labelled ranked tree T is a $(\mathit{dom}(T) \times Q)$-labelled unranked tree r satisfying:

(i) $\epsilon \in \mathit{dom}(r)$ and $r(\epsilon) = (\epsilon, q_I)$; and
(ii) for every $\beta \in \mathit{dom}(r)$ with $r(\beta) = (\alpha, q)$, there is a set S that satisfies $\delta(q, T(\alpha))$; and for each $(i, q') \in S$, there is some j such that $\beta\, j \in \mathit{dom}(r)$ and $r(\beta\, j) = (\alpha\, i, q')$.

Let $\pi = \pi_1\, \pi_2 \cdots$ be an infinite path in r; for each $i \geq 0$, let the state label of the node $\pi_1 \cdots \pi_i$ be q_{n_i} where q_{n_0}, the state label of ϵ, is q_I. We say that π satisfies the *parity* condition just if the largest priority that occurs infinitely often in $\Omega(q_{n_0})\, \Omega(q_{n_1})\, \Omega(q_{n_2}) \cdots$ is even. A run-tree r is *accepting* if every infinite path in it satisfies the parity condition. An APT $\mathcal{A}$ accepts a (possibly infinite) ranked tree T if there is an accepting run-tree of $\mathcal{A}$ over T.

Ong [12] has shown that there is a procedure that, given a recursion scheme $\mathcal{G}$ and an APT $\mathcal{A}$, decides whether $\mathcal{A}$ accepts the value tree of $\mathcal{G}$.

Theorem 2.1 (Ong). *Let $\mathcal{G}$ be a recursion scheme of order n, and $\mathcal{A}$ be an APT. The problem of deciding whether $\mathcal{A}$ accepts $[\![\mathcal{G}]\!]$ is n-EXPTIME-complete.*

As usual [12], we restrict our attentions to recursion schemes whose value trees do not contain $\perp$ in the rest of the paper. Given a recursion scheme $\mathcal{G}$ that may generate $\perp$ and an APT $\mathcal{A}$, one can construct $\mathcal{G}'$ and $\mathcal{A}'$ such that (i) $\mathcal{A}$ accepts $[\![\mathcal{G}]\!]$ if and only if $\mathcal{A}'$ accepts $[\![\mathcal{G}']\!]$, and (ii) $\mathcal{G}'$ does not generate $\perp$. [2]

3. Trivial APT and the Complexity of Model Checking

APT with a trivial acceptance condition, or *trivial APT* (for short), is an APT that has exactly one priority which is even. Note that trivial APT are equivalent to Aehlig's "trivial automata" [1] (for defining languages of ranked trees).

The first result of this paper is a logical characterization of the class of Σ-labelled ranked trees accepted by trivial APT. Call $\mathcal{S}$ the following "safety fragment" of the modal mu-calculus:

$$\phi, \psi \ ::= \ P_f \mid Z \mid \phi \wedge \psi \mid \phi \vee \psi \mid \langle i \rangle \phi \mid \nu Z.\phi$$

where f ranges over symbols in a Σ, and i ranges over $\{1, \cdots, \mathit{arity}(\Sigma)\}$. We give a characterization of trivial APT. A proof is given in Appendix A.

Proposition 3.1 (Equi-Expressivity). *The logic $\mathcal{S}$ and trivial APT are equivalent for defining possibly-infinite ranked trees. I.e. for every closed $\mathcal{S}$-formula, there is a trivial APT that defines the same tree language, and vice versa.*

We show that the model checking problem for recursion schemes is n-EXPTIME complete for trivial APT. The upper-bound of n-EXPTIME follows immediately from Ong's result [12]. To show the lower-bound, we reduce the decision problem of $w \overset{?}{\in} \mathcal{L}(\mathcal{A})$, where w is a word and $\mathcal{A}$ is an order-n alternating PDA, to the model checking problem for recursion

[2] Note, however, that the transformation does not preserve the class of trivial APT considered in Section 3.

schemes. n-EXPTIME hardness follows from the reduction, since the problem of $w \overset{?}{\in} \mathcal{L}(\mathcal{A})$ is n-EXPTIME hard [4].

Definition 3.2. An *order-n alternating PDA* (order-n APDA, for short) for finite words is a 7-tuple:

$$\mathcal{A} \;=\; \langle P,\ \lambda,\ p_0,\ \Gamma,\ \Sigma,\ \Delta,\ F\rangle$$

where P is a set of states, $\lambda : P \to \{\mathtt{A}, \mathtt{E}\}$ partitions states into universal and existential, p_0 is the initial state, Γ is a stack alphabet, Σ is an input alphabet, $F \subseteq P$ is the set of final states, and $\Delta \subseteq P \times \Gamma \times (\Sigma \cup \{\epsilon\}) \times P \times \mathit{Op}_n$ is a transition relation. A *configuration* of an order-n APDA is of the form (p, s) where s is an order-n stack (an order-1 stack is an ordinary stack, and an order-$(k+1)$ stack is a stack of order-k stacks). The induced transition relation on configurations is defined by the rule:

$$\text{if } (p, \mathit{top}_1(s), \alpha, p', \theta) \in \Delta, \text{ then } (p, s) \longrightarrow_\alpha (p', \theta(s))$$

where $\theta \in \mathit{Op}_n$ is an order-n stack operation[3] and $\mathit{top}_1(s)$ is the stack top of s.

Let w be a word over Σ. We write w_i (where $0 \le i < |w|$) for the i-th element of w. A *run tree* of an order-n APDA over a word w is a *finite*, unranked tree satisfying the following.

(i) The root is labelled by $(p_0, \perp_n, 0)$, where $\perp_n$ is the empty order-n stack.

(ii) If a node is labelled by (p, s, i), then one of the following conditions holds, where $\Xi := \{(p', \theta(s), i{+}1) \mid (p, \mathit{top}_1(s), w_i, p', \theta) \in \Delta \wedge i < |w|\} \cup \{(p', \theta(s), i) \mid (p, \mathit{top}_1(s), \epsilon, p', \theta) \in \Delta\}$.
   - $p \in F$ and $i = |w|$;
   - $\lambda(p) = \mathtt{A}$ and the set of labels of the child nodes is Ξ; or
   - $\lambda(p) = \mathtt{E}$ and there is exactly one child node, which is labelled by an element of Ξ.

   (It follows that the leaves of a run tree are labelled by $(p, s, |w|)$ with $p \in F$, or (p, s, i) with $\lambda(p) = \mathtt{A}$ and $\Xi = \emptyset$.)

An order-n APDA $\mathcal{A}$ *accepts* w if there exists a run tree of $\mathcal{A}$ over w.

Engelfriet [4] has shown that the word acceptance problem for order-n APDA is n-EXPTIME complete.

Theorem 3.3 (Engelfriet)**.** *Let $\mathcal{A}$ be an order-n APDA and w a finite word over Σ. The problem of $w \overset{?}{\in} \mathcal{L}(\mathcal{A})$ is n-EXPTIME complete.*

To reduce the word acceptance problem of order-n APDA to the model checking problem for recursion schemes, we use the equivalence [9] between order-n *safe* recursion schemes and order-n PDA as (deterministic) devices for generating trees.

Definition 3.4. An *order-n tree-generating PDA* is a tuple $\mathcal{A} = \langle\, \Sigma, \Gamma, Q, \delta, q_0\rangle$ where Σ is a ranked alphabet, Γ is a stack alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial

[3] Assume an order-n stack, where $n \ge 2$. An *order-1 push* operation is just the standard operation that pushes a symbol onto the top of the top order-1 stack; the *order-1 pop* operation removes the top symbol from the top order-1 stack. For $2 \le i \le n$, the *order-i push* operation duplicates the top order-$(i-1)$ stack of the order-n stack; the *order-i pop* operation removes the top order-$(i-1)$ stack. The set Op_n of order-n stack operations consists of order-i push and order-i pop for each $1 \le i \le n$. For a formal definition, see, for example, the FoSSaCS 2002 paper [9] of Knapik et al.

state, and

$$\delta : Q \times \Gamma \longrightarrow (Q \times Op_n \cup \{(f; q_1, \cdots, q_{arity(f)}) \mid f \in \Sigma, q_i \in Q\})$$

is the transition function. A *configuration* is either a pair (q, s) where $q \in Q$ and s is an order-n stack, or a triple of the form $(f; q_1 \cdots q_{arity(f)}; s)$ where $f \in \Sigma$ and $q_1 \cdots q_{arity(f)} \in Q^*$. Let $\overline{\Sigma}$ be the label-set $\{(f, i) \mid f \in \Sigma, 1 \leq i \leq arity(f)\} \cup \{a \in \Sigma \mid arity(a) = 0\}$. We define the labelled transition relation between configurations induced by δ:

$$\begin{array}{ll}(q,s) \xrightarrow{\epsilon} (q', \theta(s)) & \text{if } \delta(q, top_1(s)) = (q', \theta) \\ (q,s) \xrightarrow{\epsilon} (f; \overline{q}; s) & \text{if } \delta(q, top_1(s)) = (f; \overline{q}) \text{ and } arity(f) \geq 1 \\ (q,s) \xrightarrow{a} (a; \epsilon; s) & \text{if } \delta(q, top_1(s)) = (a; \epsilon) \text{ and } arity(a) = 0 \\ (f; \overline{q}; s) \xrightarrow{(f,i)} (q_i, s) & \text{where } 1 \leq i \leq arity(f)\end{array}$$

Let w be a finite or infinite word over the alphabet $\overline{\Sigma}$. We say that w is a *trace* of $\mathcal{A}$ just if there is a possibly-infinite sequence of transitions $(q_0, \bot_n) \xrightarrow{\ell_1} \gamma_1 \cdots \xrightarrow{\ell_m} \gamma_m \xrightarrow{\ell_{m+1}} \cdots$ such that $w = \ell_1 \ell_2 \cdots$. We say that $\mathcal{A}$ *generates* a Σ-labelled tree t just in case the branch language[4] of t coincides with the set of traces of $\mathcal{A}$.

Theorem 3.5 (Knapik et al. [9]). *There is an effective transformation that, given an order-n tree-generating PDA $\mathcal{M}$, returns an order-n safe recursion scheme $\mathcal{G}$ that generates the same tree as $\mathcal{M}$. Moreover, both the running time of the transformation algorithm and the size of $\mathcal{G}$ are polynomial in the size of $\mathcal{M}$.*

By Theorems 3.3 and 3.5, it suffices to show that, given a word w and an order-n APDA $\mathcal{A}$, one can construct an order-n tree-generating PDA $\mathcal{M}_{\mathcal{A},w}$ and a trivial APT $\mathcal{B}$ such that w is accepted by $\mathcal{A}$ if, and only if, the tree generated by $\mathcal{M}_{\mathcal{A},w}$ is accepted by $\mathcal{B}$.

