Pushdown Specifications

Orna Kupferman[1⋆] and Nir Piterman[2⋆⋆] and Moshe Y. Vardi[3⋆⋆⋆]

[1] Hebrew University, School of Engineering and Computer Science, Jerusalem 91904, Israel
Email: orna@cs.huji.ac.il, URL: http://www.cs.huji.ac.il/~orna
[2] Weizmann Institute of Science, Department of Computer Science, Rehovot 76100, Israel
Email: nirp@wisdom.weizmann.ac.il, URL: http://www.wisdom.weizmann.ac.il/~nirp
[3] Rice University, Department of Computer Science, Houston, TX 77251-1892, U.S.A.
Email: vardi@cs.rice.edu, URL: http://www.cs.rice.edu/~vardi

Abstract. Traditionally, model checking is applied to finite-state systems and regular specifications. While researchers have successfully extended the applicability of model checking to infinite-state systems, almost all existing work still consider regular specification formalisms. There are, however, many interesting non-regular properties one would like to model check.
In this paper we study model checking of *pushdown specifications*. Our specification formalism is nondeterministic pushdown parity tree automata (PD-NPT). We show that the model-checking problem for regular systems and PD-NPT specifications can be solved in time exponential in the system and the specification. Our model-checking algorithm involves a new solution to the nonemptiness problem of nondeterministic pushdown tree automata, where we improve the best known upper bound from a triple-exponential to a single exponential. We also consider the model-checking problem for context-free systems and PD-NPT specifications and show that it is undecidable.

1 Introduction

One of the most significant developments in the area of formal design verification is the discovery of algorithmic methods for verifying on-going behaviors of reactive systems [QS81,LP85,CES86,VW86]. In *model-checking*, we verify the correctness of a system with respect to a desired behavior by checking whether a mathematical model of the system satisfies a formal specification of this behavior (for a survey, see [CGP99]). Traditionally, model checking is applied to *finite-state* systems, typically modeled by labeled state-transition graphs, and to behaviors that are formally specified as *temporal-logic* formulas or *automata on infinite objects*. Symbolic methods that enable model-checking of very large state spaces, and the great ease of use of fully algorithmic methods, led to industrial acceptance of model-checking [BLM01,CFF+01].

In recent years, researchers have tried to extend the applicability of model-checking to infinite-state systems. An active field of research is model-checking of *infinite-state*

⋆ Supported in part by BSF grant 9800096.
⋆⋆ Supported in part by the European Commission (FET project ADVANCE, contract No IST-1999-29082). This work was carried out at the John von Neumann Minerva Center.
⋆⋆⋆ Supported in part by NSF grants CCR-9988322, IIS-9908435, IIS-9978135, and EIA-0086264, by BSF grant 9800096, and by a grant from the Intel Corporation.

sequential systems. These are systems in which each state carries a finite, but unbounded, amount of information, e.g., a pushdown store. The origin of this research is the result of Müller and Schupp that the monadic second-order theory of *context-free graphs* is decidable [MS85]. As the complexity involved in that decidability result is nonelementary, researchers sought decidability results of elementary complexity. Various algorithms for simpler logics and more general systems have been proposed. The most powerful result so far is an exponential-time algorithm by Burkart for model checking formulas of the μ-calculus with respect to prefix-recognizable graphs [Bur97b].

An orthogonal line of research considers the applicability of model checking to *infinite-state specifications*. Almost all existing work on model checking considers specification formalisms that define *regular* sets of words, trees, or graphs: formulas of LTL, μ-calculus, and even monadic-second order logic can all be translated to automata [Büc62,Rab69,EJ91], and in fact many model-checking algorithms (for both finite-state and infinite-state systems) first translate the given specification into an automaton and reason about the structure of this automaton (cf., [VW86,BEM97,KV00]). Sometimes, however, the desired behavior is non-regular and cannot be specified by a finite-state automaton. Consider for example the property "p is inevitable", for a proposition p. That is, in every computation of the system, p eventually holds. Clearly, this property is regular and is expressible as $\forall\Diamond p$ in both CTL [CES86] and LTL [Pnu77]. On the other hand, the property "p is uniformly inevitable", namely, there is some time i such that in every computation of the system, p holds at time i, is not expressible by a finite automaton on infinite trees [Eme87], and hence, it is non-regular. As another example, consider a system that handles requests and acknowledgments, and the property "every acknowledgment is preceded by some request". Again, this property is regular and is expressible in LTL as $(\neg ack)\mathcal{W}req$. On the other hand, consider the property of "no redundant acknowledgments", namely the number of acknowledgments does not exceed the number of requests. The technique of [Eme87] can be used in order to show that the property is non-regular. More examples to useful non-regular properties are given in [SCFG84], where the specification of unbounded message buffers is considered.

The need to specify non-regular behaviors led Bouajjani et al. [BER94,BEH95] to consider logics that are a combination of CTL and LTL with Presburger Arithmetic. The logics, called PCTL and PLTL, use variables that range over natural numbers. The variables are bound to the occurrences of state formulas and comparison between such variables is allowed. The non-regular properties discussed above can be specified in PCTL and PLTL. For example, we can specify uniform inevitability in PCTL as $\widetilde{\exists} i \,.\, \forall[x : \mathbf{true}](x = i \rightarrow p)$, where the $\widetilde{\exists}$ quantifier quantifies over natural numbers, the $\forall$ quantifier quantifies over computations of the system, and the combinator $[x : \mathbf{true}]$ binds the variable x to count the number of occurrences of the state formula $\mathbf{true}$. Bouajjani et al. consider the model-checking problem for the logics PCTL and PLTL over finite-state (regular) systems and over infinite-state (non-regular) systems. The logics turned out to be too strong: the model-checking of both PCTL and PLTL over finite-state systems is undecidable. They proceed to restrict the logics to fragments for which model-checking over finite-state systems and context-free systems is decidable. The properties "p is uniformly inevitable" and "no redundant acknowledgments" are both expressible in the restricted (decidable) fragments of PCTL and PLTL.

Uniform inevitability is clearly expressible by a *nondeterministic pushdown tree automaton*. Pushdown tree automata are finite-state automata augmented by a pushdown store. Like a nondeterministic finite-state tree automaton, a nondeterministic pushdown tree automaton starts reading a tree from the root. At each node of the tree, the pushdown automaton consults the transition relation and splits into independent copies of itself to each of the node's successors. Each copy has an independent pushdown store that diverges from the pushdown store of the parent. We then check what happens along every branch of the run tree and determine acceptance. In order to express uniform inevitability, the automaton guesses the time i, pushes i elements into the pushdown store, and, along every computation, pops one element with every move of the system. When the pushdown store becomes empty, the automaton requires p to hold. Similarly, in order to express "no redundant acknowledgments", a nondeterministic pushdown tree automaton can push an element into the pushdown store whenever the system sends a request, pop one element with every acknowledgment, and reject the tree when an acknowledgment is issued when the pushdown store is empty. In [PI95], Peng and Iyer study more properties that are non-regular and propose to use nondeterministic pushdown tree automata as a strong specification formalism. The model studied by [PI95] is *empty store*: a run of the automaton is accepting if the automaton's pushdown store gets empty infinitely often along every branch in the run tree.

