Global Model-Checking of Infinite-State Systems

Nir Piterman[1,⋆] and Moshe Y. Vardi[2,⋆⋆]

[1] Weizmann Institute of Science, Department of Computer Science, Rehovot 76100, Israel
nir.piterman@weizmann.ac.il
http://www.wisdom.weizmann.ac.il/~nirp

[2] Rice University, Department of Computer Science, Houston, TX 77251-1892, USA
vardi@cs.rice.edu
http://www.cs.rice.edu/~vardi

Abstract. In this paper we extend the automata-theoretic framework for reasoning about infinite-state sequential systems to handle also the global model-checking problem. Our framework is based on the observation that states of such systems, which carry a finite but unbounded amount of information, can be viewed as nodes in an infinite tree, and transitions between states can be simulated by finite-state automata. Checking that the system satisfies a temporal property can then be done by a two-way automaton that navigates through the tree. The framework is known for local model checking. For branching time properties, the framework uses two-way alternating automata. For linear time properties, the framework uses two-way path automata. In order to solve the global model-checking problem we show that for both types of automata, given a regular tree, we can construct a nondeterministic word automaton that accepts all the nodes in the tree from which an accepting run of the automaton can start.

1 Introduction

An important research topic over the past decade has been the application of model checking to infinite-state systems. A major thrust of research in this area is the application of model checking to *infinite-state sequential systems*. These are systems in which a state carries a finite, but unbounded, amount of information, e.g., a pushdown store. The origin of this thrust is the important result by Muller 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. This started with Burkart and Steffen, who developed an exponential-time algorithm for model-checking formulas in the *alternation-free* μ-calculus with respect to context-free graphs [BS92]. Researchers then went on to extend this result to the μ-calculus, on one hand, and to more general graphs on the other

[⋆] 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, CCR-0124077, CCR-0311326, IIS-9908435, IIS-9978135, EIA-0086264, and ANI-0216467 by BSF grant 9800096, and by a grant from the Intel Corporation.

R. Alur and D.A. Peled (Eds.): CAV 2004, LNCS 3114, pp. 387–400, 2004.

hand, such as *pushdown graphs* [BS95,Wal96], *regular graphs* [BQ96], and *prefix-recognizable graphs* [Cau96]. One of the most powerful results so far is an exponential-time algorithm by Burkart for model checking formulas of the μ-calculus with respect to prefix-recognizable graphs [Bur97][1]. Some of this theory has also been reduced to practice. Pushdown model-checkers such as Mops [CW02], Moped [ES01], and Bebop [BR00] (to name a few) have been developed. Successful applications of these model-checkers to the verification of software are reported, for example, in [CW02].

We usually distinguish between *local* and *global* model-checking. In the first setting we are given a specific state of the system and determine whether it satisfies a given property. In the second setting we compute (a finite representation) of the set of states that satisfy a given property. For many years global model-checking algorithms were the standard; in particular, CTL model checkers [CES86], and symbolic model-checkers [BCM$^+$92] perform global model-checking. While local model checking holds the promise of reduced computational complexity [SW91] and is more natural for explicit LTL model-checking [CVWY92], global model-checking is especially important where the model-checking is only part of the verification process. in [CKKV01] global model-checking is used to supply coverage information, which informs us what parts of the design under verification are relevant to the specified properties. In [LBBO01] an infinite-state system is abstracted into a finite-state system. Global model-checking is performed over the finite-state system and the result is then used to compute invariants for the infinite-state system. In [PRZ01] results of global model-checking over small instances of a parameterized system are generalized to invariants for every value of the system's parameter.

An automata-theoretic framework for reasoning about infinite-state sequential systems was developed in [KV00,KPV02]. The automata-theoretic approach uses the theory of automata as a unifying paradigm for system specification, verification, and synthesis [EJ91,Kur94,VW94]. Automata enable the separation of the logical and the algorithmic aspects of reasoning about systems, yielding clean and asymptotically optimal algorithms. Traditionally automata-theoretic techniques provide algorithms only for local model-checking [CVWY92,KV00,KPV02]. As model-checking in the automata-theoretic approach is reduced to the emptiness of an automaton, it seems that this limitation to local model checking is inherent to the approach. For finite-state systems we can reduce global model-checking to local model-checking by iterating over all the states of the system, which is essentially what happens in symbolic model checking of LTL [BCM$^+$92]. For infinite-state systems, however, such a reduction cannot be applied. In this paper we remove this limitation of automata-theoretic techniques. We show that the automata-theoretic approach to infinite-state sequential systems generalizes nicely to global model-checking. Thus, all the advantages of using automata-theoretic methods, e.g., the ability to handle regular labeling and regular fairness constraints, the ability to handle μ-calculus with backward modalities, and the ability to check realizability [KV00,ATM03], apply also to the more general problem of global model checking.

We use two-way tree alternating automata to reason about properties of infinite-state sequential systems. The idea is based on the observation that states of such systems can

[1] The monadic second-order theory and μ-calculus model-checking over *high-order pushdown graphs* is also decidable [KNU03,Cac03]. The complexity of both problems is nonelementary.

be viewed as nodes in an infinite tree, and transitions between states can be simulated by finite-state automata. Checking that the system satisfies a temporal property can then be done by a two-way alternating automaton. Local model checking is then reduced to emptiness or membership problems for two-way tree automata.

In this work, we give a solution to the global model-checking problem. The set of configurations of a prefix-recognizable system satisfying a μ-calculus property can be infinite, but it is regular, so it is finitely represented. We show how to construct a nondeterministic word automaton that accepts all the configurations of the system that satisfy (resp., do not satisfy) a branching-time (resp., linear-time) property. In order to do that, we study the *global membership* problem for two-way alternating parity tree automata and two-way path automata. Given a regular tree, the global membership problem is to find the set of states of the automaton and locations on the tree from which the automaton accepts the tree. We show that in both cases the question is not harder than the simple membership problem (is the tree accepted from the root and the initial state). Our result matches the upper bounds for global model checking established in [EHRS00,EKS01,Cac02]. Our contribution is in showing how this can be done uniformly in an automata-theoretic framework rather than via an eclectic collection of techniques.