Let w be a word over Σ. From w and $\mathcal{A} = \langle P, \lambda, p_0, \Gamma, \Sigma, \Delta, F \rangle$ above, we construct an order-k PDA $\mathcal{M}_{\mathcal{A},w}$ for generating a $\{\mathtt{A}, \mathtt{E}, \mathtt{R}, \mathtt{T}\}$-labelled tree, which is a kind of run tree of $\mathcal{A}$ over the input word w. The node label $\mathtt{A}$ ($\mathtt{E}$, respectively) means that $\mathcal{A}$ is in a universal (existential, respectively) state; $\mathtt{T}$ means that $\mathcal{A}$ has accepted the word, and $\mathtt{R}$ means that $\mathcal{A}$ is stuck (no outgoing transition).

Let$N := max_{q \in P, a \in \Sigma, \gamma \in \Gamma} |\{(q', a', \theta) \mid (q, \gamma, a', q', \theta) \in \Delta, a' \in \{a, \epsilon\}\}|$. I.e. N is the degree of non-determinacy of $\mathcal{A}$.

We define

$$\mathcal{M}_{\mathcal{A},w} = \langle \{\mathtt{A} \mapsto N, \mathtt{E} \mapsto N, \mathtt{T} \mapsto 0, \mathtt{R} \mapsto 0\}, \Gamma, Q, \delta, (p_0, 0) \rangle$$

where:

- $Q = (P \times \{0, \ldots, |w|\}) \cup \{q_\top, q_\bot\} \cup (P \times \{0, \ldots, |w|\} \times Op_n)$

[4] The *branch language* of $t : dom(t) \longrightarrow \Sigma$ consists of

(i) infinite words $(f_1, d_1)(f_2, d_2) \cdots$ just if there exists $d_1 \, d_2 \cdots \in \{1, 2, \cdots, m\}^\omega$ (where m is the maximum arity of the Σ-symbols) such that $t(d_1 \cdots d_i) = f_{i+1}$ for every $i \geq 0$; and

(ii) finite words $(f_1, d_1) \cdots (f_n, d_n) \, f_{n+1}$ just if there exists $d_1 \cdots d_n \in \{1, \cdots, m\}^*$ such that $t(d_1 \cdots d_i) = f_{i+1}$ for $0 \leq i \leq n$, and the arity of f_{n+1} is 0.

- $\delta : Q \times \Gamma \longrightarrow (Q \times Op_n + \{(g;\widetilde{q}) : g \in \{\mathtt{A},\mathtt{E},\mathtt{T},\mathtt{R}\}, q_i \in Q\})$ is given by:

  (1) $\delta((p,|w|),\gamma) = (\mathtt{T};\epsilon)$, if $p \in F$

  (2) $\delta((p,i),\gamma) = (\mathtt{A};(p_1,j_1,\theta_1),\ldots,(p_m,j_m,\theta_m),\underbrace{q_\top,\ldots,q_\top}_{N-m})$
  if $\lambda(p) = \mathtt{A}$ and $\{(p_1,j_1,\theta_1),\ldots,(p_m,j_m,\theta_m)\}$ is:
  $\{(p',i+1,\theta) \mid (p,\gamma,w_i,p',\theta) \in \Delta \wedge i < |w|\} \cup \{(p',i,\theta) \mid (p,\gamma,\epsilon,p',\theta) \in \Delta\}$

  (3) $\delta((p,i),\gamma) = (\mathtt{E};(p_1,j_1,\theta_1),\ldots,(p_m,j_m,\theta_m),\underbrace{q_\bot,\ldots,q_\bot}_{N-m})$
  if $\lambda(p) = \mathtt{E}$ and $\{(p_1,j_1,\theta_1),\ldots,(p_m,j_m,\theta_m)\}$ is:
  $\{(p',i+1,\theta) \mid (p,\gamma,w_i,p',\theta) \in \Delta \wedge i < |w|\} \cup \{(p',i,\theta) \mid (p,\gamma,\epsilon,p',\theta) \in \Delta\}$

  (4) $\delta((p,i,\theta),\gamma) = ((p,i),\theta)$

  (5) $\delta(q_\top,\gamma) = (\mathtt{T};\epsilon)$

  (6) $\delta(q_\bot,\gamma) = (\mathtt{R};\epsilon)$

Rules (2) and (3) are applied only when rule (1) is inapplicable. $\mathcal{M}_{\mathcal{A},w}$ simulates $\mathcal{A}$ over the word w, and constructs a tree representing the computation of $\mathcal{A}$. A state $(p,i) \in P \times \{0,\ldots,|w|-1\}$ simulates $\mathcal{A}$ in state p reading the letter w_i. A state (p,i,θ) simulates an intermediate transition state of $\mathcal{A}$, where θ is the stack operation to be applied. The states $q_\top$ and $q_\bot$ are for creating dummy subtrees of nodes labelled with A or E, so that the number of children of these nodes adds up to N, the arity of A and E. Rule (1) ensures that when $\mathcal{A}$ has read the input word and reached a final state, $\mathcal{M}_{\mathcal{A},w}$ stops simulating $\mathcal{A}$ and outputs T. Rule (2) is used to simulate transitions of $\mathcal{A}$ in a universal state, reading the i-th input: $\mathcal{M}_{\mathcal{A},w}$ constructs a node labelled A (to record that $\mathcal{A}$ was in a universal state) and spawns threads to simulate all possible transitions of $\mathcal{A}$. Rule (3) is for simulating $\mathcal{A}$ in an existential state. Note that, if $\mathcal{A}$ gets stuck (i.e. if there is no outgoing transition), all children of the E-node are labelled R; thus failure of the computation can be recognized by the trivial APT given in the following. Rule (4) is just for intermediate transitions. Note that a transition of $\mathcal{A}$ is simulated by $\mathcal{M}_{\mathcal{A},w}$ in two steps: the first for outputting A or E, and the second for changing the stack.

Now we construct a trivial APT $\mathcal{B}$ that accepts the tree generated by $\mathcal{M}_{\mathcal{A},w}$ if, and only if, w is *not* accepted by $\mathcal{A}$. The trivial APT $\mathcal{B}$ is given by:

$$\mathcal{B} := \langle\, \{q_0\}, \{\mathtt{A},\mathtt{E},\mathtt{T},\mathtt{R}\}, q_0, \delta, \{q_0 \mapsto 0\}\rangle$$

where:

$$\delta(q_0,\mathtt{A}) = \textstyle\bigvee_{i=1}^{N}(i,q_0) \quad \delta(q_0,\mathtt{E}) = \bigwedge_{i=1}^{N}(i,q_0) \quad \delta(q_0,\mathtt{T}) = \mathsf{f} \quad \delta(q_0,\mathtt{R}) = \mathtt{t}$$

Intuitively, $\mathcal{B}$ accepts all trees representing a failure computation tree of $\mathcal{A}$. If the automaton in state q_0 reads T (which corresponds to an accepting state of $\mathcal{A}$), it gets stuck. Upon reading A, the automaton non-deterministically chooses one of the subtrees, and checks whether the subtree represents a failure computation of $\mathcal{A}$. On the other hand, upon reading E, the automaton checks that all subtrees represent failure computation trees of $\mathcal{A}$.

By the above construction, we have:

Theorem 3.6. *Let w be a word, and $\mathcal{A}$ an order-n APDA. Then w is* not *accepted by $\mathcal{A}$ if, and only if, the tree generated by $\mathcal{M}_{\mathcal{A},w}$ is accepted by $\mathcal{B}$.*

Corollary 3.7. *The trivial APT acceptance problem for the tree generated by an order-n recursion scheme (i.e. whether the tree generated by a given order-n recursion scheme is accepted by a given trivial APT) is n-EXPTIME hard in the size of the recursion scheme.*

By modifying the encoding, we can also show that the model checking problem is n-EXPTIME-hard in the size of the APT. The idea is to modify $\mathcal{M}_{\mathcal{A},w}$ so that it generates a tree representing computation of $\mathcal{A}$ over not just w but all possible input words, and let a trivial APT check the part of the tree corresponding to the input word w. As a result, the trivial APT depends on the input word w, but the tree-generating PDA does not.

We make the following two assumptions on $\mathcal{A}$ (without loss of generality):

(i) In each state, if $\mathcal{A}$ can perform an ϵ-transition, then $\mathcal{A}$ cannot perform any input transition i.e. $\{(p',\theta) \mid \exists a \in \Sigma.(p,\gamma,a,p',\theta) \in \Delta\} \neq \emptyset$ implies $\{(p',\theta) \mid (p,\gamma,\epsilon,p',\theta) \in \Delta\} = \emptyset$.

(ii) There is no transition from a final state i.e. if $p \in F$ then $\{(p',\theta) \mid \exists a \in \Sigma \cup \{\epsilon\}.(p,\gamma,a,p',\theta) \in \Delta\} = \emptyset$.

Given an order-n APDA $\mathcal{A}$ and a word w, we shall construct $\mathcal{M}'_{\mathcal{A}}$ and $\mathcal{B}_w$, such that w is *not* accepted by $\mathcal{A}$ if, and only if, the tree generated by $\mathcal{M}'_{\mathcal{A}}$ is accepted by $\mathcal{B}_w$. The difference from the construction of $\mathcal{M}_{\mathcal{A},w}$ and $\mathcal{B}$ above is that $\mathcal{M}'_{\mathcal{A}}$ does not depend on w. The idea is to let $\mathcal{M}'_{\mathcal{A}}$ generate a tree representing the computations of $\mathcal{A}$ over all possible inputs. We then let $\mathcal{B}_w$ traverse the part of the tree corresponding to the computation over w, and check whether the computation is successful.