In this paper we study the model-checking problem for specifications given by nondeterministic pushdown tree automata. We consider both finite-state (regular) and infinite-state (non-regular) systems. We show that for finite-state systems, the model-checking problem is solvable in time exponential in both the system and the specification, even for nondeterministic pushdown parity tree automata – a model that is much stronger than the one studied in [PI95]. On the other hand, the model-checking problem for context-free systems is undecidable – already for a weak type of pushdown tree automata. Note that by having tree automata as our specification formalism, we follow here the branching-time paradigm, where the specification describes allowed computation trees and a system is correct if its computation tree is allowed [CES86]. In Remark 2, we discuss the undecidability of the linear-time paradigm, and the reasons that make the (seemingly more general) branching-time framework decidable.

In order to solve the model-checking problem for nondeterministic pushdown tree automata and finite-state systems, we use the automata theoretic approach to branching time model checking [KVW00]. In [KVW00], model checking is reduced to the *emptiness* problem for nondeterministic finite tree automata, here we reduce the model checking problem to the emptiness problem for nondeterministic pushdown tree automata. The first to show that this emptiness problem is decidable were Harel and Raz [HR94]. The automata considered by Harel and Raz use the Büchi acceptance condition, where some states are designated as accepting states and a run is accepting if it visits the accepting states infinitely often along every branch in the run tree. It is shown in [HR94] that the problem can be solved in triple-exponential time. Recall that Peng and Iyer [PI95] consider a simpler acceptance condition, where a run is accepting if the automaton's pushdown store gets empty infinitely often along every branch in the run tree. For this acceptance condition, it is shown in [PI95] that the nonemptiness problem can be solved in exponential time. Nevertheless, empty store pushdown automata are strictly weaker than nondeterministic Büchi pushdown tree automata [PI95] and the algorithm in [PI95] cannot be extended to handle the Büchi acceptance condition.

The main result of this paper is an exponential algorithm for the emptiness problem of nondeterministic *parity* pushdown tree automata. Thus, apart from improving the known triple-exponential upper bound to a single exponential, we handle a more general acceptance condition. Our algorithm is based on a reduction of the emptiness problem to the membership problem for *two-way alternating parity tree automata* with no pushdown store. We note that our technique can be applied also to specifications given by *alternating* pushdown parity tree automata. Indeed, the automata-theoretic approach to branching-time model checking involves some type of a product between the system and the specification automaton, making alternation as easy as nondeterminism [KVW00]. In Remark 3, we discuss this point further, and also show that, unlike the case of regular automata, alternating pushdown automata are strictly more expressive than nondeterministic pushdown tree automata.

Once one realizes that the difficulties in handling the pushdown store of the tree automaton are similar to the difficulties in handling the pushdown store of infinite-state sequential systems, it is possible to solve the nonemptiness problem for pushdown automata with various methods that have been suggested for the latter. In particular, it is possible to reduce the nonemptiness problem for nondeterministic pushdown parity tree automata to the μ-calculus model-checking problem for pushdown systems [Wal01]. The solution we suggest here is the first to suggest the application of methods developed for reasoning about infinite-state sequential systems to the solution of automata-theoretic problems for pushdown automata. In particular, we believe that methods based on two-way alternating tree automata [KV00,KPV02] are particularly appropriate for this task, as the solution stays in the clean framework of automata.

Finally, in order to show the undecidability result, we reduce the problem of deciding whether a two-counter machine accepts the empty tape to the model-checking problem of a context-free system with respect to a nondeterministic pushdown tree automaton. Intuitively, the pushdown store of the system can simulate one counter, and the pushdown store of the specification can simulate the second counter.

The study of pushdown specifications completes the picture described in the table below regarding model checking of regular and context-free systems with respect to regular and pushdown specifications. When both the system and the specification are regular, the setting is that of traditional model checking [CGP99]. When only one parameter has a pushdown store, the problem is still decidable. Yet, when both the system and the specification have a pushdown store, model checking becomes undecidable. The complexities in the table refer to the case where the specification is given by a nondeterministic or an alternating parity tree automaton of size n and index k. The size of the system is m.

	Regular Specifications	Pushdown Specifications
Regular Systems	decidable; $O((nm)^k)$ [EJS93]	decidable; $exp(mnk)$ [Theorem 5]
Pushdown Systems	decidable; $exp(mnk)$ [KV00]	undecidable [Theorem 6]

2 Definitions

Trees Given a finite set Υ of directions, an Υ*-tree* is a set $T \subseteq \Upsilon^*$ such that if $\upsilon \cdot x \in T$, where $\upsilon \in \Upsilon$ and $x \in \Upsilon^*$, then also $x \in T$. The elements of T are called *nodes*, and the empty word ε is the *root* of T. For every $\upsilon \in \Upsilon$ and $x \in T$, the node x is the *parent* of $\upsilon \cdot x$ and $\upsilon \cdot x$ is a *successor* of x. If $z = x \cdot y \in T$ then z is a descendant of y. Each node $x \neq \varepsilon$ of T has a *direction* in Υ. The direction of the root is the symbol $\perp$ (we assume that $\perp \notin \Upsilon$). The direction of a node $\upsilon \cdot x$ is υ. We denote by $dir(x)$ the direction of the node x. An Υ-tree T is a *full infinite tree* if $T = \Upsilon^*$. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $x \in \pi$ there exists a unique $\upsilon \in \Upsilon$ such that $\upsilon \cdot x \in \pi$. Note that our definitions here reverse the standard definitions (e.g., when $\Upsilon = \{0, 1\}$, the successors of the node 0 are 00 and 10, rather than 00 and 01[1].

Given two finite sets Υ and Σ, a Σ*-labeled* Υ*-tree* is a pair $\langle T, \tau \rangle$ where T is an Υ-tree and $\tau : T \to \Sigma$ maps each node of T to a letter in Σ. When Υ and Σ are not important or clear from the context, we call $\langle T, \tau \rangle$ a labeled tree. A tree is *regular* if it is the unwinding of some finite labeled graph. More formally, a *transducer* is a tuple $\mathcal{D} = \langle \Upsilon, \Sigma, Q, q_0, \eta, L \rangle$, where Υ is a finite set of directions, Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\eta : Q \times \Upsilon \to Q$ is a deterministic transition function, and $L : Q \to \Sigma$ is a labeling function. We define $\eta : \Upsilon^* \to Q$ in the standard way: $\eta(\varepsilon) = q_0$ and for $x \in \Upsilon^*$ and $A \in \Upsilon$ we have $\eta(Ax) = \eta(\eta(x), A)$. Intuitively, a transducer is a labeled finite graph with a designated start node, where the edges are labeled by Υ and the nodes are labeled by Σ. A Σ-labeled Υ-tree $\langle \Upsilon^*, \tau \rangle$ is regular if there exists a transducer $\mathcal{D} = \langle \Upsilon, \Sigma, Q, q_0, \eta, L \rangle$, such that for every $x \in \Upsilon^*$, we have $\tau(x) = L(\eta(x))$. We then say that the size of the regular tree $\langle \Upsilon^*, \tau \rangle$, denoted $\|\tau\|$, is $|Q|$, the number of states of $\mathcal{D}$.

Alternating two-way tree automata *Alternating automata* on infinite trees generalize nondeterministic tree automata and were first introduced in [MS87]. Here we describe alternating *two-way* tree automata. For a finite set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X. For a set $Y \subseteq X$ and a formula $\theta \in \mathcal{B}^+(X)$, we say that Y *satisfies* θ iff assigning $\mathbf{true}$ to elements in Y and assigning $\mathbf{false}$ to elements in $X \setminus Y$ makes θ true. For a set Υ of directions, the *extension* of Υ is the set $ext(\Upsilon) = \Upsilon \cup \{\varepsilon, \uparrow\}$ (we assume that $\Upsilon \cap \{\varepsilon, \uparrow\} = \emptyset$). An *alternating two-way automaton* over Σ-labeled Υ-trees is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F \rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+(ext(\Upsilon) \times Q)$ is the transition function, $q_0 \in Q$ is an initial state, and F is the acceptance condition.