2 Preliminaries

Labeled Rewrite Systems. A *labeled transition graph* is $G = \langle \Sigma, S, L, \rho, s_0 \rangle$, where Σ is a finite set of labels, S is a (possibly infinite) set of states, $L : S \to \Sigma$ is a labeling function, $\rho \subseteq S \times S$ is a transition relation, and $s_0 \in S_0$ is an initial state. When $\rho(s, s')$, we say that s' is a *successor* of s, and s is a *predecessor* of s'. For a state $s \in S$, we denote by $G^s = \langle \Sigma, S, L, \rho, s \rangle$, the graph G with s as its initial state. An *s-computation* is an infinite sequence of states $s_0, s_1, \ldots \in S^\omega$ such that $s_0 = s$ and for all $i \geq 0$, we have $\rho(s_i, s_{i+1})$. An s-computation $s_0, s_1, \ldots$ induces the *s-trace* $L(s_0) \cdot L(s_1) \cdots \in \Sigma^\omega$. Let $\mathcal{T}_s \subseteq \Sigma^\omega$ be the set of all s-traces.

A *rewrite system* is $R = \langle \Sigma, V, Q, L, T \rangle$, where Σ is a finite set of labels, V is a finite alphabet, Q is a finite set of states, $L : Q \times V^* \to \Sigma$ is a labeling function that depends only on the first letter of x (Thus, we may write $L : Q \times V \cup \{\epsilon\} \to \Sigma$. Note that the label is defined also for the case that x is the empty word ϵ). The finite set of rewrite rules T is defined below. The set of *configurations* of the system is $Q \times V^*$. Intuitively, the system has finitely many control states and an unbounded store. Thus, in a configuration $(q, x) \in Q \times V^*$ we refer to q as the *control state* and to x as the *store*. We consider here two types of rewrite systems. In a *pushdown* system, each rewrite rule is $\langle q, A, x, q' \rangle \in Q \times V \times V^* \times Q$. Thus, $T \subseteq Q \times V \times V^* \times Q$. In a *prefix-recognizable* system, each rewrite rule is $\langle q, \alpha, \beta, \gamma, q' \rangle \in Q \times \mathit{reg}(V) \times \mathit{reg}(V) \times \mathit{reg}(V) \times Q$, where $\mathit{reg}(V)$ is the set of regular expressions over V. Thus, $T \subseteq Q \times \mathit{reg}(V) \times \mathit{reg}(V) \times \mathit{reg}(V) \times Q$. For a word $w \in V^*$ and a regular expression $r \in \mathit{reg}(V)$ we write $w \in r$ to denote that w is in the language of the regular expression r. We note that the standard definition of prefix-recognizable systems does not include control states. Indeed, a prefix-recognizable system without states can simulate a prefix-recognizable system with states by having the state as the first letter of the unbounded store. We use prefix-recognizable systems with control states for the sake of uniform notation.

The rewrite system R starting in configuration (q_0, x_0) induces the labeled transition graph $G_R^{(q_0,x_0)} = \langle \Sigma, Q \times V^*, L', \rho_R, (q_0, x_0)\rangle$. The states of G_R are the configurations of R and $\langle (q, z), (q', z')\rangle \in \rho_R$ if there is a rewrite rule $t \in T$ leading from configuration (q, z) to configuration (q', z'). Formally, if R is a pushdown system, then $\rho_R((q, A \cdot y), (q', x \cdot y))$ if $\langle q, A, x, q'\rangle \in T$; and if R is a prefix-recognizable system, then $\rho_R((q, x \cdot y), (q', x' \cdot y))$ if there are regular expressions α, β, and γ such that $x \in \alpha$, $y \in \beta$, $x' \in \gamma$, and $\langle q, \alpha, \beta, \gamma, q'\rangle \in T$. Note that in order to apply a rewrite rule in state $(q, z) \in Q \times V^*$ of a pushdown graph, we only need to match the state q and the first letter of z with the second element of a rule. On the other hand, in an application of a rewrite rule in a prefix-recognizable graph, we have to match the state q and we should find a partition of z to a prefix that belongs to the second element of the rule and a suffix that belongs to the third element. A labeled transition graph that is induced by a pushdown system is called a *pushdown graph*. A labeled transition system that is induced by a prefix-recognizable system is called a *prefix-recognizable graph*.

Consider a prefix-recognizable system $R = \langle \Sigma, V, Q, L, T\rangle$. For a rewrite rule $t_i = \langle\, s, \alpha_i, \beta_i, \gamma_i, s'\,\rangle \in T$, let $\mathcal{U}_\lambda = \langle V, Q_\lambda, q_\lambda^0, \eta_\lambda, F_\lambda\rangle$, for $\lambda \in \{\alpha_i, \beta_i, \gamma_i\}$, be the nondeterministic automaton for the language of the regular expression λ. We assume that all initial states have no incoming edges and that all accepting states have no outgoing edges. We collect all the states of all the automata for α, β, and γ regular expressions. Formally, $Q_\alpha = \bigcup_{t_i \in T} Q_{\alpha_i}$, $Q_\beta = \bigcup_{t_i \in T} Q_{\beta_i}$, and $Q_\gamma = \bigcup_{t_i \in T} Q_{\gamma_i}$.

We define the *size* $\|T\|$ of T as the space required in order to encode the rewrite rules in T and the labeling function. Thus, in a pushdown system, $\|T\| = \sum_{\langle q,A,x,q'\rangle \in T} |x|$, and in a prefix-recognizable system, $\|T\| = \sum_{\langle q,\alpha,\beta,\gamma,q'\rangle \in T} |\mathcal{U}_\alpha| + |\mathcal{U}_\beta| + |\mathcal{U}_\gamma|$.

We are interested in specifications expressed in the μ-calculus [Koz83] and in LTL [Pnu77]. For introduction to these logics we refer the reader to [Eme97]. We want to model check pushdown and prefix-recognizable systems with respect to specifications in these logics. We differentiate between *local* and *global* model-checking. In *local model-checking*, given a graph G and a specification φ, one has to determine whether G satisfies φ. In *global model-checking* we are interested in the set of configurations s such that G^s satisfies φ. As G is infinite, we hope to find a finite representation for this set. It is known that the set of configurations of a prefix-recognizable system satisfying a monadic second-order formula is regular [Cau96,Rab72], which implies that this also holds for pushdown systems and for μ-calculus and LTL specifications.

In this paper, we extend the automata-theoretic approach to model-checking of sequential infinite state systems [KV00,KPV02] to global model-checking. Our model-checking algorithm returns a nondeterministic finite automaton on words (NFW, for short) recognizing the set of configurations that satisfy (not satisfy, in the case of LTL) the specification. The complexity of our algorithms matches the previously known upper bounds [EHRS00,EKS01,Cac02][2].