We define a tree-generating PDA $\mathcal{M}'_{\mathcal{A}} = \langle \Sigma', \Gamma, Q, \delta, q_0 \rangle$ where:

- $\Sigma' = \{\mathtt{Read} \mapsto |\Sigma|, \mathtt{Accept} \mapsto 0, \mathtt{Epsilon} \mapsto 1, \mathtt{A} \mapsto N, \mathtt{E} \mapsto N, \mathtt{T} \mapsto 0, \mathtt{R} \mapsto 0\}$
- $Q = P \ \cup \ (P \times (\Sigma \cup \{\epsilon\})) \ \cup \ \{q_\top, q_\bot\} \ \cup \ (P \times Op_k)$
- $q_0 = p_0$
- δ is given by:

$$\begin{array}{l}
\delta(p,\gamma) = (\mathtt{Accept}; \epsilon) \text{ if } p \in F \\
\delta(p,\gamma) = (\mathtt{Epsilon}; ((p,\epsilon), \mathtt{id})) \text{ if } \{(p',\theta) \mid (p,\gamma,\epsilon,p',\theta) \in \Delta\} \neq \emptyset. \\
\delta(p,\gamma) = (\mathtt{Read}; ((p,a_1),\mathtt{id}), \ldots, ((p,a_n),\mathtt{id})) \\
\qquad \text{if } p \notin F,\ \{(p',\theta) \mid (p,\gamma,\epsilon,p',\theta) \in \Delta\} = \emptyset \text{ and } \Sigma = \{a_1,\ldots,a_n\}. \\
\delta((p,\alpha),\gamma) = (\mathtt{A}; ((p_1,\theta_1),\ldots,(p_m,\theta_m), q_\top,\ldots,q_\top)) \\
\qquad \text{if } \lambda(p) = \mathtt{A} \text{ and} \\
\qquad\quad \{(p_1,\theta_1),\ldots,(p_m,\theta_m)\} = \{(p',\theta) \mid (p,\gamma,\alpha,p',\theta) \in \Delta\} \\
\delta((p,\alpha),\gamma) = (\mathtt{E}; ((p_1,\theta_1),\ldots,(p_m,\theta_m), q_\bot,\ldots,q_\bot)) \\
\qquad \text{if } \lambda(p) = \mathtt{E} \text{ and} \\
\qquad\quad \{(p_1,\theta_1),\ldots,(p_m,\theta_m)\} = \{(p',\theta) \mid (p,\gamma,\alpha,p',\theta) \in \Delta\} \\
\delta((p,\theta),\gamma) = (p,\theta) \\
\delta(q_\top,\gamma) = (\mathtt{T}; \epsilon) \\
\delta(q_\bot,\gamma) = (\mathtt{R}; \epsilon)
\end{array}$$

In a final state of $\mathcal{A}$, $\mathcal{M}'_{\mathcal{A}}$ outputs a node labelled with `Accept`, to indicate that $\mathcal{A}$ has reached a final state, and stops simulating $\mathcal{A}$ (as, by assumption (ii) above, there is no outgoing transition). In a state where $\mathcal{A}$ has ϵ-transitions, $\mathcal{M}'_{\mathcal{A}}$ outputs a node labelled with `Epsilon`, and then simulates all the possible ϵ-transitions of $\mathcal{A}$. In a state where $\mathcal{A}$ has input transitions, $\mathcal{M}'_{\mathcal{A}}$ outputs a ndoe labelled with `Read` to indicate that $\mathcal{A}$ makes an input transition, and then simulates the input transition for each possible input symbol. Note that by the assumptions (i) and (ii) above, these three transitions are disjoint. The remaining transition rules are analogous to those of $\mathcal{M}_{\mathcal{A},w}$.

Define the trivial APT $\mathcal{B}_w$ by $\mathcal{B}_w = \langle \Sigma', Q', \delta, q_0, \Omega \rangle$ where:

$$\begin{array}{l}
Q' = \{q_0, \dots, q_{|w|}\} \\
\delta(q, \mathtt{Epsilon}) = (1, q) \text{ for every } q \in Q' \\
\delta(q_i, \mathtt{Read}) = (j, q_{i+1}) \text{ if } 0 \le i \le |w| - 1 \text{ and } w_i = a_j \\
\delta(q_{|w|}, \mathtt{Read}) = \mathtt{t} \\
\delta(q_i, \mathtt{A}) = (1, q_i) \vee \dots \vee (N, q_i) \\
\delta(q_i, \mathtt{E}) = (1, q_i) \wedge \dots \wedge (N, q_i) \\
\delta(q_{|w|}, \mathtt{Accept}) = \mathtt{f} \\
\delta(q_i, \mathtt{Accept}) = \mathtt{t} \text{ for every } 0 \le i < |w| \\
\delta(q, \mathtt{T}) = \mathtt{f} \text{ for every } q \in Q' \\
\delta(q, \mathtt{R}) = \mathtt{t} \text{ for every } q \in Q'
\end{array}$$

and Ω is the trivial priority function.

The trivial APT $\mathcal{B}_w$ traverses the tree generated by $\mathcal{M}'_{\mathcal{A}}$ (which represents transitions of $\mathcal{A}$ for all possible inputs), while keeping track of the position of the input head of $\mathcal{A}$ in its state (q_i means that $\mathcal{A}$ is reading the i-th letter of the word w). Upon reading `Read` in state q_i, $\mathcal{B}_w$ proceeds to traverse the branch corresponding to the i-th letter (i.e. w_i). Reading `Accept` in state $q_{|w|}$ means that $\mathcal{A}$ accepts the word w, so that the run of $\mathcal{B}_w$ fails (recall that $\mathcal{B}_w$ accepts the tree just if $\mathcal{A}$ does *not* accept w). Reading `Accept` in state q_i (with $i < |w|$) on the other hand means that $\mathcal{A}$ does not accept w, so that the run of $\mathcal{B}_w$ succeeds. The remaining transition rules are analogous to those of $\mathcal{B}$.

By the construction above, w is *not* accepted by $\mathcal{A}$ if, and only if, the tree generated by $\mathcal{M}'_{\mathcal{A}}$ is accepted by $\mathcal{B}_w$. Since only $\mathcal{B}_w$ depends on the input word w, we get:

Theorem 3.8. *The trivial APT acceptance problem of trees generated by order-n recursion schemes is n-EXPTIME-hard in the size of the APT.*

To our knowledge, the lower bound (of the complexity of model-checking recursion schemes) in terms of the size of APT for the entire class of APT is new.

4. Disjunctive APT and Complexity of Model Checking

A *disjunctive APT* is an APT whose transition function δ is disjunctive, i.e. δ maps each state to a positive boolean formula θ that contains only disjunctions and no conjunctions, as given by the grammar $\theta ::= \mathtt{t} \mid \mathtt{f} \mid (i, q) \mid \theta \vee \theta$. Disjunctive APT can be used to describe path (or linear-time) properties of trees.

First we give a logical characterization of disjunctive APT as follows. Call $\mathcal{D}$ the following "disjunctive fragment" of the modal mu-calculus:

$$\phi, \psi ::= P_f \wedge \phi \mid Z \mid \phi \vee \psi \mid \langle i \rangle \phi \mid \nu Z.\phi \mid \mu Z.\phi$$

where f ranges over symbols in Σ, and i over $\{1, \cdots, m\}$ where m is the largest arity of the symbols in Σ. A proof of the following proposition is given in Appendix A.

Proposition 4.1 (Equi-Expressivity). *The logic $\mathcal{D}$ and disjunctive APT are equivalent for defining possibly-infinite ranked trees. I.e. for every closed $\mathcal{D}$-formula, there is a disjunctive APT that defines the same tree language, and vice versa.*

Remarks 4.2. For defining languages of ranked trees, disjunctive APT are a proper subset of the *disjunctive formulas* in the sense of Walukiewicz and Janin [7]. For example, the disjunctive formula $(1 \to \{\mathtt{t}\}) \wedge (2 \to \{\mathtt{t}\})$ is not equivalent to any disjunctive APT.

In the rest of the section, we show that the model checking problem for order-n recursion schemes is $(n-1)$-EXPTIME complete for disjunctive APT.

4.1. **Upper Bound.** Since our proof is based on Kobayashi and Ong's type system for recursion schemes [11] and relies heavily on the machinery and techniques developed therein, we shall just sketch a proof here; a detailed proof will be presented in the journal version of [11]. An alternative proof, also sketched but based on variable profiles [12], is given in Appendix B.

Theorem 4.3. *Let $\mathcal{G}$ be an order-n recursion scheme and $\mathcal{B}$ a disjunctive APT. It is decidable in $(n-1)$-EXPTIME whether $\mathcal{B}$ accepts the value tree $[\![\mathcal{G}]\!]$.*

In a recent paper [11], we constructed an intersection type system equivalent to the modal mu-calculus model checking of recursion schemes, in the sense that for every APT, there is a type system such that the tree generated by a recursion scheme is accepted by the APT if, and only if, the recursion scheme is typable in the type system. Thus, the model checking problem is reduced to a type checking problem. The main idea of the type system is to refine the tree type o by the states and priorities of an APT: the type q describes a tree that is accepted by the APT with q as the start state. The intersection type $(\theta_1, m_1) \wedge (\theta_2, m_2) \to q$, which refines the type $\mathsf{o} \to \mathsf{o}$, describes a tree function that takes an argument which has types θ_1 and θ_2, and returns a tree of type q.

The type checking algorithm presented in *ibid.* is n-EXPTIME in the combined size of the order-n recursion scheme and APT (precisely the complexity is $O(r^{1+\lfloor m/2 \rfloor}\mathbf{exp}_n((a\,|Q|\,m)^{1+\epsilon}))$[5] for $n \geq 2$, where r is the number of rules and a the largest arity of the symbols in the scheme, m is the largest priority, $|Q|$ is the number of states). The bottleneck of the algorithm is the number of (atomic) intersection types, where the set $\mathcal{T}(\kappa)$ of atomic types refining a simple type κ is inductively defined by:

$$\begin{array}{rcl} \mathcal{T}(\mathsf{o}) & := & Q \\ \mathcal{T}(\kappa_1 \to \kappa_2) & := & \{\bigwedge S \to \theta \mid \theta \in \mathcal{T}(\kappa_2), S \subseteq \mathcal{T}(\kappa_1) \times P\} \end{array}$$

where Q and P are the sets of states and priorities respectively.