A run of an alternating automaton $\mathcal{A}$ over a labeled tree $\langle \Upsilon^*, \tau \rangle$ is a labeled tree $\langle T_r, r \rangle$ in which every node is labeled by an element of $\Upsilon^* \times Q$. A node in T_r, labeled

[1] As will get clearer in the sequel, the reason for that is that rewrite rules refer to the prefix of words.

by (x, q), describes a copy of the automaton that is in the state q and reads the node x of Υ^*. Note that many nodes of T_r can correspond to the same node of Υ^*; there is no one-to-one correspondence between the nodes of the run and the nodes of the tree. The labels of a node and its successors have to satisfy the transition function. Formally, a run $\langle T_r, r\rangle$ is a Σ_r-labeled Γ-tree, for some set Γ of directions, where $\Sigma_r = \Upsilon^* \times Q$ and $\langle T_r, r\rangle$ satisfies the following:

1. $\varepsilon \in T_r$ and $r(\varepsilon) = (\varepsilon, q_0)$.
2. Consider $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, \tau(x)) = \theta$. Then there is a (possibly empty) set $S \subseteq ext(\Upsilon) \times Q$, such that S satisfies θ, and for all $\langle c, q'\rangle \in S$, there is $\gamma \in \Gamma$ such that $\gamma \cdot y \in T_r$ and the following hold:
   - If $c \in \Upsilon$, then $r(\gamma \cdot y) = (c \cdot x, q')$.
   - If $c = \varepsilon$, then $r(\gamma \cdot y) = (x, q')$.
   - If $c =\uparrow$, then $x = \upsilon \cdot z$, for some $\upsilon \in \Upsilon$ and $z \in \Upsilon^*$, and $r(\gamma \cdot y) = (z, q')$.

Thus, ε-transitions leave the automaton on the same node of the input tree, and $\uparrow$-transitions take it up to the parent node. Note that the automaton cannot go up the root of the input tree, as whenever $c =\uparrow$, we require that $x \neq \varepsilon$.

A run $\langle T_r, r\rangle$ is *accepting* if all its infinite paths satisfy the acceptance condition. We consider here *Büchi* and *parity* acceptance conditions [Büc62,EJ91]. A parity condition over a state set Q is a finite sequence $F = \{F_1, F_2, \ldots, F_k\}$ of subsets of Q, where $F_1 \subseteq F_2 \subseteq \ldots \subseteq F_k = Q$. The number k of sets is called the *index* of $\mathcal{A}$. Given a run $\langle T_r, r\rangle$ and an infinite path $\pi \subseteq T_r$, let $\mathit{inf}(\pi) \subseteq Q$ be such that $q \in \mathit{inf}(\pi)$ if and only if there are infinitely many $y \in \pi$ for which $r(y) \in \Upsilon^* \times \{q\}$. That is, $\mathit{inf}(\pi)$ contains exactly all the states that appear infinitely often in π. A path π satisfies the parity condition F if there is an even $1 \leq i \leq k$ such that $\mathit{inf}(\pi) \cap F_i \neq \emptyset$ and for all $j < i$, we have $\mathit{inf}(\pi) \cap F_j = \emptyset$. A Büchi acceptance condition consists of a set $F_1 \subseteq Q$ and it can be viewed as a special case of a parity condition of index 3, where $F = \{\emptyset, F_1, Q\}$. Thus, a run is accepting according to the Büchi condition F_1 if every path in the run visits F_1 infinitely often. An automaton accepts a labeled tree if there exists a run that accepts it. We denote by $\mathcal{L}(\mathcal{A})$ the set of all Σ-labeled trees that $\mathcal{A}$ accepts. The automaton $\mathcal{A}$ is *nonempty* iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$.

An automaton is 1-way if it does not use ϵ-transitions nor $\uparrow$-transitions. Formally, an automaton is 1-way if for every state $q \in Q$ and letter $\sigma \in \Sigma$ the transition $\delta(q, \sigma)$ is restricted to formulas in $B^+(Q)$. An automaton is nondeterministic if in every transition exactly one copy of the automaton is sent in every direction in Υ. Formally, an automaton is nondeterministic if for every state $q \in Q$ and letter $\sigma \in \Sigma$ there exists some set I such that $\delta(q, \sigma) = \bigvee_{i \in I} \bigwedge_{\upsilon \in \Upsilon} (s_{i,\upsilon}, \upsilon)$. Equivalently, we can describe the transition function of a nondeterministic automaton as $\delta : Q \times \Sigma \to 2^{(Q^{|\Upsilon|})}$. The tuple $\langle q_1, \ldots, q_{|\Upsilon|}\rangle \in \delta(q, \sigma)$ is equivalent to the disjunct $(q_1, \upsilon_1) \wedge \ldots \wedge (q_{|\Upsilon|}, \upsilon_{|\Upsilon|})$. We use acronyms in $\{1, 2\} \times \{A, N\} \times \{B, P\} \times \{T, W\}$ to denote the different types of automata. The first symbol stands for the type of movement of the automaton: 1 stands for 1-way automata (we often omit the 1) and 2 stands for 2-way automata. The second symbol stands for the branching mode of the automaton: A for alternating and N for nondeterministic. The third symbol stands for the type of acceptance used by the automaton: B for Büchi and P for parity, and the last symbol stands for the object the automaton is reading: W for words (not used in this paper) and T for trees. For example, a 2APT is a 2-way alternating parity tree automaton and an NPT is a 1-way nondeterministic parity tree automaton.

Theorem 1. *Given a 2APT $\mathcal{A}$ with n states and index k, we can construct an equivalent NPT whose number of states is $(nk)^{O(nk)}$ and whose index is linear in nk* [Var98]*, and we can check the nonemptiness of $\mathcal{A}$ in time $(nk)^{O((nk)^2)}$* [EJS93].

The *membership problem* of a 2APT $\mathcal{A}$ and a regular tree $\langle \Upsilon^*, \tau \rangle$ is to decide whether $\langle \Upsilon^*, \tau \rangle \in \mathcal{L}(\mathcal{A})$. As described in Theorem 2 below, the membership problem can be reduced to the emptiness problem.

Theorem 2. *The membership problem of a regular tree $\langle \Upsilon^*, \tau \rangle$ and a 2APT $\mathcal{A}$ with n states and index k is solvable in time $(|\tau| nk)^{O((nk)^2)}$.*

Proof. According to Theorem 1, we construct a 1NPT $\mathcal{N} = \langle \Sigma, Q, q_0, \delta, F \rangle$ that accepts the language of $\mathcal{A}$. The number of states of $\mathcal{N}$ is exponential in nk and its index is linear in nk. Let $\mathcal{D} = \langle \Upsilon, \Sigma, S, s_0, \eta, L \rangle$ be the transducer generating τ, with $\Upsilon = \{\upsilon_1, \ldots, \upsilon_d\}$. Consider the NPT $\mathcal{N}' = \langle \{a\}, Q \times S, (q_0, s_0), \delta', F \rangle$ where $\delta((q,s),a) = \{ \langle (q_1, s_1), \ldots, (q_d, s_d) \rangle \mid \langle q_1, \ldots, q_d \rangle \in \delta(q, L(s))$ and $s_c = \eta(s, \upsilon_c)$ $\}$. It is easy to see that $\mathcal{L}(\mathcal{N}') \neq \emptyset$ iff $\langle \Upsilon^*, \tau \rangle \in \mathcal{L}(\mathcal{N})$. As $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{A})$, we are done. Note that the number of states of N' is $(|\tau| nk)^{O(nk)}$ and its index is linear in nk. Thus, emptiness of N' can be determined in time $(|\tau| nk)^{O((nk)^2)}$.