Theorem 1. *Global model-checking for a system R and a specification φ is solvable*

- *in time $(\|T\|)^3 \cdot 2^{O(|\varphi|)}$ and space $(\|T\|)^2 \cdot 2^{O(|\varphi|)}$, where R is a pushdown system and φ is an LTL formula.*

[2] In order to obtain the stated bound for prefix-recognizable systems and LTL specifications one has to combine the result in [EKS01] with our reduction from prefix-recognizable systems to pushdown systems with regular labeling [KPV02].

- *in time* $(\|T\|)^3 \cdot 2^{O(|\varphi|\cdot|Q_\beta|)}$ *and space* $(\|T\|)^2 \cdot 2^{O(|\varphi|\cdot|Q_\beta|)}$*, where* R *is a prefix-recognizable system and* φ *is an LTL formula.*
- *in time* $2^{O(\|T\|\cdot|\varphi|\cdot k)}$*, where* R *is a prefix-recognizable system and* φ *is a* μ*-calculus formula of alternation depth* k*.*

Alternating Two-Way Automata. 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 $\upsilon \cdot x$. 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 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 $\upsilon \cdot x \in \pi$.

Given two finite sets Υ and Σ, a Σ*-labeled* Υ*-tree* is a pair $\langle T, V\rangle$ where T is an Υ-tree and $V : T \to \Sigma$ maps each node of T to a letter in Σ. When Υ and Σ are not important or clear from the context, we call $\langle T, V\rangle$ a labeled tree.

A tree is *regular* if it is the unwinding of some finite labeled graph. More formally, a *transducer* $\mathcal{D}$ is a tuple $\langle \Upsilon, \Sigma, Q, q_0, \eta, L\rangle$, 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 a start 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 $\eta(ax) = \eta(\eta(x), a)$. 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 $\tau(x) = L(\eta(x))$. The size of $\langle \Upsilon^*, \tau\rangle$, denoted $\|\tau\|$, is $|Q|$, the number of states of $\mathcal{D}$.

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 (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$), where we also allow the formulas **true** and **false**, and, as usual, $\wedge$ has precedence over $\vee$. For a set $Y \subseteq X$ and a formula $\theta \in \mathcal{B}^+(X)$, we say that Y *satisfies* θ iff assigning **true** to elements in Y and assigning **false** to elements in $X \setminus Y$ makes θ true. For a set Υ of directions, the *extension* of Υ is the set $ext(\Upsilon) = \Upsilon \cup \{\varepsilon, \uparrow\}$ (assuming $\Upsilon \cap \{\varepsilon, \uparrow\} = \emptyset$). An *alternating two-way automaton* over Σ-labeled Υ-trees is $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F\rangle$, where Σ is the input alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to \mathcal{B}^+(ext(\Upsilon) \times Q)$ is the transition function, and F specifies the acceptance condition.

A run of an alternating automaton $\mathcal{A}$ over a labeled tree $\langle \Upsilon^*, V\rangle$ is a Σ_r-labeled Γ-tree, for some set Γ of directions, where $\Sigma_r = \Upsilon^* \times Q$. The root ε of T_r is labeled by (ε, q_0). The labels of a node and its successors have to satisfy the transition function: Consider $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \theta$. 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: (a) If $c \in \Upsilon$, then $r(\gamma \cdot y) = (c \cdot x, q')$. (b) If $c = \varepsilon$, then $r(\gamma \cdot y) = (x, q')$. (c) If $c = \uparrow$, then $x = \upsilon \cdot z$, for some $\upsilon \in \Upsilon$ and $z \in \Upsilon^*$, and $r(\gamma \cdot y) = (z, q')$.

A run $\langle T_r, r\rangle$ is *accepting* if all its infinite paths satisfy the acceptance condition. We consider here *parity* acceptance conditions [EJ91]. A parity condition over a state set Q is a finite sequence $F = \{F_1, F_2, \ldots, F_m\}$ of subsets of Q, where $F_1 \subseteq F_2 \subseteq \ldots \subseteq F_m = Q$. The number m 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, $\mathit{inf}(\pi)$ is the set of states that appear infinitely often in π. A path π satisfies the condition F if there is an even i for which $\mathit{inf}(\pi) \cap F_i \neq \emptyset$ and $\mathit{inf}(\pi) \cap F_{i-1} = \emptyset$. An automaton accepts a labeled tree if and only 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 $\mathcal{L}(\mathcal{A}) \neq \emptyset$. The Büchi condition $F \subseteq Q$ is equivalent to the parity condition $\langle \emptyset, F, Q\rangle$ [Büc62]. A path π satisfies the Büchi condition F iff $\mathit{inf}(\pi) \cap F \neq \emptyset$.

The size of an automaton is determined by the number of its states and the size of its transition function. The size of the transition function is $|\eta| = \Sigma_{q\in Q}\Sigma_{a\in\Sigma}|\eta(q,a)|$ where, for a formula in $B^+(ext(\Upsilon) \times Q)$ we define $|(\Delta, q)| = |\mathbf{true}| = |\mathbf{false}| = 1$ and $|\theta_1 \vee \theta_2| = |\theta_1 \wedge \theta_2| = |\theta_1| + |\theta_2| + 1$.

We say that $\mathcal{A}$ is one-way if δ is restricted to formulas in $B^+(\Upsilon \times Q)$, it is advancing if δ is restricted to $B^+((\Upsilon \cup \{\varepsilon\}) \times Q)$. We say that $\mathcal{A}$ is nondeterministic if its transitions are of the form $\bigvee_{i\in I} \bigwedge_{\upsilon\in\Upsilon}(\upsilon, q^i_\upsilon))$, in such cases we write $\delta : Q \times \Sigma \rightarrow 2^{Q^{|\Upsilon|}}$. In the case that $|\Upsilon| = 1$, $\mathcal{A}$ is a word automaton.

Given an alternating two-way parity tree automaton $\mathcal{A}$ with n states and index k, we can construct an equivalent nondeterministic one-way parity tree automaton whose number of states is exponential in nk and whose index is linear in nk [Var98], and we can check the nonemptiness of $\mathcal{A}$ in time exponential in nk [EJS93]. The *membership problem* of an automaton $\mathcal{A}$ and a regular tree $\langle \Upsilon^*, \tau\rangle$ is to determine whether $\mathcal{A}$ accepts $\langle \Upsilon^*, \tau\rangle$; or equivalently whether $\langle \Upsilon^*, \tau\rangle \in \mathcal{L}(\mathcal{A})$. For $q \in Q$ and $w \in \Upsilon^*$, we say that $\mathcal{A}$ accepts $\langle \Upsilon^*, \tau\rangle$ from (q, w) if there exists an accepting run of $\mathcal{A}$ that starts from state q reading node w (i.e. a run whose root is labeled by (w, q) and satisfies the transition). The *global membership problem* of $\mathcal{A}$ and regular tree $\langle \Upsilon^*, \tau\rangle$ is to determine the set $\{(q, w) \mid \mathcal{A} \text{ accepts } \langle \Upsilon^*, \tau\rangle \text{ from } (q, w)\}$.