According to the syntax of atomic types above, the number of atomic types refining a simple type of order n is n-exponential in general. In the case of disjunctive APT, however, for each type of the form $\mathsf{o} \to \cdots \to \mathsf{o} \to \mathsf{o}$, we need to consider only atomic types of the form $\bigwedge S_1 \to \cdots \to \bigwedge S_k \to q$, where at most one of the S_i's is a singleton set and the other S_j's are empty. Intuitively, this is because a run-tree of a disjunctive APT consists of a single path, so that the run-tree visits only one of the arguments, at most once. In fact, we can show that, if a recursion scheme is typable in the type system for a disjunctive APT, the recursion scheme is typable in a restricted type system in which order-1 types are constrained as described above: this follows from the proof of completeness of the type system [11], along with the property of the accepting run-tree mentioned above. Thus, the number of atomic types is $k \times |Q| \times |P| \times |Q|$ (whereas it is exponential for an arbitrary APT). Therefore, the number of atomic types possibly assigned to a symbol of order n is $(n-1)$-exponential. By running the same type checking algorithm as *ibid.* (but with order-1 types constrained as above), order-n recursion schemes can be type-checked (i.e. model-checked) in $(n-1)$-EXPTIME.

[5] According to Schewe's recent result [13] on the complexity of parity games, the part $r^{1+\lfloor m/2 \rfloor}$ can be further reduced to roughly $r^{1+m/3}$.

4.2. **Lower Bound.** We show the lower bound by a reduction of the emptiness problem of the finite-word language accepted by an order-n deterministic PDA, which is $(n-1)$-EXPTIME complete [4].

Let $\mathcal{A}$ be an order-n deterministic PDA, given by $\mathcal{A} = \langle P, p_0, \Gamma, \Sigma, \delta, F\rangle$ where δ is a partial function from $P \times \Gamma \times (\Sigma \cup \{\epsilon\})$ to $P \times Op_n$. We shall construct an order-n tree-generating PDA $\mathcal{M}_\mathcal{A}$, which simulates all possible input and ϵ-transitions of $\mathcal{A}$, and outputs $\mathtt{e}$ only when $\mathcal{A}$ reaches a final state.

The order-n PDA $\mathcal{M}_\mathcal{A}$ is given by:

$$\mathcal{M}_\mathcal{A} = \langle \{\mathtt{e} \mapsto 0\} \cup \{\mathtt{br}_m \mapsto m \mid 0 \leq m \leq N\}, \Gamma, P \cup (P \times Op_n), \delta', p_0\rangle$$

$$\begin{array}{l}
N = max_{p\in P,\gamma\in\Gamma}|\{(p',\theta') \mid \exists\alpha \in \Sigma\cup\{\epsilon\}.\delta(p,\alpha,\gamma) = (p',\theta')\}| \\
\delta'(p,\gamma) = (\mathtt{e};\epsilon) \text{ if } p \in F \\
\delta'(p,\gamma) = (\mathtt{br}_m;(p_1,\theta_1),\ldots,(p_m,\theta_m)) \\
\qquad \text{if } p \notin F \text{ and } \{(p_1,\theta_1),\ldots,(p_m,\theta_m)\} = \{(p',\theta') \mid \exists\alpha \in \Sigma\cup\{\epsilon\}.\delta(p,\alpha,\gamma) = (p',\theta')\} \\
\delta'((p,\theta),\gamma) = (p,\theta)
\end{array}$$

A state of $\mathcal{M}_\mathcal{A}$ is either a state of $\mathcal{A}$ (i.e. an element of P), or a pair (p,θ). In state $p \in P$, $\mathcal{M}_\mathcal{A}$ constructs a node labeled by $\mathtt{br}_m$, and spawns subtrees for simulating possible input or ϵ-transitions of $\mathcal{A}$ from state p.

By a result of Knapik et al. [9], we can construct an equi-expressive order-n safe recursion scheme $\mathcal{G}$. Let $\mathcal{G}'$ be the recursion scheme obtained from $\mathcal{G}$ by (i) replacing each terminal symbol $\mathtt{br}_m (m > 2)$ with a non-terminal Br_m of the same arity, and (ii) adding the rule:

$$Br_m\, x_1 \cdots x_m \to \mathtt{br}_2\, x_1\, (\mathtt{br}_2\, x_2 (\cdots (\mathtt{br}_2\, x_{m-1}\, x_m))).$$

By the construction, the finite word-language accepted by $\mathcal{A}$ is non-empty if, and only if, the value tree of $\mathcal{G}'$ has a node labelled $\mathtt{e}$. The latter property can be expressed by a disjunctive APT. (The purpose of transforming $\mathcal{G}$ into $\mathcal{G}'$ was to make the disjunctive APT independent of $\mathcal{A}$.) Thus, we have:

Theorem 4.4. *The disjunctive APT acceptance problem for the tree generated by an order-n recursion scheme is $(n-1)$-EXPTIME-hard in the size of the recursion scheme.*

The problem is $(n-1)$-EXPTIME hard also in the size of the disjunctive APT.

As above, let $\mathcal{A} = \langle P, p_0, \Gamma, \Sigma, \delta, F\rangle$ be an order-n deterministic PDA for words. We may assume that the stack alphabet is $\{\gamma_0, \gamma_1\}$ (as we can encode an arbitrary stack symbol as a sequence of γ_0 and γ_1).

We first define an order-n tree-generating PDA $\mathcal{M}$ by:

$$\begin{array}{l}
\mathcal{M} = \langle\, \{\gamma_0,\gamma_1\}, \{\gamma_0,\gamma_1\}, \{q_0,\theta_1,\ldots,\theta_k\}, q_0, \delta\rangle \\
\delta(q_0,\gamma_i) = (\gamma_i;\theta_1,\ldots,\theta_k) \\
\delta(\theta_i,\gamma_j) = (q_0,\theta_i)
\end{array}$$

where $\{\theta_1,\ldots,\theta_k\}$ is the set of order-n stack operations. The role of $\mathcal{M}$ is to generate a tree simulating all the possible changes of the stack top. Note that $\mathcal{M}$ is independent of $\mathcal{A}$.

Now let us define a disjunctive APT $\mathcal{D}_\mathcal{A} = \langle\, P, \{\gamma_0,\gamma_1\}, \delta', p_0, \Omega\rangle$ as follows.

$$\delta'(p,\gamma_i) \;=\; \left\{\begin{array}{ll} \bigvee\{(j,p') | \exists\alpha.\delta(p,\gamma_i,\alpha) = (p',\theta_j)\} & \text{if } p \notin F \\ \mathtt{t} & \text{if } p \in F \end{array}\right.$$

$$\Omega(p) = \left\{\begin{array}{ll} 2 & \text{if } p \in F \\ 1 & \text{otherwise} \end{array}\right.$$

The idea of the above encoding is to let $\mathcal{D}_{\mathcal{A}}$ simulate transitions of $\mathcal{A}$, while extracting information about the stack top from the tree generated by $\mathcal{M}$. Let $\mathcal{G}$ be an order-n recursion scheme that generates the same tree as $\mathcal{M}$. By the above construction, the language of $\mathcal{A}$ is non-empty if, and only if, $\mathcal{D}_{\mathcal{A}}$ accepts the tree generated by $\mathcal{G}$. Since the size of $\mathcal{G}$ does not depend on $\mathcal{A}$, and the size of $\mathcal{D}_{\mathcal{A}}$ is polynomial in the size of $\mathcal{A}$, we have:

Theorem 4.5. *The disjunctive APT acceptance problem for trees generated by order-n recursion schemes is $(n-1)$-EXPTIME hard in the size of the APT.*

4.3. **Path Properties.** The *path language* of a Σ-labelled tree t is the image of the map F, which acts on the elements of the branch language of t by "forgetting the argument positions" i.e.

$$F \ : \ \begin{cases} (f_1, d_1)\,(f_2, d_2)\cdots & \mapsto \quad f_1\, f_2 \cdots \\ (f_1, d_1)\cdots(f_n, d_n)\, f_{n+1} & \mapsto \quad f_1 \cdots f_n\, f_{n+1}^{\omega}. \end{cases}$$

For example $\{f\,a^{\omega}, f\,f\,a^{\omega}, f\,f\,b^{\omega}\}$ is the path language of the term-tree $f\,a\,(f\,a\,b)$. Let $\mathcal{G}$ be a recursion scheme. We write $\mathcal{W}\,(\mathcal{G})$ for the *path language* of $[\![\mathcal{G}]\!]$. Thus elements of $\mathcal{W}\,(\mathcal{G})$ are infinite words over the alphabet Σ which is now considered unranked (i.e. arities of the symbols are forgotten).

Theorem 4.6. *Let $\mathcal{G}$ be an order-n recursion scheme. The following problems are $(n-1)$-EXPTIME complete.*

(i) $\mathcal{W}\,(\mathcal{G}) \cap \mathcal{L}(\mathcal{C}) \overset{?}{=} \emptyset$*, where $\mathcal{C}$ is a* non-deterministic *parity word automaton.*

(ii) $\mathcal{W}\,(\mathcal{G}) \overset{?}{\subseteq} \mathcal{L}(\mathcal{C})$*, where $\mathcal{C}$ is a* deterministic *parity word automaton.*

Furthermore, the problem (i) is $n-1$-EXPTIME hard not only in the size of $\mathcal{G}$ but also in the size of $\mathcal{C}$.

Proof. (i) Let $\mathcal{C} = \langle\, Q,\, \Sigma,\, \Delta,\, \Omega \rangle$ be a *non-deterministic* parity word automaton, where $\Delta \subseteq Q \times \Sigma \times Q$ and $\Omega : Q \longrightarrow \{0, \cdots, p\}$. Let m be the largest arity of the symbols in Σ. (Büchi automata are equivalent to parity automata with two priorities.) We have $\mathcal{W}\,(\mathcal{G}) \cap \mathcal{L}(\mathcal{C}) \neq \emptyset$ if, and only if, $[\![\mathcal{G}]\!]$ is accepted by the APT $\mathcal{B} = \langle\, Q,\, \Sigma,\, \delta,\, \Omega \rangle$ where $\delta : Q \times \Sigma \longrightarrow \mathsf{B}^{+}(\{1, \cdots, m\} \times Q)$ is a *disjunctive* transition function

$$\delta \ : \ (q, f) \ \mapsto \ \bigvee \{(i, p) : 1 \le i \le \Sigma(f), (q, f, p) \in \Delta\}.$$

It follows from Theorem 4.3 that the problem $\mathcal{W}\,(\mathcal{G}) \cap \mathcal{L}(\mathcal{C}) \overset{?}{=} \emptyset$ can be decided in $(n-1)$-EXPTIME.