Once we translate $\mathcal{A}$ to $\mathcal{N}$, the reduction above is similar the one described in [KVW00]. The translation of $\mathcal{A}$ to $\mathcal{N}$, however, involves an exponential blow up. In the full version of [KPV02] we show that the membership problem for 2ABT is EXPTIME-hard. Thus, the membership problem for 2APT is EXPTIME-complete.

Pushdown tree automata Pushdown tree automata are finite-state automata augmented by a pushdown store. Like a nondeterministic finite-state tree automaton, a nondeterministic pushdown tree automaton starts reading a tree from the root. At each node of the tree, the pushdown automaton consults the transition relation and sends independent copies of itself to each of the node's successors. Each copy has an independent pushdown store that diverges from the pushdown store of the parent. We then check what happens along every branch of the run tree and determine acceptance.

Let $d = |\Upsilon|$ and $\Upsilon = \{\upsilon_1, \ldots, \upsilon_d\}$. A *nondeterministic parity pushdown tree automaton* (with ϵ-transitions) over infinite Υ-trees (or *PD-NPT* for short) is $\mathcal{P} = \langle \Sigma, \Gamma, P, p_0, \alpha_0, \rho, F \rangle$, where Σ is a finite input alphabet, Γ is a finite set of pushdown symbols, P is a finite set of states, $p_0 \in P$ is an initial state, $\alpha_0 \in \Gamma^* \cdot \bot$ is an initial pushdown store content, ρ is a transition function (see below), and F is a parity condition over P.

The transition function $\rho : P \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\bot\}) \to 2^{P \times \Gamma^*} \cup 2^{(P \times \Gamma^*)^d}$ is defined such that for every state $p \in P$ and symbol $A \in \Gamma$, we have $\delta(p, a, A) \in 2^{(P \times \Gamma^*)^d}$, for $a \in \Sigma$, and $\delta(p, \epsilon, A) \in 2^{P \times \Gamma^*}$. Intuitively, when the automaton is in state p, reading a node x labeled by $a \in \Sigma$, and the pushdown store contains a word in $A \cdot \Gamma^*$, it can apply one of the following two types of transitions.

– An *ϵ-transition* in $\delta(p, \epsilon, A)$, where the automaton stays in node x. Accordingly, each ϵ-transition is a pair $(p', \beta) \in P \times \Gamma^*$. Once the automaton chooses a pair (p', β), it moves to state p', and updates the pushdown store by removing A and pushing β.
– An *advancing transition* in $\delta(p, a, A)$, where the automaton splits into d copies, each reading a different successor of x. Accordingly, each advancing transition is a tuple $\langle (p_1, \beta_1), \ldots, (p_d, \beta_d) \rangle \in (P \times \Gamma^*)^d$. Once the automaton chooses a tuple, it splits into d copies, the ith copy moves to the node $i \cdot x$ in the input tree, changes to state p_i, and updates the pushdown store by removing A and pushing β_i.

We assume that the bottom symbol on the pushdown store is $\perp$. This symbol cannot be removed (so, when we say that the pushdown store is empty, we mean that it contains only $\perp$). Every transition that removes $\perp$ also pushes it back. Formally, if $(p', \beta) \in \delta(p, \epsilon, \perp)$, then $\beta \in \Gamma^* \cdot \perp$. Similarly, if $\langle (p_1, \beta_1), \ldots, (p_d, \beta_d) \rangle \in \delta(p, a, \perp)$, then $\beta_i \in \Gamma^* \cdot \perp$ for all $1 \leq i \leq d$. The symbol $\perp$ is not used in another way.

The size $|\rho|$ of the transition function is the sum of all the lengths of the words used in the function. Formally, $|\rho| = \big(\Sigma_{\langle (p_1,\beta_1),\ldots,(p_d,\beta_d)\rangle \in \rho(p,a,A)} |\beta_1| + \ldots + |\beta_d|\big) + \big(\Sigma_{(p',\beta)\in\rho(p,\epsilon,A)} |\beta|\big)$.

We note that the automata defined above assume input trees with a fixed and known branching degree, and can distinguish between the different successors of the node (say, impose a requirement only on the leftmost successor). In many cases, it is useful to consider *symmetric* tree automata [JW95], which refer to the successors of a node in a universal or an existential manner, and thus can handle trees with unknown and varying branching degrees. While symmetry is naturally defined for alternating automata, it can also be defined for nondeterministic automata [KV01], and for PD-NPT.

Example 1. In Section 1, we mentioned the non-regular property "p is uniformly inevitable", namely there is some time i such that p hold at time i in all the computations. We now describe a PD-NPT for the property. We define $\mathcal{P} = \langle 2^{\{p\}}, \{a\}, \{q_0, q_1, q_2\}, q_0, \perp, \delta, F \rangle$, where $F = \{\{q_0, q_1\}, \{q_0, q_1, q_2\}\}$ is such that q_0 and q_1 has to be visited only finitely often, and the transition function is as follows.

– $\rho(q_0, \epsilon, \perp) = \{(q_0, a\perp), (q_1, \perp)\}$,
– $\rho(q_0, \epsilon, a) = \{(q_0, aa), (q_1, a)\}$,
– $\delta(q_1, \{p\}, a) = \delta(q_1, \emptyset, a) = \langle (q_1, \epsilon), \ldots, (q_1, \epsilon) \rangle$,
– $\delta(q_1, \{p\}, \perp) = \langle (q_2, \perp), \ldots, (q_2, \perp) \rangle$, and
– $\delta(q_2, \epsilon, \perp) = \{(q_2, \perp)\}$.

Intuitively, $\mathcal{P}$ starts reading the tree in state q_0 with empty pushdown store. It stays in state q_0 taking ϵ-transitions while pushing a's into the pushdown store. In some stage, $\mathcal{P}$ takes a nondeterministic choice to move to state q_1, from which it proceeds with advancing transitions while removing a's from the pushdown store. When the pushdown store becomes empty, $\mathcal{P}$ takes an advancing transition to state q_2 while checking that the label it reads is indeed $\{p\}$.

A *run* of the PD-NPT $\mathcal{P}$ on an infinite tree $\langle \Upsilon^*, \tau \rangle$ is an $(\Upsilon^* \times P \times \Gamma^*)$-labeled $\mathbb{N}$-tree $\langle T_r, r \rangle$. A node $y \in T_r$ labeled by (x, p, α) represents a copy of $\mathcal{P}$ in state p, with pushdown store content α, reading node x in $\langle \Upsilon^*, \tau \rangle$. Formally, $r(\epsilon) = (\epsilon, p_0, \alpha_0)$, and for all $x \in T_r$ such that $r(x) = (y, p, A \cdot \alpha)$ one of the following holds.