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: 1 for 1-way or advancing automata (we often omit the 1) and 2 for 2-way automata. The second symbol stands for the branching mode: A for alternating and N for nondeterministic. The third symbol stands for the type of acceptance: B for Büchi and P for parity, and the last symbol stands for the object the automaton is reading: W for words and T for trees. For example, a 2APT is a 2-way alternating parity tree automaton and an NBW is a 1-way nondeterministic Büchi word automaton.

Alternating Automata on Labeled Transition Graphs. Consider a labeled transition graph $G = \langle \Sigma, S, L, \rho, s_0\rangle$. Let $\Delta = \{\varepsilon, \Box, \Diamond\}$. An alternating automaton on labeled transition graphs (*graph automaton*, for short) [Wil99][3] is a tuple $\mathcal{S} = \langle \Sigma, Q, q_0, \delta, F\rangle$, where Σ, Q, q_0, and F are as in alternating two-way automata, and $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(\Delta \times Q)$ is the transition function. Intuitively, when $\mathcal{S}$ is in state q and it reads a state s of G, fulfilling an atom $\langle \Diamond, t\rangle$ (or $\Diamond t$, for short) requires $\mathcal{S}$ to send a copy in state t to some successor of s. Similarly, fulfilling an atom $\Box t$ requires $\mathcal{S}$ to send copies

[3] See related formalism in [JW95].

in state t to all the successors of s. Thus, graph automata cannot distinguish between the various successors of a state and treat them in an existential or universal way.

Like runs of alternating two-way automata, a run of a graph automaton $\mathcal{S}$ over a labeled transition graph $G = \langle \Sigma, S, L, \rho, s_0 \rangle$ is a Σ_r-labeled Γ-tree $\langle T_r, r \rangle$, where Γ is some set of directions, $\Sigma_r = S \times Q$. The root ε of T_r is labeled by (s_0, q_0). The labels of a node and its successors have to satisfy the transition function: Consider $y \in T_r$ with $r(y) = (s, q)$ and $\delta(q, L(s)) = \theta$. Then there is a (possibly empty) set $S \subseteq \Delta \times Q$, such that S satisfies θ, and for all $\langle c, q' \rangle \in S$, we have: (a) If $c = \varepsilon$, then there is $\gamma \in \Gamma$ such that $\gamma \cdot y \in T_r$ and $r(\gamma \cdot y) = (s, q')$. (b) If $c = \Box$, then for every successor s' of s, there is $\gamma \in \Gamma$ such that $\gamma \cdot y \in T_r$ and $r(\gamma \cdot y) = (s', q')$. (c) If $c = \Diamond$, then there is a successor s' of s and $\gamma \in \Gamma$ such that $\gamma \cdot y \in T_r$ and $r(\gamma \cdot y) = (s', q')$. Acceptance is defined as in 2APT runs. The graph G is accepted by $\mathcal{S}$ if there is an accepting run on it. We denote by $\mathcal{L}(\mathcal{S})$ the set of all graphs that $\mathcal{S}$ accepts and by $\mathcal{S}^q = \langle \Sigma, Q, q, \delta, F \rangle$ the automaton $\mathcal{S}$ with q as its initial state.

We use graph automata as our branching time specification language. A labeled transition graph G satisfies a graph automaton $\mathcal{S}$, denoted $G \models \mathcal{S}$, if $\mathcal{S}$ accepts G. Graph automata have the same expressive power as the μ-calculus. Formally, given a μ-calculus formula ψ, of length n and alternation depth k, we can construct a graph parity automaton $\mathcal{S}_\psi$ such that $\mathcal{L}(\mathcal{S}_\psi)$ is exactly the set of graphs satisfying ψ. The automaton $\mathcal{S}_\psi$ has n states and index k [Wil99].

We use NBW as our linear time specification language. A labeled transition graph G satisfies an NBW N, denoted $G \models N$, if $\mathcal{T}_{s_0} \cap L(N) \neq \emptyset$ (where s_0 is the initial state of G)[4]. We are especially interested in cases where $\Sigma = 2^{AP}$, for some set AP of atomic propositions AP, and in languages $L \subseteq (2^{AP})^\omega$ definable by NBW or formulas of the linear temporal logic LTL [Pnu77]. For an LTL formula φ, the *language* of φ, denoted $L(\varphi)$, is the set of infinite words that satisfy φ. For every LTL formula φ, we can construct an NBW N_φ with $2^{O(|\varphi|)}$ states such that $L(N_\varphi) = L(\varphi)$ [VW94].

Given a graph G and a specification $\mathcal{S}$, the *global model-checking* problem is to compute the set of configurations s of G such that $G^s \models \mathcal{S}$. Whether we are interested in branching or linear time model-checking is determined by the type of automaton.

3 Global Membership for 2APT

In this section we solve the global membership problem for 2APT. Consider a 2APT $\mathcal{A} = \langle \Sigma, S, s_0, \rho, \alpha \rangle$ and a regular tree $T = \langle \Upsilon^*, \tau \rangle$. Our construction consists of a few stages. First, we modify $\mathcal{A}$ into a 2APT $\mathcal{A}'$ that starts its run from the root of the tree in an idle state. In this idle state it goes to a node in the tree that is marked with a state of $\mathcal{A}$. From that node, the new automaton starts a fresh run of $\mathcal{A}$ from the marked state. We convert $\mathcal{A}'$ into an NPT $\mathcal{P}$ [Var98]. Second, we combine $\mathcal{P}$ with an NBT $\mathcal{D}'$ that accepts only trees that are identical to the regular tree T and in addition have exactly one node marked by some state of $\mathcal{A}$. We check now the emptiness of this automaton $\mathcal{A}''$. From the emptiness information we derive an NFW N that accepts a word $w \in \Upsilon^*$

[4] Notice, that our definition dualizes the usual definition for LTL. Here, we say that a linear time specification is satisfied if there exists a trace that satisfies it. Usually, a linear time specification is satisfied if all traces satisfy it.

in state $s \in S$ (i.e. the run ends in state s of $\mathcal{A}$; state s is an accepting state of N) iff $\mathcal{A}$ accepts T from (s, w).