Let $\mathcal{C}$ be a parity word automaton that accepts $\Sigma^{*}\,\mathtt{e}^{\omega}$, and $\mathcal{G}'$ be the recursion scheme in Section 4.2. Then, $\mathcal{W}\,(\mathcal{G}') \cap \mathcal{L}(\mathcal{C}) \neq \emptyset$ if, and only if, $\mathcal{G}'$ has a node labelled $\mathtt{e}$. Thus, the problem $\mathcal{W}\,(\mathcal{G}) \cap \mathcal{L}(\mathcal{C}) \overset{?}{=} \emptyset$ is $(n-1)$-EXPTIME-hard in the size of $\mathcal{G}$.

To show the lower bound in the size of $\mathcal{C}$, we modify the construction of $\mathcal{M}$ and $\mathcal{D}_{\mathcal{A}}$ as follows. Let $\mathcal{M}'$ be the order-n tree-generating PDA given by:

$$\begin{array}{l} \mathcal{M} := \langle\, \{\gamma_0, \gamma_1, \theta_1, \ldots, \theta_k\}, \{\gamma_0, \gamma_1\}, \{q_0, q_1, \ldots, q_k, \theta_1, \ldots, \theta_k,\}, q_0, \delta \rangle \\ \delta(q_0, \gamma_i) = (\gamma_i; q_1, \ldots, q_k) \text{ for } 0 \le i \le 1 \\ \delta(q_j, \gamma_i) = (\theta_i; \theta_i) \text{ for } 0 \le i \le 1, 1 \le j \le k \\ \delta(\theta_j, \gamma_i) = (q_0, \theta_i) \text{ for } 0 \le i \le 1, 1 \le j \le k \end{array}$$

The difference from $\mathcal{M}$ is that $\mathcal{M}'$ outputs not only stack top symbols but also stack operations (which were coded as branch information in the case of $\mathcal{M}$). Let $\mathcal{C}_{\mathcal{A}}$ be the non-deterministic parity word automaton given by:

$$\begin{array}{l}
\mathcal{C}_{\mathcal{A}} := \langle\, P \cup (P \times \{0,1\}), \{\gamma_0, \gamma_1\}, \delta', p_0, \Omega \rangle \\
\delta'(p, \gamma_i) = \{(p, i)\} \text{ if } p \notin F \\
\delta'((p, i), \theta_j) = \{p' \mid \exists \alpha.\delta(p, \gamma_i, \alpha) = (p', \theta_j)\} \\
\delta'(p, \gamma_i) = \{p\} \text{ if } p \in F \\
\delta'(p, \theta_j) = \{p\} \\
\Omega(p) = \begin{cases} 2 & \text{if } p \in F_{\mathcal{A}} \\ 1 & \text{otherwise} \end{cases}
\end{array}$$

Let $\mathcal{G}$ be a recursion scheme that generates the same tree as $\mathcal{M}'$. Then, the language of $\mathcal{A}$ is empty if, and only if, $\mathcal{W}(\mathcal{G}) \cap \mathcal{L}(\mathcal{C}_{\mathcal{A}}) = \emptyset$. Since $\mathcal{G}$ does not depend on $\mathcal{A}$, $\mathcal{W}(\mathcal{G}) \cap \mathcal{L}(\mathcal{C}) \overset{?}{=} \emptyset$ is $(n-1)$-EXPTIME hard also in the size of $\mathcal{C}$.

(ii) Let $\mathcal{C}$ be a *deterministic* parity word automaton $\mathcal{C} = \langle\, Q, \Sigma, \delta_{\mathcal{C}}, q_0, \Omega \rangle$, where $\delta_{\mathcal{C}} : Q \times \Sigma \longrightarrow Q$ and $\Omega : Q \longrightarrow \{0, \cdots, p\}$. Define $\overline{A} = \langle\, Q, \Sigma, \delta_{\mathcal{C}}, q_0, \overline{\Omega} \rangle$ where $\overline{\Omega} : q \mapsto (\Omega(q) + 1)$. Note that because of determinacy, $\mathcal{L}(\overline{\mathcal{C}}) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{C})$. Now we have $\mathcal{W}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{C})$ if, and only if, $\mathcal{W}(\mathcal{G}) \cap \mathcal{L}(\overline{\mathcal{C}}) = \emptyset$. Thus, the problem $\mathcal{W}(\mathcal{G}) \overset{?}{\subseteq} \mathcal{L}(\mathcal{C})$ is $(n-1)$-EXPTIME. Moreover, since the language $\Sigma^* \, \mathtt{e}^\omega$ is accepted by a deterministic parity word automaton, the problem is also $(n-1)$-EXPTIME hard (in the size of $\mathcal{G}$).

□

The decision problems Reachability (i.e. whether $[\![\mathcal{G}]\!]$ has a node labelled by a given symbol $\mathtt{e}$) and Finiteness (i.e. whether $[\![\mathcal{G}]\!]$ is finite) are instances of Problem (i) of Theorem 4.6; hence they are in $(n-1)$-EXPTIME (the former is $(n-1)$-EXPTIME complete, by the proof of Section 4.2).

Consider the problem LTL Model-Checking:

> "Given an LTL-formula ϕ (generated from atomic propositions of the form P_f with $f \in \Sigma$) and an order-n recursion scheme $\mathcal{G}$, does every path in $[\![\mathcal{G}]\!]$ satisfy ϕ? (Precisely, is $\mathcal{W}(\mathcal{G}) \subseteq [\![\phi]\!]$?)"

As a corollary of Theorem 4.6, we have:

Corollary 4.7. LTL Model-Checking *(i.e. given order-n recursion scheme $\mathcal{G}$ and LTL-formula ϕ, is $\mathcal{W}(\mathcal{G}) \subseteq [\![\phi]\!]$?) is $(n-1)$-EXPTIME complete in the size of $\mathcal{G}$.*

Proof. The upper bound follows from Theorem 4.6(i): note that $\mathcal{W}(\mathcal{G}) \subseteq [\![\phi]\!]$ is equivalent to $\mathcal{W}(\mathcal{G}) \cap [\![\neg\phi]\!] = \emptyset$, and because $[\![\neg\phi]\!]$ is ω-regular, it is recognizable [14] by a parity automaton.

The lower bound follows from the $(n-1)$-EXPTIME hardness of Reachability: checking whether a recursion scheme satisfies the formula $G(\neg\mathtt{e})$ is $(n-1)$-EXPTIME hard in the size of the recursion scheme. □

Note however that LTL Model-Checking is n-EXPTIME in the size of the LTL-formula ϕ, as the size of the corresponding parity word automaton is exponential in ϕ in general [15].

5. Application to Resource Usage Verification

Now we apply the result of the previous section to show that the resource usage verification problem [6] is $(n-1)$-EXPTIME complete. The aim of resource usage verification is to check whether a program accesses each resource according to a given resource specification. For example, consider the following program.

```
let rec g x = if rand() then close(x) else read(x); g(x) in
let r = open_in "foo" in g(r)
```

Here, `rand()` returns a non-deterministic boolean. The program first defines a recursive function `g` that takes a file pointer `x` as an argument parameter, closes it after some read operations. The program then opens a read-only file "`foo`", and passes it to `g`. For this program, the goal of the verification is to statically check that the file is eventually closed before the program terminates, and after it is closed, it is never read from or written to.

Kobayashi [10] recently showed that the resource usage verification problem is decidable for the simply-typed λ-calculus with recursion, generated from a base type of booleans, and augmented by resource creation/access primitives, by reduction to the model checking problem for recursion schemes. Prior to Kobayashi's work [10], only sound but incomplete verification methods have been proposed.

Following [10], we consider below a simply-typed, call-by-name functional language with only top-level function definitions and resource usage primitives.[6] A *program* is a triple $(D, S, \mathcal{C})$ where D is a set of function definitions, S is a function name (representing the main function), and $\mathcal{C} = (Q_{\mathcal{C}}, \Sigma_{\mathcal{C}}, \delta_{\mathcal{C}}, q_{0,\mathcal{C}}, F_{\mathcal{C}})$ is a deterministic word automaton, which describes how the state of a resource is changed by each access primitive. A function definition is of the form $F\ \widetilde{x} = e$, where e is given by:

$$e ::= \star \mid x \mid F \mid e_1 e_2 \mid \mathbf{If}{*}\ e_1\ e_2 \mid \mathbf{New}^q\ e \mid \mathbf{Acc}_a\ e_1\ e_2$$

The term $\star$ is the unit value. The term $\mathbf{If}{*}\ e_1\ e_2$ is a non-deterministic branch between e_1 and e_2. The term $\mathbf{New}^q\ e$ creates a fresh resource, and passes it to e (which is a function that takes a resource as an argument). Here, q represents the initial state of a resource; the automaton $\mathcal{C}$ specifies how the resource should be accessed afterwards: see the operational semantics given later. The term $\mathbf{Acc}_a\ e_1\ e_2$ accesses the resource e_1 with the primitive of name $a (\in \Sigma_{\mathcal{C}})$ and then executes e_2.

Programs must be simply typed; the two base types are **unit** for unit values and **R** for resources. The body of each definition must have type **unit** (in other words, resources cannot be used as return values; this requirement can be enforced by the CPS transformation). The constants $\mathbf{If}{*}$, $\mathbf{New}^L$, and $\mathbf{Acc}_a$ are given the following types.

$$\mathbf{If}{*} : \mathbf{unit} \to \mathbf{unit} \to \mathbf{unit}, \mathbf{New}^L : (\mathbf{R} \to \mathbf{unit}) \to \mathbf{unit}, \mathbf{Acc}_a : \mathbf{R} \to \mathbf{unit} \to \mathbf{unit}$$

Example 5.1. The program given at the beginning of this section can be expressed as $(D, S, \mathcal{C})$ where

$$\begin{array}{l} D = \{S = \mathbf{New}^{q_1}\ (G\ \star), G\ k\ x = \mathbf{If}{*}\ (\mathbf{Acc}_c\ x\ k)\ (\mathbf{Acc}_r\ x\ (G\ k\ x))\} \\ \mathcal{C} = (\{q_1, q_2\}, \{\mathtt{r}, \mathtt{c}\}, \delta, q_1, \{q_2\}) \\ \delta(q_1, \mathtt{r}) = q_1 \qquad \delta(q_1, \mathtt{c}) = q_2 \end{array}$$

[6] Note that programs in call-by-value languages can be transformed into this language by using the standard CPS transformation and λ-lifting.

Here, G corresponds to the function g in the original program, and the additional parameter k represents a continuation. The automaton $\mathcal{C}$ specifies that the resource should be accessed according to $\mathtt{r}^*\mathtt{c}$.