– There is a unique successor $c \cdot x$ of x in T_r such that $r(c \cdot x) = (y, p', \beta \cdot \alpha)$ for some $(p', \beta) \in \delta(p, \epsilon, A)$.
– There are d successors $1 \cdot x, \ldots, d \cdot x$ of x in T_r such that for all $1 \leq c \leq d$, we have $r(c \cdot x) = (\upsilon_c \cdot y, p_c, \beta_c \cdot \alpha)$ for some $\langle (p_1, \beta_1), \ldots, (p_d, \beta_d) \rangle \in \delta(p, V(y), A)$.

Given a path $\pi \subseteq T_r$, we define $inf(\pi) \subseteq P$ to be such that $p \in inf(\pi)$ if and only if there are infinitely many nodes $y \in \pi$ for which $r(y) \in \Upsilon^* \times \{p\} \times \Gamma^*$. As with 2APTs, a path satisfies the parity condition $F = \{F_1, \ldots, F_k\}$ if there is an even $1 \leq i \leq k$ such that $inf(\pi) \cap F_i \neq \emptyset$ and for all $j < i$, we have $inf(\pi) \cap F_j = \emptyset$. A run is *accepting* if every path $\pi \subseteq T$ is accepting. A PD-NPT $\mathcal{P}$ accepts a tree $\langle T, \tau \rangle$ if there exists an accepting run of $\mathcal{P}$ over $\langle T, \tau \rangle$. The language of $\mathcal{P}$, denoted $\mathcal{L}(\mathcal{P})$ contains all trees accepted by $\mathcal{P}$. The PD-NPT $\mathcal{P}$ is *empty* if $\mathcal{L}(\mathcal{P}) = \emptyset$.

Harel and Raz consider only the Büchi acceptance condition (PD-NBT for short) . They showed that the emptiness problem of PD-NBT can be reduced to the emptiness problem of a PD-NBT with one-letter input alphabet [HR94]. The parity acceptance condition is more general than the Büchi acceptance condition. The following theorem generalizes the result of [HR94] to PD-NPT.

Theorem 3. *The emptiness problem for a PD-NPT $\mathcal{P}$ with n states, index k, and input alphabet Σ, is reducible to the emptiness problem for a PD-NPT $\mathcal{P}'$ with $n \cdot |\Sigma|$ states and index k that has a one-letter input alphabet.*

Note that since our automata have ϵ-transitions, we cannot use the classical reduction to one-letter input alphabet [Rab69]. For a nondeterministic tree automaton $\mathcal{N} = \langle \Sigma, P, p_0, \rho, F \rangle$, Rabin constructs the automaton $\mathcal{N}' = \langle \{a\}, P, p_0, \rho', F \rangle$ such that for every state $p \in P$ we have $\rho'(p, a) = \bigcup_{\sigma \in \Sigma} \rho(p, \sigma)$. Thus, $\mathcal{P}'$ guesses which of the input letters $\sigma \in \Sigma$ labels the node and chooses a state in $\rho(p, \sigma)$. For automata with ϵ-transitions, we have to make sure that successor states that read the same node guess the same label for the node, and we augment the automaton $\mathcal{P}'$ with a mechanism that remembers the guessed input letter.

3 The Emptiness Problem for PD-NPT

In this section we give an algorithm to decide the emptiness of a PD-NPT. According to Theorem 3, we can restrict attention to PD-NPT with one-letter input alphabet. We reduce the emptiness of a PD-NPT with one-letter input alphabet $\mathcal{P}$ to the membership of a regular tree in the language of a 2APT $\mathcal{A}$. The idea behind the construction is that since the one-letter tree is homogeneous, the location of a copy of $\mathcal{P}$ in the input tree is not important. Accordingly, when a copy of $\mathcal{A}$ simulates a copy of $\mathcal{P}$, it does not care about the location on the input tree, and it has to remember only the state of the copy and the content of the pushdown store.

It is easy for a copy of $\mathcal{A}$ to remember a state of $\mathcal{P}$. How can $\mathcal{A}$ remember the content of the pushdown store? Let Γ denote the pushdown alphabet of $\mathcal{P}$. Note that the content of the pushdown store of $\mathcal{P}$ corresponds to a node in the full infinite Γ-tree. So, if $\mathcal{A}$ reads the tree Γ^*, it can refer to the location of its reading head in Γ^* as the content of the pushdown store. We would like $\mathcal{A}$ to "know" the location of its reading head in Γ^*. A straightforward way to do so is to label a node $x \in \Gamma^*$ by x. This, however, involves an infinite alphabet, and results in trees that are not regular. Since $\mathcal{P}$ does not read the entire pushdown store's content and (in each transition) it only reads the top symbol on the pushdown store, it is enough to label x by its direction.

Let $\langle \Gamma^*, \tau_\Gamma \rangle$ be the Γ labeled Γ-tree such that for every $x \in \Gamma^*$, we have $\tau_\Gamma(x) = dir(x)$. Note that $\langle \Gamma^*, \tau_\Gamma \rangle$ is a regular tree of size $|\Gamma| + 1$. We reduce the emptiness of a one-letter PD-NPT to the membership problem of $\langle \Gamma^*, \tau_\Gamma \rangle$ in the language of a 2APT. Given a PD-NPT $\mathcal{P}$ we construct a 2APT $\mathcal{A}$ such that $\mathcal{L}(\mathcal{P}) \neq \emptyset$ iff $\langle \Gamma^*, \tau_\Gamma \rangle \in \mathcal{L}(\mathcal{A})$. The 2APT memorizes a control state of the PD-NPT as part of its finite control. When it has to apply some transition of $\mathcal{P}$, it consults its finite control and the label of the tree $\langle \Gamma^*, \tau_\Gamma \rangle$. Knowing the state of $\mathcal{P}$ and the top symbol of the pushdown store, the 2APT can decide which transition of $\mathcal{P}$ to apply. Moving to a new state of $\mathcal{P}$ is done by changing the state of the 2APT. Adjusting the pushdown store's content is done by navigating to a new location in $\langle \Gamma^*, \tau_\Gamma \rangle$.

Theorem 4. *Given a one-letter PD-NPT $\mathcal{P} = \langle \{a\}, \Gamma, P, p_0, \alpha_0, \rho, F \rangle$ with n states and index k, there exists a 2APT $\mathcal{A}$ with $n \cdot |\rho|$ states and index k such that $\mathcal{L}(\mathcal{P}) \neq \emptyset$ iff $\langle \Gamma^*, \tau_\Gamma \rangle \in \mathcal{L}(\mathcal{A})$.*

As before, let $d = |\Upsilon|$ and $\Upsilon = \{\upsilon_1, \ldots, \upsilon_d\}$. Formally, given the PD-NPT $\mathcal{P} = \langle \{a\}, \Gamma, P, p_0, \alpha_0, \delta, F \rangle$, we construct the 2APT $\mathcal{A} = \langle \Gamma, Q, q_0, \eta, F' \rangle$, where

- $Q = P \times tails(\delta)$ where $tails(\delta) \subseteq \Gamma^*$ is the set of all suffixes of words $x \in \Gamma^*$ for which one of the following holds.
  - There are states $p, p_1, \ldots, p_d \in P$, words $\beta_1, \ldots, \beta_d \in \Gamma^*$, and a letter $\gamma \in \Gamma$ such that $\langle (p_1, \beta_1), \ldots, (p_d, \beta_d) \rangle \in \delta(p, \gamma, a)$ and $x = \beta_i$ for some $1 \leq i \leq d$.
  - There are states $p, p' \in P$ and a letter $\gamma \in \Gamma$ such that $(p', x) \in \delta(p, \epsilon, \gamma)$.
  - $x = \alpha_0$.