Theorem 2. *Consider a 2APT $\mathcal{A} = \langle \varSigma, S, s_0, \rho, \alpha \rangle$ and a regular tree $T = \langle \varUpsilon^*, \tau \rangle$. We can construct an NFW $N = \langle \varUpsilon, R' \cup S, r_0, \varDelta, S \rangle$ that accepts the word w in state $s \in S$ iff $\mathcal{A}$ accepts T from (s, w). Let n be the number of states of $\mathcal{A}$ and h its index; the NFW N is constructible in time exponential in nh and polynomial in $||\tau||$.*

Proof. Let $S_+ = S \cup \{\bot\}$ and $\varUpsilon = \{\upsilon_1, \ldots, \upsilon_k\}$. Consider the 2APT $\mathcal{A}' = \langle \varSigma \times S_+, S', s'_0, \rho', \alpha' \rangle$ where $S' = S \cup \{s'_0\}$, s'_0 is a new initial state, α' is identical to α except having s'_0 belonging to some odd set of α', and ρ' is defined as follows.

$$\rho'(s,(\sigma,t)) = \begin{cases} \rho(s,\sigma) & s \neq s'_0 \\ \bigvee_{\upsilon \in \varUpsilon}(\upsilon, s'_0) & s = s'_0 \text{ and } t = \bot \\ \bigvee_{\upsilon \in \varUpsilon}(\upsilon, s'_0) \vee (\varepsilon, s') & s = s'_0 \text{ and } t = s' \end{cases}$$

Clearly, $\mathcal{A}'$ accepts a $(\varSigma \times S_+)$-labeled tree T' iff there is a node x in T' labeled by (σ, s) for some $(\sigma, s) \in \varSigma \times S$ and $\mathcal{A}$ accepts the projection of T' on $\varSigma$ when it starts its run from node x in state s. Let $\mathcal{P} = \langle \varSigma \times \mathcal{S}_+, P, p_0, \rho_1, \alpha_1 \rangle$ be the NPT that accepts exactly those trees accepted by $\mathcal{A}'$ [Var98]. If $\mathcal{A}$ has n states and index h then $\mathcal{P}$ has $(nh)^{O(nh)}$ states and index $O(nh)$.

Let $\mathcal{D} = \langle \varUpsilon, \varSigma, Q, q_0, \eta, L \rangle$ be the transducer inducing the labeling τ of T. We construct an NBT $\mathcal{D}'$ that accepts $(\varSigma \times S_+)$-labeled trees whose projection on $\varSigma$ is τ and have exactly one node marked by a state in S. Consider the NBT $\mathcal{D}' = \langle \varSigma \times S_+, Q \times \{\bot, \top\}, (q_0, \bot), \eta', Q \times \{\top\} \rangle$ where η' is defined as follows. For $q \in Q$ let $pend_i(q) = \langle (\eta(q, \upsilon_1), \top), \ldots, (\eta(q, \upsilon_i), \bot), \ldots, (\eta(q, \upsilon_k), \top) \rangle$ be the tuple where the j-th element is the υ_j-successor of q and all elements are marked by $\top$ except for the i-th element, which is marked by $\bot$. Intuitively, a state $(q, \top)$ accepts a subtree all of whose nodes are marked by $\bot$. A state $(q, \bot)$ means that D' is still searching for the unique node labeled by a state in S. The transition to $pend_i$ means that D' is looking for that node in direction $\upsilon_i \in \varUpsilon$.

$$\eta'((q,\beta),(\sigma,\gamma)) = \begin{cases} \{\langle (\eta(q,\upsilon_1),\top), \ldots, (\eta(q,\upsilon_k),\top) \rangle\} & \beta = \top,\ \gamma = \bot \text{ and } \sigma = L(q) \\ \{\langle (\eta(q,\upsilon_1),\top), \ldots, (\eta(q,\upsilon_k),\top) \rangle\} & \beta = \bot,\ \gamma \in S \text{ and } \sigma = L(q) \\ \{pend_i(q) \mid i \in [1..k]\} & \beta = \gamma = \bot \text{ and } \sigma = L(q) \\ \emptyset & \text{Otherwise} \end{cases}$$

Clearly, $\mathcal{D}'$ accepts a $(\varSigma \times S_+)$-labeled tree T' iff the projection of T' on $\varSigma$ is exactly τ and all nodes of T' are labeled by $\bot$ except one node labeled by some state $s \in S$.

Let $\mathcal{A}'' = \langle \varSigma \times S_+, R, r_0, \delta, \alpha_2 \rangle$ be the product of $\mathcal{D}'$ and $\mathcal{P}$ where $R = (Q \times \{\bot, \top\}) \times P$, $r_0 = ((q_0, \bot), p_0)$, δ is defined below and $\alpha_2 = \langle F'_1, \ldots, F'_m \rangle$ is obtained from $\alpha_1 = \langle F_1, \ldots, F_m \rangle$ by setting $F'_1 = ((Q \times \{\bot, \top\}) \times F_1) \cup (Q \times \{\bot\} \times P)$ and for $i > 1$ we have $F'_i = (Q \times \{\top\}) \times F_i$. Thus, $\bot$ states are visited finitely often, and only the state of $\mathcal{P}$ is important for acceptance. For every state $((q, \beta), p) \in (Q \times \{\bot, \top\}) \times P$ and letter $(\sigma, \gamma) \in \varSigma \times S_+$ the transition function δ is defined by:

$$\delta(((q,\beta),p),(\sigma,\gamma)) = \left\{ \langle ((q_1,\beta_1),p_1), \ldots, ((q_k,\beta_k),p_k) \rangle \;\middle|\; \begin{array}{c} \langle p_1, \ldots, p_k \rangle \in \rho_1(p,(\sigma,\gamma)) \text{ and} \\ \langle (q_1,\beta_1), \ldots, (q_k,\beta_k) \rangle \in \eta'((q,\beta),(\sigma,\gamma)) \end{array} \right\}$$

A tree T' accepted by $\mathcal{A}''$ has a unique node x labeled by a state s of $\mathcal{A}$, all other nodes are labeled by $\perp$, and if T is the projection of T' on Σ then $\mathcal{A}$ accepts T from (s, x).

The number of states of $\mathcal{A}''$ is $\|\tau\| \cdot (nh)^{O(nh)}$ and its index is $O(nh)$. We can check whether $\mathcal{A}''$ accepts the empty language in time exponential in nh. The emptiness algorithm returns the set of states of $\mathcal{A}''$ whose language is not empty [EJS93]. From now on we remove from the state space of $\mathcal{A}''$ all states whose language is empty. Thus, transitions of $\mathcal{A}''$ contain only tuples in which all states have non empty language.