We introduce the operational semantics to formally define the resource usage verification problem. A run-time state is either an error state **Error** or a pair (ρ, e) where ρ is a finite map from variables to $Q_{\mathcal{C}}$, which represents the state of each resource. The reduction relation $\longrightarrow_{D,\mathcal{C}}$ on run-time states is defined by:

$$\frac{F\ \widetilde{x} = e' \in D}{(\rho, F\ \widetilde{e}) \longrightarrow_{D,\mathcal{C}} (\rho, [\widetilde{e}/\widetilde{x}]e')}$$

$$\frac{}{(\rho, \mathbf{If}*\ e_1\ e_2) \longrightarrow_{D,\mathcal{C}} (\rho, e_1)}$$

$$\frac{}{(\rho, \mathbf{If}*\ e_1\ e_2) \longrightarrow_{D,\mathcal{C}} (\rho, e_2)}$$

$$\frac{x \notin dom(\rho)}{(\rho, \mathbf{New}^q\ e) \longrightarrow_{D,\mathcal{C}} (\rho\{x \mapsto q\}, e\,x)}$$

$$\frac{\delta_{\mathcal{C}}(q, a) = q'}{(\rho\{x \mapsto q\}, \mathbf{Acc}_a\ x\ e) \longrightarrow_{D,\mathcal{C}} (\rho\{x \mapsto q'\}, e)}$$

$$\frac{\delta_{\mathcal{C}}(q, a) \text{ is undefined}}{(\rho\{x \mapsto q\}, \mathbf{Acc}_a\ x\ e) \longrightarrow_{D,\mathcal{C}} \mathbf{Error}}$$

Example 5.2. Recall the program in Example 5.1. It can be reduced as follows.

$$\begin{array}{lll}
(\emptyset, S) & \longrightarrow_{D,\mathcal{C}} & (\emptyset, \mathbf{New}^{q_1}\ (G\star)) \\
 & \longrightarrow_{D,\mathcal{C}} & (\{y \mapsto q_1\}, G\ \star\ y) \\
 & \longrightarrow_{D,\mathcal{C}} & (\{y \mapsto q_1\}, \mathbf{If}*\ (\mathbf{Acc}_c\ y\ \star)\ (\mathbf{Acc}_r\ y\ (G\ \star\ y))) \\
 & \longrightarrow_{D,\mathcal{C}} & (\{y \mapsto q_1\}, \mathbf{Acc}_r\ y\ (G\ \star\ y)) \\
 & \longrightarrow_{D,\mathcal{C}} & (\{y \mapsto q_1\}, G\ \star\ y) \\
 & \longrightarrow_{D,\mathcal{C}} & (\{y \mapsto q_1\}, \mathbf{If}*\ (\mathbf{Acc}_c\ y\ \star)\ (\mathbf{Acc}_r\ y\ (G\ \star\ y))) \\
 & \longrightarrow_{D,\mathcal{C}} & (\{y \mapsto q_1\}, \mathbf{Acc}_c\ y\ \star) \\
 & \longrightarrow_{D,\mathcal{C}} & (\{y \mapsto q_2\}, \star)
\end{array}$$

We can now formally define the resource usage verification problem.

Definition 5.3 (resource usage verification problem)**.** A program $(D, S, \mathcal{C})$ is *resource-safe* if (i) $(\emptyset, S) \not\longrightarrow^*_{D,\mathcal{C}}$ **Error**, and (ii) if $(\emptyset, S) \longrightarrow^*_{D,\mathcal{C}} (\rho, \star)$ then $\rho(x) \in F_{\mathcal{C}}$ for every $x \in dom(\rho)$. The *resource usage verification* is the problem of checking whether a program is resource-safe.

Example 5.4. The program given in Example 5.1 is resource-safe. The program obtained by replacing the body of G (i.e. $\mathbf{If}*\ (\mathbf{Acc}_c\ x\ k)\ (\mathbf{Acc}_r\ x\ (G\ k\ x))$) with $\mathbf{Acc}_r\ x\ (G\,k\,x)$ is also resource-safe; it does not terminate, so that it satisfies condition (ii) of Definition 5.3 vacuously. The program D' obtained by replacing the definition of G with:

$$G\ k\ x = \mathbf{If}*\ k\ (\mathbf{Acc}_r\ x\ (G\ k\ x))$$

is not resource-safe, as $(\emptyset, S) \longrightarrow^*_{D',\mathcal{C}} (\{y \mapsto q_1\}, \star)$ and $q_1 \notin F_{\mathcal{C}}$.

We show below that the resource usage verification is $(n-1)$-EXPTIME complete for $n \geq 3$, where n is the largest order of types of terms in the source program. Here, the order of a type is defined by:

$$\mathit{order}(\mathbf{unit}) = 0 \qquad \mathit{order}(\mathbf{R}) = 1 \qquad \mathit{order}(\kappa_1 \to \kappa_2) = \mathit{max}(\mathit{order}(\kappa_1)+1, \mathit{order}(\kappa_2))$$

Note that 3 is the lowest order of a closed program that creates a resource, since $\mathbf{New}^L$ has order 3.

The lower-bound can be shown by reduction of the reachability problem for a recursion scheme to the resource usage verification problem: Given a recursion scheme $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, let $(D, S, \mathcal{C})$ be the program given by:

$$\begin{array}{l}
D = \{F\,\widetilde{x} = \mathit{g2p}(t) \mid F\,\widetilde{x} \to t \in \mathcal{R}\} \cup \{\mathit{Fail}\ x = \mathbf{Acc}_{\mathtt{fail}}\ x \star\}\\
\mathit{g2p}(F\,t_1\ \cdots\ t_m) = F\ \mathit{g2p}(t_1)\ \cdots\ \mathit{g2p}(t_m)\\
\mathit{g2p}(\mathtt{e}) = \nu^q\ \mathit{Fail}\\
\mathit{g2p}(a\,t_1\ \cdots\ t_m) = \mathbf{If}{*}\ \mathit{g2p}(t_1)\ (\cdots(\mathbf{If}{*}\ \mathit{g2p}(t_{m-1})\ \mathit{g2p}(t_m)))\ (a \neq \mathtt{e})\\
\mathcal{C} = (\{q\}, \{\mathtt{fail}\}, \emptyset, q, \{q\})
\end{array}$$

Then, the value tree of $\mathcal{G}$ contains $\mathtt{e}$ if and only if the program $(D, S, \mathcal{C})$ is resource-safe. Since resource primitives occur only in the encoding of $\mathtt{e}$, the order of the program is the maximum of 3 and the order of the recursion scheme.

To show the upper-bound, we transform a program $(D, S, \mathcal{C})$ into a recursion scheme $\mathcal{G}_{(D,S,\mathcal{C})}$, which generates a tree representing all possible (resource-wise) access sequences of the program [10], and a disjunctive APT $\mathcal{D}_{(D,S,\mathcal{C})}$, which accepts trees containing an invalid resource access sequence, so that $(D, S, \mathcal{C})$ is resource-safe if, and only if, $\mathcal{D}_{(D,S,\mathcal{C})}$ accepts the value tree of $\mathcal{G}_{(D,S,\mathcal{C})}$.

The recursion scheme $\mathcal{G}_{(D,S,\mathcal{C})} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ is given by:

$$\begin{array}{l}
\Sigma = \{a \mapsto 1 \mid a \in A\} \cup \{\nu^q \mapsto 2 \mid q \in Q_{\mathcal{C}}\} \cup \{\star \mapsto 0, \mathtt{i} \mapsto 1, \mathtt{k} \mapsto 1, \mathtt{br} \mapsto 2\}\\
\mathcal{N} = (\text{the set of function symbols in } D)\\
\qquad \cup\{\mathbf{If}{*} \mapsto \mathtt{o} \to \mathtt{o} \to \mathtt{o}\} \cup \{\mathbf{Acc}_a \mapsto (\mathtt{o} \to \mathtt{o}) \to \mathtt{o} \to \mathtt{o} \mid a \in A\}\\
\qquad \cup\{\mathbf{New}^q\ \mapsto ((\mathtt{o} \to \mathtt{o}) \to \mathtt{o}) \to \mathtt{o} \mid q \in Q_{\mathcal{C}}\}\\
\mathcal{R} = \{F\,\widetilde{x} \to e \mid F\,\widetilde{x} = e \in D\}\\
\qquad \cup\{\mathbf{If}{*}\ x\ y \to \mathtt{br}\,x\,y, \quad \mathbf{Acc}_a\ x\ k \to x\,(a\,k), \quad \mathbf{New}^q\ k \to \nu^q(k\,\mathtt{i})\,(k\,\mathtt{k})\}
\end{array}$$

Here, A is the set of the names of access primitives that occur in D.

Here, the encoding is slightly different from the one presented in [10]. The terminal symbol $\mathtt{br}$ represents a non-deterministic choice. In the rule for $\mathbf{New}^L$, a fresh resource is instantiated to either $\mathtt{i}$ or $\mathtt{k}$ of arity 1. This is a trick used to extract resource-wise access sequences, by tracking or ignoring the new resource in a non-deterministic manner. In the first-branch, the resource is instantiated to $\mathtt{i}$, so that all the accesses to the resource are kept track of. In the second branch, the resource is instantiated to $\mathtt{k}$, so that all the accesses to the resource should be ignored. The above transformation preserves types, except that **unit** and **R** are replaced by $\mathtt{o}$ and $\mathtt{o} \to \mathtt{o}$ respectively.

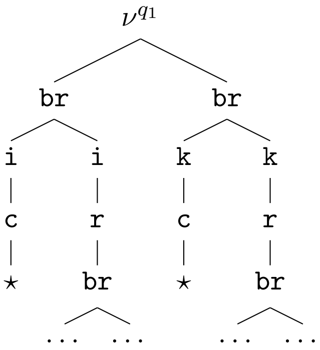

Figure 2: The tree generated by the recursion scheme of Example 5.5

Example 5.5. The program in Example 5.1 is transformed into the recursion scheme consisting of the following rules:

$$\begin{array}{rcl} S & \rightarrow & \mathbf{New}^{q_1}\ (G\ \star) \\ G\ k\ x & \rightarrow & \mathbf{If}{*}\ (\mathbf{Acc}_c\ x\ k)\ (\mathbf{Acc}_r\ x\ (G\ k\ x)) \\ \mathbf{If}{*}\ x\ y & \rightarrow & \mathtt{br}\, x\, y \\ \mathbf{Acc}_a\ x\ k & \rightarrow & x\,(a\,k) \\ \mathbf{New}^q\ k & \rightarrow & \nu^q(k\,\mathtt{i})\,(k\,\mathtt{k}) \end{array}$$

Figure 5 shows the value tree of the recursion scheme. The root node represents creation of a new resource (whose initial state is q). The nodes labeled by `c` or `r` express resource accesses. The left and right children are the same, except that each resource access is prefixed by `i` in the left child, while it is prefixed by `k` in the right child.