  Intuitively, when $\mathcal{A}$ visits a node $x \in \Gamma^*$ in state $\langle p, y \rangle$, it checks that $\mathcal{P}$ with initial configuration $(p, y \cdot x)$ accepts the one-letter Υ-tree. In particular, when $y = \varepsilon$, then $\mathcal{P}$ with initial configuration (p, x) needs to accept the one-letter Υ-tree.
  States of the form $\langle p, \varepsilon \rangle$ are called *action states*. From these states $\mathcal{A}$ consults δ in order to impose new requirements on $\langle \Gamma^*, \tau_\Gamma \rangle$. States of the form $\langle p, y \rangle$, for $y \in \Gamma^+$, are called *navigation states*. From these states $\mathcal{A}$ only navigates downward y to reach new action states.
- $q_0 = (p_0, \alpha_0)$. Thus, in its initial state $\mathcal{A}$ checks that $\mathcal{P}$ with initial configuration (p_0, α_0) accepts the one-letter Υ-tree.
- The transition function η is defined for every state $\langle p, x \rangle \in P \times tails(\delta)$ and $A \in \Gamma$ as follows.
  - $\eta((p,\epsilon), A) = \left[\bigvee_{\substack{(t,\alpha) \in \\ \delta(p,\epsilon,A)}} (\uparrow, (t, \alpha)) \right] \vee \left[\bigvee_{\substack{((t_1,\beta_1),\ldots,(t_d,\beta_d)) \in \\ \delta(s,a,A)}} \bigwedge_{i=1}^{d} (\uparrow, (t_i, \beta_i)) \right]$
  - $\eta((p, B \cdot \alpha), A) = (B, (p, \alpha))$

  Thus, in action states, $\mathcal{A}$ reads the direction of the current node and applies a transition from δ. In navigation states, $\mathcal{A}$ needs to go downward to $B \cdot \alpha$, so it continues in direction B.

– $F' = F \times \{\epsilon\}$. Note that only action states can be accepting states.

We show that $\mathcal{A}$ accepts $\langle \Gamma^*, \tau_\Gamma \rangle$ iff $\mathcal{P}$ accepts the one letter Υ-tree. Let $\langle \Upsilon^*, \tau_a \rangle$ denote the labeled tree such that for all $x \in \Upsilon^*$, we have $\tau_a(x) = a$.

Claim. $\mathcal{L}(\mathcal{P}) \neq \emptyset$ iff $\langle \Gamma^*, \tau_\Gamma \rangle \in \mathcal{L}(\mathcal{A})$.

The proof consists of 'translating' a run tree of $\mathcal{P}$ on $\langle \Upsilon^*, \tau_a \rangle$ to a run tree of $\mathcal{A}$ on $\langle \Gamma^*, \tau_\Gamma \rangle$ and vice versa. Every transition $\mathcal{P}$ takes in this run is translated to the appropriate transition of $\mathcal{A}$. The skeleton of the two run trees is identical, the alternating automaton takes a series of moves (the navigation states leading to an action state) to mimick a single move of the pushdown automaton. An action state in the run of $\mathcal{A}$ in location w in the tree Γ^* corresponds to a state of $\mathcal{P}$ with w on the pushdown store.

Combining Theorem 4 with Theorems 2 and 3, we get the following.

Corollary 1. *The emptiness problem for a PD-NPT with n states, index k, and transition function ρ can be solved in time exponential in $nk \cdot |\rho|$.*

Remark 1. Harel and Raz [HR94] show that the emptiness of stack automata on infinite trees is also decidable. Stack automata can read the entire contents of the stack but can change the content only when standing on the top of the stack. They give a doubly exponential reduction from the emptiness problem of stack automata to the emptiness problem of pushdown automata. As their emptiness of pushdown automata is triple exponential, it induces a five fold exponential algorithm for the emptiness of stack automata on infinite trees that use the Büchi acceptance condition. Thus, our emptiness algorithm induces a triple exponential algorithm for the emptiness of Büchi stack automata. We believe that the reduction used in [HR94] for Büchi stack automata can be extended to parity stack automata and furthermore that using our techniques, the emptiness of parity stack automata can be solved in less than triple exponential time.

4 Checking Pushdown Specifications of Finite-State Systems

The *model-checking* problem is to decide whether a given system $\mathcal{S}$ satisfies a specification $\mathcal{P}$. In this section we consider the case where the system is finite state and the specification is a PD-NPT. In order to solve the model-checking problem we combine the system with the PD-NPT and get a PD-NPT whose language is empty iff the system satisfies the specification.

We use *labeled transition graphs* to represent finite-state systems. A labeled transition graph is a quadruple $\mathcal{S} = \langle W, Act, R, w_0 \rangle$, where W is a (possibly infinite) set of states, Act is a finite set of actions, $R \subseteq W \times Act \times W$ is a labeled transition relation, and $w_0 \in W$ is an initial state. We assume that the transition relation R is total (i.e. for every state there exists some action a and some state w' such that $R(w, a, w')$). When $R(w, a, w')$, we say that w' is an *a-successor* of w, and w is an *a-predecessor* of w'. For a state $w \in W$, we denote by $\mathcal{S}^w = \langle W, Act, R, w \rangle$, the system $\mathcal{S}$ with w as its initial state. A finite-state system is given as a labeled transition graph with a finite set of states. The *unwinding* of $\mathcal{S}$ from state $w \in W$ induces an infinite tree $T_\mathcal{S}$. Every node of the tree $T_\mathcal{S}$ is associated with some state $w' \in W$, the root of $T_\mathcal{S}$ is associated with state

w. A node $x \in T_{\mathcal{S}}$ associated with $w' \in W$ has $|\{w'' \mid \exists a \in Act \text{ s.t. } R(w', a, w'')\}|$ successors, each associated with a successor w'' of w' and labeled by the action a such that $R(w', a, w'')$. The root of $T_{\mathcal{S}}$ is labeled by $\perp \notin Act$. As R is total, $T_{\mathcal{S}}$ is infinite. We say that a system $\mathcal{S}$ *satisfies* a tree automaton $\mathcal{P}$ over the alphabet Act if $T_{\mathcal{S}}$ is accepted by $\mathcal{P}$.[2]

The unwinding of a finite labeled transition graph $\mathcal{S}$ results in a regular tree. In order to determine whether $\mathcal{S}$ satisfies $\mathcal{P}$, we have to solve the membership problem of regular trees in the language of a PD-NPT. We reduce the membership problem to the emptiness problem by a construction similar to the one in the proof of Theorem 2. We construct a PD-NPT that either accepts $T_{\mathcal{S}}$ or is empty, and then check its emptiness.