We are ready to construct the NFW N. The states of N are the states of $\mathcal{A}''$ in $(Q \times \{\perp\}) \times P$ in addition to S (the set of states of $\mathcal{A}$). Every state in S is an accepting sink of N. For the transition of N we follow transitions of $\perp$-states. Once we can transition into a tuple where the $\perp$ is removed, we transition into the appropriate accepting states.

Let $N = \langle \Upsilon, R' \cup S, r_0, \Delta, S \rangle$, where $R' = R \cap (Q \times \{\perp\} \times P)$, r_0 is the initial state of $\mathcal{A}''$, S is the set of states of $\mathcal{A}$ (accepting sinks in N), and Δ is defined as follows. Consider a state $((q, \perp), p) \in R'$. For every tuple $\langle\ ((q_1, \top), p_1), \ldots, ((q_i, \perp), p_i), \ldots, ((q_k, \top), p_k)\ \rangle$ in $\delta(((q, \perp), p), (L(q), \perp))$, we add $((q_i, \perp), p_i)$ to $\Delta(((q, \perp), p), \upsilon_i)$. For every tuple $\langle ((q_1, \top), p_1), \ldots, ((q_k, \top), p_k) \rangle$ in $\delta(((q, \perp), p), (L(q), s))$, we add s to $\Delta(((q, \perp), p), \epsilon)$. In the full version we show that N accepts $w \in \Upsilon^*$ in a state $s \in S$ iff $\mathcal{A}$ accepts T from (w, s).

4 Two-Way Path Automata on Trees

Path automata on trees are a hybrid of nondeterministic word automata and nondeterministic tree automata: they run on trees but have linear runs. Here we describe *two-way* nondeterministic Büchi path automata. We introduced path automata in [KPV02], where they are used to give an automata-theoretic solution to the local linear time model checking problem. A *two-way nondeterministic Büchi path automaton* (2NBP, for short) on Σ-labeled Υ-trees is a 2ABT where the transition is restricted to disjunctions. Formally, $\mathcal{S} = \langle \Sigma, P, p_0, \delta, F \rangle$, where Σ, P, p_0, and F are as in an NBW, and $\delta : P \times \Sigma \to 2^{(ext(\Upsilon) \times P)}$ is the transition function. A path automaton that visits the state p and reads the node $x \in T$ chooses a pair $(d, p') \in \delta(p, \tau(x))$, and then follows direction d and moves to state p'. It follows that a *run* of a 2NBP on a labeled tree $\langle \Upsilon^*, \tau \rangle$ is a sequence of pairs $r = (x_0, p_0), (x_1, p_1), \ldots$. The run is *accepting* if it visits F infinitely often. As usual, $\mathcal{L}(\mathcal{S})$ denotes the set of trees accepted by $\mathcal{S}$.

We studied in [KPV02] the emptiness and membership problems for 2NBP. Here, we consider the global membership problem of 2NBP. Consider a 2NBP $\mathcal{S} = \langle\ \Sigma, P, p_0, \delta, F\ \rangle$ and a regular tree $\langle \Upsilon^*, \tau \rangle$. Just like in the case of 2APT, we first modify $\mathcal{S}$ into a 2NBP $\mathcal{S}'$ that starts its run from the root of the tree in an idle state. In this idle state it goes to some node in the tree. From that node, the new automaton starts a fresh run of $\mathcal{S}$ from some state in P. We then construct an ABW $\mathcal{A}$ such that $\mathcal{A}$ accepts the word a^ω iff $\mathcal{S}'$ accepts the tree T. We check the emptiness of $\mathcal{A}$, and from the emptiness information derive an NFW N that accepts a word $w \in \Upsilon^*$ in state $p \in P$ (i.e. the run ends in state p of $\mathcal{S}$; state p is an accepting state of N) iff $\mathcal{S}$ accepts $\langle \Upsilon^*, \tau \rangle$ from (p, w).

Theorem 3. *Consider a 2NBP $\mathcal{S} = \langle \Sigma, P, p_0, \delta, F \rangle$ and a regular tree $\langle \Upsilon^*, \tau \rangle$. We can construct an NFW $N = \langle \Upsilon, Q' \cup P, q_0, \Delta, P \rangle$ that accepts the word w in a state*

$p \in P$ iff $\mathcal{S}$ accepts T from (p, w). We construct N in time $O(|P|^2 \cdot |\delta| \cdot \|\tau\|)$ and space $O(|P|^2 \cdot \|\tau\|)$.

Proof. Consider the 2NBP $\mathcal{S}' = \langle \Sigma, P', p_0, \delta', F\rangle$ where $P' = P \cup \{p_0\}$ and $p_0 \notin P$ is a new state, for every $p \in P$ and $\sigma \in \Sigma$ we have $\delta'(p, \sigma) = \delta(p, \sigma)$, and for every $\sigma \in \Sigma$ we have $\delta'(p_0, \sigma) = \bigvee_{\upsilon \in \Upsilon}(p_0, \upsilon) \vee \bigvee_{p \in P}(\varepsilon, p)$. Thus, $\mathcal{S}'$ starts reading $\langle \Upsilon^*, \tau\rangle$ from the root in state p_0, the transition of p_0 includes either transitions down the tree that remain in state p_0 or transitions into one of the other states of $\mathcal{S}$. Thus, every accepting run of $\mathcal{S}'$ starts with a sequence $(p_0, w_0), (p_0, w_1), \ldots, (p_0, w_n), (p, w_n), \ldots$. Such a run is a witness to the fact that $\mathcal{S}$ accepts $\langle \Upsilon^*, \tau\rangle$ from (p, w_n). We would like to recognize all words $w \in \Upsilon^*$ and states $p' \in P$ for which there exist runs as above with $p = p'$ and $w_n = w$.