Example 5.6. Consider the following program, which creates and accesses two resources:

$$\begin{array}{l} S = \mathbf{New}^{q_1}\ F \\ F\ x = \mathbf{New}^{q_1}\ (G\ \star\ x) \\ G\ k\ x\ y = \mathbf{If}{*}\ (\mathbf{Acc}_c\ x\ (\mathbf{Acc}_c\ y\ k))\ (\mathbf{Acc}_r\ x\ (\mathbf{Acc}_r\ y\ (G\ k\ x\ y))) \end{array}$$

It is transformed into the recursion scheme consisting of the following rules:

$$\begin{array}{rcl} S & \rightarrow & \mathbf{New}^{q_1}\ F \\ F\ x & \rightarrow & \mathbf{New}^{q_1}\ (G\ \star\ x) \\ G\ k\ x\ y & \rightarrow & \mathbf{If}{*}\ (\mathbf{Acc}_c\ x\ (\mathbf{Acc}_c\ y\ k))\ (\mathbf{Acc}_r\ x\ (\mathbf{Acc}_r\ y\ (G\ k\ x\ y))) \\ \mathbf{If}{*}\ x\ y & \rightarrow & \mathtt{br}\, x\, y \\ \mathbf{Acc}_a\ x\ k & \rightarrow & x\,(a\,k) \\ \mathbf{New}^{q_1}\ k & \rightarrow & \nu^{q_1}(k\,\mathtt{i})\,(k\,\mathtt{k}) \end{array}$$

Figure 5 shows the value tree of the recursion scheme. Of the four subtrees whose roots are labeled by `br`, the leftmost subtree represents accesses to both resources x and y; in other words, all the accesses to x and y are prefixed by `i`. In the second subtree, only the accesses to x are prefixed by `i`. In the third subtree, only the accesses to y are prefixed by `i`, while in the rightmost subtree, no accesses are prefixed by `i`.

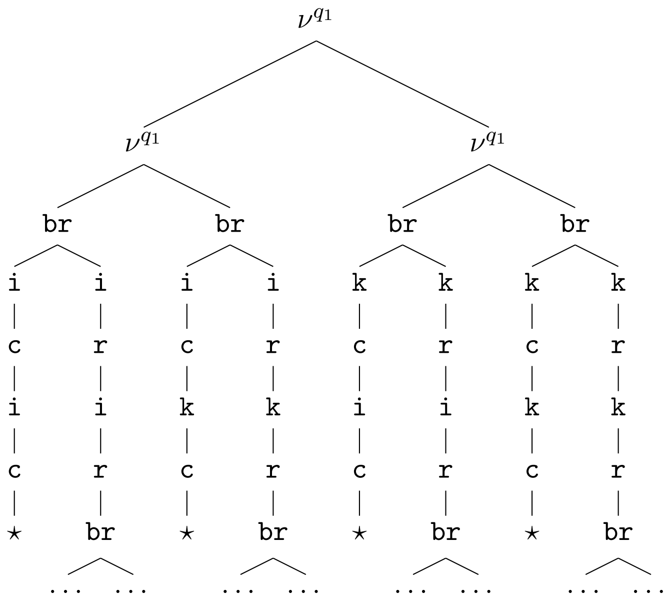

Figure 3: The tree generated by the recursion scheme of Example 5.6

The disjunctive APT $\mathcal{D}_{(D,S,\mathcal{C})} = (\Sigma, Q, \delta, q_I, \Omega)$, which accepts trees having a path corresponding to an invalid access sequence, is given by:

$$Q = Q_{\mathcal{C}} \cup \{\overline{q} \mid q \in Q_{\mathcal{C}}\} \cup \{q_I\}$$

$$\delta(q_I, a) = \begin{cases} (1, q_I) \vee (2, q_I) & \text{if } a = \mathtt{br} \\ (1, q) \vee (2, q_I) & \text{if } a = \nu^q \\ \mathsf{f} & \text{if } a = \star \\ (1, q_I) & \text{otherwise} \end{cases}$$

$$\delta(q, a)(\text{where } q \in Q_{\mathcal{C}}) = \begin{cases} (1, q) \vee (2, q) & \text{if } a = \mathtt{br} \\ (1, q) & \text{if } a = \mathtt{i} \\ (1, \overline{q}) & \text{if } a = \mathtt{k} \\ (2, q) & \text{if } a = \nu^q \\ \mathsf{f} & \text{if } a = \star \text{ and } q \in F_{\mathcal{C}} \\ \mathsf{t} & \text{if } a = \star \text{ and } q \notin F_{\mathcal{C}} \\ (1, q') & \text{if } a \in A \text{ and } \delta_{\mathcal{C}}(q, a) = q' \\ \mathsf{t} & \text{if } a \in A \text{ and } \delta_{\mathcal{C}}(q, a) \text{ is undefined} \end{cases}$$

$$\delta(\overline{q}, a)(\text{where } q \in Q_{\mathcal{C}}) = (1, q)$$

$$\Omega(q) = 1 \text{ for every } q \in Q$$

Σ is the same as that of $\mathcal{G}_{(D,S,\mathcal{C})}$.

The APT reads the root of a tree with state q_I, and traverses a tree to find a path corresponding to an invalid resource access sequence. After reading ν^q in state q_I, the APT either (i) chooses the left branch and changes its state to q, the initial state of the new resource, tracking accesses to the resource afterwards; or (ii) chooses the right branch, ignoring accesses to the new resource. In the mode to track resource accesses (i.e., in state $q \in Q$), the APT changes its state according to resource accesses, except: (i) upon reading k, it skips the next symbol, which represents an access to a resource not being tracked, (ii) upon reading ν^q, it only reads the right branch, ignoring the resource created by this ν^q (as it is already keeping track of another resource), (iii) upon reading $a \in A$ such that $\mathcal{C}(q,a)$ is undefined or reading $\star$ when $q \notin F_{\mathcal{C}}$, it terminates successfully (as an invalid access sequence has been found), and (iv) upon reading $\star$ at state $q \in F_{\mathcal{C}}$, it aborts (as a path being read was actually a valid access sequence). The priority function maps every state to 1, so that no infinite run (that corresponds to an infinite execution sequence of the program without any invalid resource access) is considered an accepting run.

From the construction above, we have:

Theorem 5.7. *$(D, S, \mathcal{C})$ is resource-safe if, and only if, the value tree of $\mathcal{G}_{(D,S,\mathcal{C})}$ is not accepted by $\mathcal{D}_{(D,S,\mathcal{C})}$.*

The proof is similar to the corresponding theorem in [10], hence omitted.[7]

Note that the order of $\mathcal{G}_{(D,S,\mathcal{C})}$ is the same as that of D. Thus, as a corollary of the above theorem and Theorem 4.3, we obtain that the resource usage verification is $(n-1)$-EXPTIME.

6. Related Work

Our analysis of the lower bound is based on Engelfriet's earlier work on the complexity of the iterated pushdown automata word acceptance and emptiness problems, and the results of Knapik et al. on the relationship between higher-order PDA and safe recursion schemes.

The model checking of recursion schemes for the class of trivial APT has been studied by Aehlig [1] (under the name "trivial automata"). He gave a model checking algorithm, but did not discuss its complexity. For the same class, Kobayashi [10] showed that the complexity is linear in the size of recursion schemes, if the types and automata are fixed. For the full modal μ-calculus, Kobayashi and Ong [11] have shown that the complexity is n-EXPTIME in the largest arity of symbols in the recursion scheme, the number of states of the APT, and the largest priory, but polynomial in the number of the rules of the recursion scheme.

Our encoding of the word acceptance problem of an order-n alternating PDA into the model checking problem of an order-n tree-generating PDA (the construction of $\mathcal{M}_{\mathcal{A},w}$ in Section 3) is similar to Cachat and Walukiewicz's encoding of the word acceptance problem into the reachability game on a higher-order pushdown system [2]. In fact, the tree generated by $\mathcal{M}_{\mathcal{A},w}$ seems to correspond to the unravelling of the game graph of the higher-order pushdown system (where the nodes labelled by `E` are Player's positions, and those labelled by `A` are Opponent's positions). Thus, n-EXPTIME-hardness of model checking for trivial APT (in the size of the recursion scheme) would follow also from n-EXPTIME hardness of the reachability game on higher-order pushdown systems [2].

7. Conclusion

We have considered two subclasses of APT, and shown that the model checking of an order-n recursion scheme is n-EXPTIME complete for trivial APT, and $(n-1)$-EXPTIME complete for disjunctive APT, both in the size of the recursion scheme and in the size of the

[7] As mentioned above, the encoding presented in this article is slightly different from the one in [10], but the proofs are similar: they are tedious but rather straightforward.

APT. As an application, we showed that the resource usage verification problem is $(n-1)$-EXPTIME complete. The lower bound for the finiteness problem (recall Section 4.3) is left as an open problem.

Acknowledgments. We would like to thank the anonymous reviewers (of our ICALP 2009 paper) for useful comments. This work was partially supported by Kakenhi 20240001 and EPSRC EP/F036361.

References

[1] K. Aehlig. A finite semantics of simply-typed lambda terms for infinite runs of automata. *Logical Methods in Computer Science*, 3(3), 2007.

[2] T. Cachat and I. Walukiewicz. The complexity of games on higher order pushdown automata. *CoRR*, abs/0705.0262, 2007.

[3] J. Engelfriet. Interated stack automata and complexity classes. *Information and Computation*, 95:21–75, 1991.

[4] J. Engelfriet. Iterated stack automata and complexity classes. *Information and Computation*, 95(1):21–75, 1991.

[5] M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *Proc. IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, 2008.

[6] A. Igarashi and N. Kobayashi. Resource usage analysis. *ACM Transactions on Programming Languages and Systems*, 27(2):264–313, 2005.

[7] D. Janin and I. Walukiewicz. Automata for the modal mu-calculus and related results. In *Proc. MFCS*, pages 552–562, 1995.

[8] M. Jurdziński. Small progress measures for solving parity games. In *Proc. STACS*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301, 2000.

[9] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer-Verlag, 2002.

[10] N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pages 416–428, 2009.