Theorem 5. *Given a finite labeled transition graph $\mathcal{S}$ with m states and a PD-NPT $\mathcal{P}$ with n states, index k, and transition function ρ, the model-checking problem of $\mathcal{S}$ with respect to $\mathcal{P}$ is solvable in time exponential in $mnk \cdot |\rho|$.*

Given a PD-NPT $\mathcal{P} = \langle 2^{AP}, \Gamma, P, p_0, \alpha_0, \rho, F\rangle$ and a finite labeled transition graph $\mathcal{S} = \langle W, Act, R, w_0\rangle$, we construct the PD-NPT $\mathcal{P}' = \langle \{b\}, \Gamma, P \times Act \times W, (p_0, \perp, w_0), \alpha_0, \rho', F'\rangle$ that is the product of the two. The states of $\mathcal{P}'$ consists of triplets of states of $\mathcal{P}$, actions of $\mathcal{S}$, and states of $\mathcal{S}$. The acceptance condition F' is $F \times Act \times W$, where we replace each set $F_i \in F$ by the set $F_i \times Act \times W$. The transition function ρ' maps a triplet (p, a, w) to all the ϵ-successors of p tagged again by a and w and to all the "a-successors" of p tagged by successors of w and the actions taken to get to them. For technical convenience, we assume that the branching degree of $\mathcal{S}$ is uniform and equivalent to the branching degree of the trees that $\mathcal{P}$ reads. Modifying the algorithm to systems with nonuniform branching degree is not too complicated. Formally, we have the following.

- $\rho'((p, a, w), \epsilon, A) = \{\langle (p', a, w), \alpha\rangle \mid \langle p, \alpha\rangle \in \rho(p, \epsilon, A)\}$.
- $\rho'((p, a, w), b, A) = \{\langle\langle (p_1, a_1, w_1), \alpha_1\rangle, \ldots, \langle (p_d, a_d, w_d), \alpha_d\rangle\rangle \mid$
  $\langle\langle p_1, \alpha_1\rangle, \ldots, \langle p_d, \alpha_d\rangle\rangle \in \rho(p, a, A)$ and
  $\{(w, a_1, w_1), \ldots, (w, a_d, w_d)\}$ is the set of transitions from $w\}$.

It is not hard to see that $\mathcal{P}'$ accepts some tree iff $\mathcal{P}$ accepts $T_{\mathcal{S}}$, the unwinding of $\mathcal{S}$.

Remark 2. By having PD-NPT as our specification formalism, we follow the branching-time paradigm to specification and verification, where the specification describes allowed computation trees and a system is correct if its (single) computation tree is allowed. Alternatively, in the linear-time paradigm, the specification describes the allowed linear computations, and the system is correct if all its computations are allowed. When the system is nondeterministic, it may have many computations, and we have to check them all. Thus, while model checking in the branching-time paradigm corresponds to membership checking, model checking in the linear-time paradigm corresponds to checking language containment.

[2] There is a slight technical difficulty as in our formalism PD-NPT run on trees with a uniform branching degrees, while labeled transition graphs are not required to have a uniform outdegree. This difficulty can be finessed by allowing automata on non-uniform trees, as, for example, in [KVW00].

Pushdown specification formalisms are helpful also in the linear-time paradigm [SCFG84]. For example, one can use pushdown word automata to specify unbounded LIFO buffers. Nevertheless, since the containment problem of regular languages in context-free languages is undecidable [HMU00], using pushdown word automata as a specification formalism leads to an undecidable model-checking problem even for finite-state systems.

The branching-time paradigm is more general than the linear-time paradigm, in the sense that we can view a (universally quantified) linear-time specification as a branching-time specification specification [Lam80,Pnu85]. This does not contradict the fact that model checking of pushdown specification is decidable in the branching-time paradigm. Indeed, a translation of a nondeterministic pushdown word automaton that recognizes a language L into a nondeterministic pushdown tree automata that recognizes the language of all the trees derived by L (that is, trees all of whose paths are in L) is not always possible. For cases where such a translation is possible (in particular, when the pushdown word automaton is deterministic), linear model checking is decidable. This is reminiscent of the situation with dense-time temporal logic, where model checking with respect to linear-time specifications is undecidable, while model checking with respect to branching-time specifications is decidable, cf. [ACD93].

Remark 3. Unlike the case of regular tree automata, it can be proved that *alternating* pushdown automata are strictly more expressive than nondeterministic pushdown automaton. For example, it is easy to define a pushdown alternating automaton over words that recognizes the non-context-free language $\{a^i b^i c^i : i \geq 1\}$. Indeed, the automaton can send two copies, one for comparing the number of a's with b's, and one for comparing the number of b's with c's. A similar argument shows that alternating pushdown tree automata are stronger than nondeterministic pushdown tree automata.

On the other hand, as studied in [KVW00], the membership problem for alternating automata is not harder than the one for nondeterministic automata. This observation does not change when pushdown automata are involved. In particular, it is easy to extend Theorem 4 to alternating automata (without changing the blow up), and to extend the model-checking algorithm described above to the stronger framework of alternating pushdown automata.

5 Checking Pushdown Specifications of Context-Free Systems

In this section we show that the decidability results of Section 4 cannot be extended to context-free systems. We show that the model checking problem for context-free systems is undecidable already for pushdown path automata, which are a special case of pushdown tree automata.

We first define context-free systems and pushdown path automata. Again we use labeled transition graphs. This time with an infinite number of states. A *rewrite system* is a quadruple $\mathcal{R} = \langle V, Act, R, x_0 \rangle$, where V is a finite alphabet, Act is a finite set of actions, R maps each action a to a finite set of rewrite rules, to be defined below, and $x_0 \in V^*$ is an initial word. Intuitively, $R(a)$ describes the possible rules that can be applied by taking the action a. We consider here *context-free* rewrite systems. Each rewrite rule is a pair $\langle A, x \rangle \in V \times V^*$. We refer to rewrite rules in $R(a)$ as a-*rules*.

The rewrite system $\mathcal{R}$ induces the labeled transition graph $G_{\mathcal{R}} = \langle V^*, Act, \rho_{\mathcal{R}}, x_0\rangle$, where $\langle x, a, y\rangle \in \rho_{\mathcal{R}}$ if there is a rewrite rule in $R(a)$ whose application on x results in y. In particular, when $\mathcal{R}$ is a context-free rewrite system, then $\rho_{\mathcal{R}}(A \cdot y, a, x \cdot y)$ if $\langle A, x\rangle \in R(a)$. A labeled transition graph that is induced by a context-free rewrite system is called a *context-free graph*.

Consider a labeled transition graph $G = \langle S, Act, \rho, s_0\rangle$. A *nondeterministic pushdown path automaton* on labeled transition graphs is a tuple $\mathcal{P} = \langle Act, \Gamma, P, \delta, p_0, \alpha_0, F\rangle$, where Γ, P, p_0, and α_0 are as in nondeterministic pushdown automata on trees, Act is a set of actions (the automaton's alphabet), and $\delta : P \times Act \times \Gamma \to 2^{P\times\Gamma^*}$ is the transition function. We consider the simpler case where F is a Büchi acceptance condition. Intuitively, when $\mathcal{P}$ is in state p with $A \cdot \alpha$ on the pushdown store and it reads a state s of G, the automaton $\mathcal{P}$ chooses an atom $\langle p', \beta\rangle \in \delta(p, a, A)$ and moves to some a-successor of s in state p' with pushdown store $\beta \cdot \alpha$. Again we assume that the first symbol in α_0 is $\perp$, and that $\perp$ cannot be removed from the pushdown store.

Like a run of a nondeterministic pushdown automaton on words, a run of a path automaton over a labeled transition graph $G = \langle S, Act, \rho, s_0\rangle$ is an infinite word in $(S \times P \times \Gamma^*)^\omega$. A letter (s, p, α), describes that the automaton is in state p of $\mathcal{P}$ with pushdown store content α reading state s of G. Formally, a run is an infinite sequence $(s_0, p_0, \alpha_0), (s_1, p_1, \alpha_1), \ldots \in (S \times P \times \Gamma^*)^\omega$ as follows.