Consider the regular tree $\langle \Upsilon^*, \tau\rangle$. Let $\mathcal{D}_\tau$ be the transducer that generates the labels of τ where $\mathcal{D}_\tau = \langle \Upsilon, \Sigma, D_\tau, d_\tau^0, \rho_\tau, L_\tau\rangle$. For a word $w \in \Upsilon^*$ we denote by $\rho_\tau(w)$ the unique state that $\mathcal{D}_\tau$ gets to after reading w. In [KPV02] we construct the ABW $\mathcal{A} = \langle \{a\}, Q, q_0, \eta, F'\rangle$ where $Q = (P' \cup (P' \times P')) \times D_\tau \times \{\bot, \top\}$, $q_0 = \langle p_0, d_\tau^0, \bot\rangle$, and $F' = (F \times D_\tau \times \{\bot\}) \cup (P' \times D_\tau \times \{\top\})$. We use the following definitions. Two functions $f_\alpha : P' \times P' \to \{\bot, \top\}$ where $\alpha \in \{\bot, \top\}$.

$$f_\bot(p, q) = \bot \qquad f_\top(p, q) = \begin{cases} \bot \text{ if } p \in F \text{ or } q \in F \\ \top \text{ otherwise} \end{cases}$$

For every state $p \in P'$ and letter $\sigma \in \Sigma$ the set C_p^σ is the set of states from which p is reachable by a sequence of ϵ-transitions reading letter σ and one final $\uparrow$-transition reading σ. Now η is defined for every state in Q as follows.

$$\eta(p, d, \alpha) = \bigvee \begin{array}{l} \bigvee_{p' \in P'} \bigvee_{\beta \in \{\bot, \top\}}(\epsilon, \langle p, p', d, \beta\rangle) \wedge (\epsilon, \langle p', d, \beta\rangle) \\ \bigvee_{\upsilon \in \Upsilon} \bigvee_{\langle \upsilon, p'\rangle \in \delta'(p, L_\tau(d))}(a, \langle p', \rho_\tau(d, \upsilon), \bot\rangle) \\ \bigvee_{\langle \epsilon, p'\rangle \in \delta'(p, L_\tau(d))}(\epsilon, \langle p', d, \bot\rangle) \end{array}$$

$$\eta(p_1, p_2, d, \alpha) = \\ \bigvee \begin{array}{l} \bigvee_{\langle \epsilon, p'\rangle \in \delta'(p_1, L_\tau(d))}(\epsilon, \langle p', p_2, d, f_\alpha(p', p_2)\rangle) \\ \bigvee_{p' \in P'} \bigvee_{\beta_1 + \beta_2 = \alpha} \left(\begin{array}{c} (\epsilon, \langle p_1, p', d, f_{\beta_1}(p_1, p')\rangle) \wedge \\ (\epsilon, \langle p', p_2, d, f_{\beta_2}(p', p_2)\rangle) \end{array} \right) \\ \bigvee_{\upsilon \in \Upsilon, \langle \upsilon, p'\rangle \in \delta'(p_1, L_\tau(d))} \bigvee_{p'' \in C_{p2}^{L_\tau(d)}}(a, \langle p', p'', \rho_\tau(d, \upsilon), f_\alpha(p', p'')\rangle) \end{array}$$

Finally, we replace every state of the form $\{\langle p, p, d, \alpha\rangle \mid p \in F \text{ or } \alpha = \bot\}$ by **true**.

In [KPV02] we show that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\langle \Upsilon^*, \tau\rangle \in \mathcal{L}(\mathcal{S}')$ by translating an accepting run of $\mathcal{S}'$ on $\langle \Upsilon^*, \tau\rangle$ into an accepting run tree of $\mathcal{A}$ on a^ω and vice versa. As shown in [KPV02] the number of states of $\mathcal{A}$ is $O(|P|^2 \cdot \|\tau\|)$ and the size of its transition is $O(|\delta| \cdot |P|^2 \cdot \|\tau\|)$. It is also shown there that because of the special structure of $\mathcal{A}$ its emptiness can be computed in space $O(|P|^2 \cdot \|\tau\|)$ and in time $O(|\delta| \cdot |P|^2 \cdot \|\tau\|)$. From the emptiness algorithm we can get a table $T : Q \to \{0, 1\}$ such that $T(q) = 1$ iff $L(\mathcal{A}^q) \neq \emptyset$. Furthermore, we can extract from the algorithm an accepting run of $\mathcal{A}^q$ on a^ω. It follows that in case $(p, d, \alpha) \in P \times D_\tau \times \{\bot, \top\}$ the run is infinite and the algorithm in [KPV02] can be used to extract from it an accepting run of P on the regular tree $\langle \Upsilon^*, \tau_d\rangle$. If $(p, p', d, \alpha) \in P \times P \times D_\tau \times \{\bot, \top\}$ the run is finite and the algorithm in [KPV02] can be used to extract from it a run of P on the regular tree $\langle \Upsilon^*, \tau_d\rangle$ that starts in state p and ends in state p' both reading the root of Υ^*.

We construct now the NFW N. Let $N = \langle \Upsilon, Q' \cup P, q_0, \Delta, P\rangle$ where $Q' = (\{p_0\} \cup (\{p_0\} \times P)) \times D_\tau \times \{\bot, \top\}$ and P is the set of states of $\mathcal{S}$ (that serves also as the set of accepting states), $q_0 = \langle p_0, d^0_\tau, \bot\rangle$ is the initial state of $\mathcal{A}$, and Δ is defined as follows.

Consider a state $\langle p_0, d, \alpha\rangle \in Q'$. For every $\upsilon \in \Upsilon$ such that the language of $\langle p_0, \rho_\tau(d, \upsilon), \bot\rangle$ is not empty, and $\{(a, \langle p_0, \rho_\tau(d, \upsilon), \bot\rangle)\} \models \eta(\langle p_0, d, \alpha\rangle)$, we add $\langle p_0, \rho_\tau(d, \upsilon), \bot\rangle$ to $\Delta(\langle p_0, d, \alpha\rangle, \upsilon)$. For every state $p \in P$ such that the language of $\langle p_0, p, d, \beta\rangle$ is not empty, the language of $\langle p, d, \beta\rangle$ is not empty, and $\{(\varepsilon, \langle p_0, p, d, \beta\rangle), (\varepsilon, p, d, \beta)\} \models \eta(\langle p_0, d, \alpha\rangle)$, we add $\langle p_0, p, d, \beta\rangle$ to $\Delta(\langle p_0, d, \alpha\rangle, \varepsilon)$. For every state $p \in P$ such that the language of $\langle p, d, \bot\rangle$ is not empty, and $\{(\varepsilon, \langle p, d, \bot\rangle)\} \models \eta(\langle p_0, d, \alpha\rangle)$, we add (the accepting state) p to $\Delta(\langle p_0, d, \alpha\rangle, \varepsilon)$.