[11] N. Kobayashi and C.-H. L. Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of LICS 2009*, pages 179–188. IEEE Computer Society Press, 2009.

[12] C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS 2006*, pages 81–90. IEEE Computer Society Press, 2006.

[13] S. Schewe. Solving parity games in big steps. In *Proceedings of FSTTCS 2007*, volume 4855 of *Lecture Notes in Computer Science*, pages 449–460. Springer-Verlag, 2007.

[14] W. Thomas. Languages, automata and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3. Springer-Verlag, 1997.

[15] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:137, 1994.

[16] I. Walukiewicz. Automata and logic. Notes for EFF Summer School, 2002.

Appendix A. Characterizing trivial APT and disjunctive APT as modal mu-calculus fragments

Preliminaries. Let Σ be a ranked alphabet, and m be the largest arity of the symbols in Σ. For technical convenience, we use a slightly different definition of *alternating parity tree automaton* (APT) over Σ-labelled trees, which is a 5-tuple

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

where $\delta : Q \times \Sigma \longrightarrow \mathcal{P}^{ne}(\{\epsilon, 1, 2, \cdots, m\} \times Q)$ and the image of δ consists of non-empty sets. The map λ partitions Q into Q_E (existential or Éloïse's states) and Q_A (universal or Abelard's states). We assume that Q_E contains distinguished accept and reject states ($\top$ and $\bot$ respectively). An APT $\mathcal{A}$ is said to be *trivial* if the only priority is 0; it is said to be *disjunctive* if $\delta(q, f)$ is a singleton[8] set, for every $q \in Q_A$ and $f \in \Sigma$.

In the following we adapt Walukiewicz' translations [16] to our setting of ranked trees of varying branching degrees.

From Logic to Automata. Let χ be a closed modal mu-calculus formula in normal form (i.e. the atomic propositions are P_f with $f \in \Sigma$, and bound variables are pairwise distinct). Let $\mathit{SubF}(\chi)$ be the set of subformulas of χ. For each variable Z that occurs in χ, we write β_Z for the (unique) subformula such that $\sigma Z.\beta_Z \in \mathit{SubF}(\chi)$ where σ is either μ or ν. We define an APT $\mathcal{A}_\chi = \langle\, Q, \lambda, \chi, \delta, \Omega\rangle$ where

- Q_A consists of subformulas of χ that are conjunctions; Q_E consists of $\bot$, $\top$ and subformulas that are not conjunctions.
- The transition function δ is given by the following table:

$\phi \in \mathit{SubF}(\chi)$	$\delta(\phi, f)$
P_g	$\{(\epsilon, [f = g])\}$
$\phi_1 \wedge \phi_2,\ \phi_1 \vee \phi_2$	$\{(\epsilon, \phi_1), (\epsilon, \phi_2)\}$
$\langle i\rangle\psi$ where $i \leq \mathit{arity}(f)$	$\{(i, \psi)\}$
$\langle i\rangle\psi$ where $i > \mathit{arity}(f)$	$\{(\epsilon, \bot)\}$
Z	$\{(\epsilon, \beta_Z)\}$
$\sigma Z.\beta_Z$	$\{(\epsilon, Z)\}$

  where $[f = g] := \begin{cases} \top & \text{if } f = g \\ \bot & \text{otherwise.} \end{cases}$
- The priority map Ω is defined as follows:

$$\Omega(\phi) \;:=\; \begin{cases} 2(d - \mathit{altdep}_\chi(Z)) & \text{if } \phi = Z \text{ is a } \nu\text{-variable} \\ 2(d - \mathit{altdep}_\chi(Z)) + 1 & \text{if } \phi = Z \text{ is a } \mu\text{-variable} \\ 0 & \text{otherwise} \end{cases}$$

  where $\mathit{altdep}_\chi(Z)$ is the alternation depth of Z in χ, and d is the largest alternation depth of variables of χ.

Let χ be a closed formula of the modal mu-calculus and t a Σ-labelled tree t. Walukiewicz *op. cit.* has shown that t satisfies χ (at the root) if, and only if, t is accepted by the APT $\mathcal{A}_\chi$.

[8] If $q \in Q_A$ and $\delta(q, f) = \{(x, q'), (\epsilon, \bot)\}$ (respectively $\{(x, q'), (\epsilon, \top)\}$), then we obtain an equivalent APT by redefining $\delta(q, f) := \{(\epsilon, \bot)\}$ (respectively $\{(x, q')\}$) instead.

Proposition A.1. (i) If χ is an $\mathcal{S}$-formula, then $\mathcal{A}_\chi$ is a trivial APT.
(ii) If χ is a $\mathcal{D}$-formula, then $\mathcal{A}_\chi$ is a disjunctive APT.

Proof. (i): It follows from the definition of Ω that $\mathcal{A}_\chi$ has only one priority which is 0. (ii): For every $\phi \in Q_A$ (by definition $\phi = P_g \wedge \psi$, say), $\delta(\phi, f) = \{(\epsilon, [f = g]), (\epsilon, \psi)\}$ as desired – see footnote 8. □

From Automata to Logic. Fix an APT $\mathcal{A} = \langle Q, \lambda, q_I, \delta, \Omega \rangle$ where $Q = \{q_1, \cdots, q_n\}$. Suppose the ordering $q_1, \cdots, q_n$ satisfies $\Omega(q_i) \leq \Omega(q_j)$ for every $i < j$. Consider the following n-tuple of modal mu-calculus formulas—call it $\chi_\mathcal{A}$—simultaneously defined by least and greatest fixpoints:

$$\sigma_1 \begin{pmatrix} Z_{11} \\ \vdots \\ Z_{1n} \end{pmatrix} . \cdots . \sigma_n \begin{pmatrix} Z_{n1} \\ \vdots \\ Z_{nn} \end{pmatrix} . \begin{pmatrix} \chi_1 \\ \vdots \\ \chi_n \end{pmatrix}$$

where $\sigma_i := \mu$ if $\Omega(q_i)$ is odd, and ν otherwise. For each $1 \leq i \leq n$

$$\chi_i \ := \ \bigvee_{f \in \Sigma} (P_f \wedge \ulcorner \delta(q_i, f) \urcorner).$$

Suppose $\delta(q_i, f) = \{(d_1, q_{i_1}), \cdots, (d_r, q_{i_r})\}$. We define

$$\ulcorner \delta(q_i, f) \urcorner \ := \ \begin{cases} \bigwedge_{j=1}^{r} \ulcorner (d_j, q_{i_j}) \urcorner & \text{if } \lambda(q_i) = A \\ \bigvee_{j=1}^{r} \ulcorner (d_j, q_{i_j}) \urcorner & \text{if } \lambda(q_i) = E \end{cases}$$

with

$$\ulcorner (d, q_j) \urcorner \ := \ \begin{cases} \langle d \rangle Z_{jj} & \text{if } 1 \leq d \leq \mathit{arity}(f) \\ Z_{jj} & \text{otherwise } d = \epsilon \end{cases}$$

Write $\pi_i(\chi_\mathcal{A})$ to be a modal mu-calculus formula (semantically) equivalent to $\chi_\mathcal{A}$ projected onto the i-th component (which is well-defined by an application of the Bekic Principle).

Let $\mathcal{A}$ be an APT and t a Σ-labelled tree. Walukiewicz *op. cit.* has shown that t is accepted by $\mathcal{A}$ if, and only if, it satisfies $\pi_I(\chi_\mathcal{A})$ at the root.

Proposition A.2. (i) If $\mathcal{A}$ is a trivial APT, then $\pi_I(\chi_\mathcal{A})$ is a $\mathcal{S}$-formula.
(ii) If $\mathcal{A}$ is a disjunctive APT, then $\pi_I(\chi_\mathcal{A})$ is a $\mathcal{D}$-formula.

Proof. (i): If $\mathcal{A}$ has only one priority 0, then it follows from the definition that $\chi_\mathcal{A}$ is constructed using only ν-fixpoint operator. (ii) Since $\delta(q_i, f)$ is a singleton set whenever $q_i \in Q_A$, it follows that every conjunction subformula of $\chi_\mathcal{A}$ is of the form $P_f \wedge \phi$. □

□

Appendix B. Alternative Proof of $(n-1)$-EXPTIME Upper-Bound for Disjunctive APT

We sketch an alternative proof of Lemma 4.3, using Ong's *variable profiles* [12].

In order to appreciate the proof sketched below, some knowledge of the workings of a traversal simulating APT is required. In particular it is necessary to know about *variable profiles* and how they are employed.

Since $\mathcal{B}$ is disjunctive, it has an accepting run-tree on $[\![G]\!]$ just in case it has an accepting run-tree that does not branch (i.e. each node of the run-tree has at most one child). It follows that $\mathcal{B}$ has an accepting traversal tree if and only if it has an accepting traversal tree that does not branch.

The key observation is that the traversal-simulating APT $\mathcal{C}$ thus need only 'guess' one exit point when it reaches a node labelled by a variable of order one, even if its type has arity greater than one. It follows that we can simplify the definition of variable profiles. A profile of a ground-type variable has the shape $(x, q, m, \emptyset)$ where q is a state and m a colour, which is the same as the general case. However a profile of a variable ϕ of a first-order type $\underbrace{o \to \cdots \to o}_{k} \to o$ now has the shape (ϕ, q, m, c) where c is either empty or a *singleton* set consisting of a profile of a ground-type variable, as opposed to a set of such profiles. The profiles of variables of order two or higher are defined as in the general case. Thus the number of variable profiles of a given order (at least one) is reduced by one level of exponentiation compared to the general case. Now viewing $\mathbf{VP}(A)$ as denoting the set of variable profiles of type A (of order at least one) restricted to containing either empty or singleton interfaces:

$$\sum_{A \text{ order } i \text{ type}} |\mathbf{VP}(A)| \;=\; O(exp_{i-1}(|Gr_G| \times |Q| \times p))$$

where Q is the state space of $\mathcal{B}$, p is the number of priorities, and Gr_G is the (finite) graph that unravels to the computation tree $\lambda(G)$. The number of nodes in the parity game induced by the traversal-simulating APT $\mathcal{C}$ and the computation tree $\lambda(G)$ will thus also have bound $O(exp_{n-1}(|Gr_G| \times |Q| \times p))$ and using Jurdziński's algorithm [8] we have it that the acceptance parity game can be solved in time $O(exp_{n-1}(|Gr_G| \times |Q| \times p))$. The problem thus lies in $(n-1)$-EXPTIME.