- s_0 is the initial state of G, p_0 is the initial state of $\mathcal{P}$, and α_0 is the initial pushdown store content.
- For every $i \geq 0$ there exists some $a \in Act$ such that s_{i+1} is an a-successor of s_i and if $\alpha_i = A \cdot \alpha$ then $(p_{i+1}, \beta) \in \delta(p_i, a, A)$ and $\alpha_{i+1} = \beta \cdot \alpha$.

A run r is *accepting* if it satisfies the acceptance condition. The graph G is accepted by $\mathcal{P}$ if there is an accepting run on it. Let $\mathcal{L}(\mathcal{P})$ denote the set of graphs accepted by $\mathcal{P}$.

We use PD-NBP (pushdown nondeterministic Büchi path automata) as our specification language. We say that a labeled transition graph G satisfies a PD-NBP $\mathcal{P}$, denoted $G \models \mathcal{P}$, if $\mathcal{P}$ accepts G.

Theorem 6. *The model-checking problem for context-free systems and pushdown path automata is undecidable.*

Proof. We do a reduction from the halting problem for two-counter machines, shown to be undecidable in [Min67] to the problem of whether a context-free graph satisfies a specification given by a PD-NBP. In order to simulate the two-counter machine by the context-free system and the PD-NBP, we use the state of the context-free system (a word in V^*) to maintain the value of the first counter, and we use the pushdown store of the PD-NBP to maintain the value of the second counter. In order to simulate the two counters, we have to be able to check whether the value of each counter is zero or not, and to increase and decrease the value of each counter. Handling of the second counter (maintained by the pushdown store of the path automaton) is straightforward: the path automaton can check whether its pushdown store is empty or not, can push one letter into the pushdown store, and can pop one letter from the pushdown store.

Handling of the first counter (maintained by the state of the context-free system) is a bit more complicated. The context-free system $\mathcal{S}$ has $V = \{a, \perp\}$, and its initial state is $\perp$. The rewrite rules of $\mathcal{S}$ are such that all the reachable states of $\mathcal{S}$ are in $a^* \cdot \perp$. The system $\mathcal{S}$ has five possible actions (which are also read by the PD-NBP $\mathcal{P}$): $push, pop, idle, empty\_push$, and $empty\_idle$. From the state $\perp$, the system $\mathcal{S}$ may apply the actions $empty\_push$ and $empty\_idle$, thus signaling to the specification that its counter is zero. From a state in $a^+ \cdot \perp$, the system $\mathcal{S}$ may apply the actions $push$, pop, or $idle$. The value of the first counter is simulated by the (number of a's in the) location of the PD-NBP $\mathcal{P}$ on the context-free graph. In order to apply a transition of M from a configuration in which the first counter equals zero, $\mathcal{P}$ tries to read the actions $empty\_push$ or $empty\_idle$. In order to increase the first counter, $\mathcal{P}$ reads the action $push$ (or $empty\_push$). Decreasing the counter and leaving it unchanged is similar. The path automaton $\mathcal{P}$ memorizes the state of M in its finite control. It accepts if it gets to an accepting state of M.

We note that path automata are indeed weaker than tree automata. Indeed, a PD-NBT can simulate a PD-NBP by sending copies in accepting sinks to all directions but the direction to which the PD-NBP chooses to go. It follows that the model checking problem for context-free systems and PD-NPT is also undecidable.

6 Conclusions

We consider the model-checking problem for specifications given by pushdown tree automata. We describe an exponential-time algorithm for model checking a finite-state system with respect to a PD-NPT. The algorithm consists of a reduction to the emptiness problem of PD-NPT. The best upper bound known for the emptiness problem is triple exponential, and we improved it to a single exponential. We also show that model checking a context-free system with respect to a PD-NPT specification is undecidable.

References

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, May 1993.

[BEH95] A. Bouajjani, R. Echahed, and P. Habermehl. On the verification problem of nonregular properties for nonregular processes. In *10th LICS*, pp. 123–133, 1995. IEEE.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *8th Concur*, LNCS 1243, pp. 135–150, 1997.

[BER94] A. Bouajjani, R. Echahed, and R. Robbana. Verification of nonregular temporal properties for context-free processes. In *5th Concur*, LNCS 836, pp. 81–97, 1994.

[BLM01] P. Biesse, T. Leonard, and A. Mokkedem. Finding bugs in an alpha microprocessors using satisfiability solvers. In *13th CAV*, LNCS 2102, pp. 454–464. 2001.

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method. and Philos. Sci. 1960*, pages 1–12, Stanford, 1962.

[Bur97b] O. Burkart. Model checking rationally restricted right closures of recognizable graphs. In *2nd INFINITY*, 1997.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

[CFF+01] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M.Y. Vardi. Benefits of bounded model checking at an industrial setting. In *13th CAV*, LNCS 2102, pp. 436–453. Springer-Verlag, 2001.

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[EJ91] E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *32nd FOCS*, pp. 368–377, 1991.

[EJS93] E.A. Emerson, C. Jutla, and A.P. Sistla. On model-checking for fragments of μ-calculus. In *5th CAV*, LNCS 697, pp. 385–396, 1993. Springer-Verlag.

[Eme87] E.A. Emerson. Uniform inevitability is tree automaton ineffable. *Information Processing Letters*, 24(2):77–79, January 1987.

[HMU00] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*. Addison-Wesley, 2000.

[HR94] D. Harel and D. Raz. Deciding emptiness for stack automata on infinite trees. *Information and Computation*, 113(2):278–299, September 1994.

[JW95] D. Janin and I. Walukiewicz. Automata for the modal μ-calculus and related results. In *Proc. 20th MFCS*, LNCS 969, pp. 552–562. Springer-Verlag, 1995.

[KPV02] O. Kupferman, N. Piterman, and M.Y. Vardi. Model checking linear properties of prefix-recognizable systems. In *Proc. 14th CAV*, LNCS 2404, pp. 371–385. 2002.

[KV00] O. Kupferman and M.Y. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In *12th CAV*, LNCS 1855, pp. 36–52. Springer-Verlag, 2000.

[KV01] O. Kupferman and M.Y. Vardi. On clopen specifications. In *Proc. 8th LPAR*, LNAI 2250, pp. 24–38. Springer-Verlag, 2001.

[KVW00] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, March 2000.

[Lam80] L. Lamport. Sometimes is sometimes "not never" - on the temporal logic of programs. In *7th POPL*, pp. 174–185, 1980.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *12th POPL*, pp. 97–107, 1985.

[Min67] M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1967.

[MS85] D.E. Muller and P.E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.

[MS87] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.

[PI95] W. Peng and S. P. Iyer. A new typee of pushdown automata on infinite tree. *IJFCS*, 6(2):169–186, 1995.

[Pnu77] A. Pnueli. The temporal logic of programs. In *18th FOCS*, pp. 46–57, 1977.

[Pnu85] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In *12th ICALP*, pp. 15–32. Springer-Verlag, 1985.

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th Int. Symp. on Programming*, LNCS 137, pp. 337–351, 1981.

[Rab69] M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.

[SCFG84] A. Sistla, E.M. Clarke, N. Francez, and Y. Gurevich. Can message buffers be axiomatized in linear temporal logic. *Information and Control*, 63(1/2):88–112, 1984.

[Var98] M.Y. Vardi. Reasoning about the past with two-way automata. In *25th ICALP*, LNCS 1443, pp. 628–641. Springer-Verlag, 1998.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *1st LICS*, pp. 332–344, 1986.

[Wal01] I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.