Consider a state $\langle p_0, p, d, \alpha\rangle \in Q'$. For every $\upsilon \in \Upsilon$ and for every $p' \in C_p^{L_\tau(d)}$ such that the language of $\langle p_0, p', \rho_\tau(d, \upsilon), f_\alpha(p_0, p')\rangle$ is not empty, and $\{(a, \langle p_0, p', \rho_\tau(d, \upsilon), f_\alpha(p_0, p')\rangle)\} \models \eta(\langle p_0, p, d, \alpha\rangle)$, we add the state $\langle p_0, p', \rho_\tau(d, \upsilon), f_\alpha(p_0, p')\rangle$ to the transition $\Delta(\langle p_0, p', d, \alpha\rangle, \upsilon)$. For every p' such that the language of $\langle p', p, d, f_\alpha(p', p)\rangle$ is not empty, and $\{(\varepsilon, \langle p', p, d, f_\alpha(p', p)\rangle)\} \models \eta(\langle p_0, p, d, \alpha\rangle)$, we add the state p' to the transition $\Delta(\langle p_0, p, d, \alpha\rangle, \varepsilon)$. For every state $p' \in P$ such that the language of $\langle p_0, p', d, \beta_1\rangle$ is not empty, the language of $\langle p', p, d, \beta_2\rangle$ is not empty, and we have $\{(\varepsilon, \langle p_0, p', d, \beta_1\rangle), (\varepsilon, \langle p', p, d, \beta\rangle)\} \models \eta(\langle p_0, p, d, \alpha\rangle)$, we add the state $\langle p_0, p', d, \beta_1\rangle$ to the transition $\Delta(\langle p_0, p, d, \alpha\rangle, \varepsilon)$.

This completes the definition of the automaton. In the full version we prove that N accepts $w \in \Upsilon^*$ in a state $p \in P$ iff $\mathcal{S}$ accepts $\langle \Upsilon^*, \tau\rangle$ from (p, w).

5 Global Model Checking

In this section we solve the global model-checking problem by a reduction to the global membership problem. The constructions are somewhat different from the constructions in [KV00,KPV02] as we use the global-membership of 2APT and 2NBP instead of the emptiness of 2APT and membership of 2NBP. We start with branching time model checking and then proceed to linear time.

Consider a rewrite system $R = \langle \Sigma, V, Q, L, T\rangle$. Recall that a configuration of R is a pair $(q, x) \in Q \times V^*$. Thus, the store x corresponds to a node in the full infinite V-tree. An automaton that reads the tree V^* can memorize in its state space the state component of the configuration and refer to the location of its reading head in V^* as the store. We would like the automaton to "know" the location of its reading head in V^*. A straightforward way to do so is to label a node $x \in V^*$ by x. This, however, involves an infinite alphabet. We show that labeling every node in V^* by its direction is sufficiently informative to provide the 2APT with the information it needs in order to simulate transitions of the rewrite system. Let $\langle V^*, \tau_V\rangle$ be the tree where $\tau_V(x) = dir(x)$.

Theorem 4. *Given a pushdown or a prefix-recognizable system $R = \langle \Sigma, V, Q, L, T\rangle$ and a graph automaton $\mathcal{W} = \langle \Sigma, W, w_0, \delta, F\rangle$, we can construct a 2APT $\mathcal{A}$ on V-trees and a function f that associates states of $\mathcal{A}$ with states of R such that $\mathcal{A}$ accepts $\langle V^*, \tau_V\rangle$ from (p, x) iff $G_R^{(f(p),x)} \models \mathcal{W}$. The automaton $\mathcal{A}$ has $O(|Q| \cdot \|T\| \cdot |V|)$ states, and has the same index as $\mathcal{W}$.*

The construction in Theorem 4 reduces the global model-checking problem to the global membership problem of a 2APT. By Theorem 2, we then have the following.

Theorem 5. *Global model-checking for a pushdown or a prefix-recognizable system $R = \langle \Sigma, V, Q, L, T\rangle$ and a graph automaton $\mathcal{W} = \langle \Sigma, W, w_0, \delta, F\rangle$, can be solved in time exponential in nk, where $n = |Q| \cdot \|T\| \cdot |V|$ and k is the index of $\mathcal{W}$.*

According to [Wil99], we can conclude with an EXPTIME bound also for the global model-checking problem of μ-calculus formulas, matching the lower bound in [Wal96]. Note that the fact the same complexity bound holds for pushdown and prefix-recognizable rewrite systems stems from the different definition of $\|T\|$ in the two cases.

We now turn to linear time specifications. As branching time model-checking is exponential in the system and linear time model-checking is polynomial in the system, we do not want to reduce linear time model-checking to branching time model-checking.

As before, the 2NBP reads the full infinite V-tree. It uses its location as the store and memorizes as part of its state the state of the rewrite system. For pushdown systems it is sufficient to label a node in the tree by its direction. For prefix-recognizable systems we label a node x, in addition to its direction, by the regular expressions β for which $x \in \beta$. We denote this tree by $\langle V^*, \tau_\beta\rangle$ and its size is exponential in $|Q_\beta|$.

Theorem 6. *Given a pushdown or a prefix-recognizable system $R = \langle \Sigma, V, Q, L, T\rangle$ and an NBW $N = \langle \Sigma, W, w_0, \eta, F\rangle$, we can construct a 2NBP $\mathcal{S}$ on V-trees and a function f that associates states of $\mathcal{S}$ with states of R such that $\mathcal{S}$ accepts $\langle V^*, \tau_V\rangle$ (or $\langle V^*, \tau_\beta\rangle$) from (s, x) iff $G_R^{(f(s),x)} \models N$. The size of the transition function of $\mathcal{S}$ is $O(\|T\| \cdot |N|)$ and it has $O(|Q| \cdot \|T\| \cdot |N|)$ states in the case of pushdown systems, and $O(|Q| \cdot (|Q_\alpha| + |Q_\gamma|) \cdot |T| \cdot |N|)$ states in the case of prefix-recognizable systems.*

Combining Theorem 3 and Theorem 6 we get the following.

Theorem 7. *Global model-checking for a rewrite system R and NBW N is solvable in time $O((\|T\| \cdot |N|)^3)$ and space $O((\|T\| \cdot |N|)^2)$ when R is a pushdown system and in time $(\|T\| \cdot |N|)^3 \cdot 2^{O(|Q_\beta|)}$ and space $(|T| \cdot |N|)^2 \cdot 2^{O(|Q_\beta|)}$ when R is a prefix-recognizable system.*

Our complexity coincides with the one in [EHRS00], for pushdown systems, and with the result of combining [EKS01] and [KPV02], for prefix-recognizable systems.

6 Conclusions

We have shown how to extend the automata-theoretic approach to model-checking infinite state sequential rewrite systems to global model-checking. In doing so we have shown that the restriction of automata-theoretic methods to local model-checking is not an inherent restriction of this approach. Our algorithms generalize previous automata-theoretic algorithms for local model-checking [KV00,KPV02]. The complexity of our algorithm matches the complexity bounds of previous algorithms for global model-checking [EHRS00,EKS01,KPV02,Cac02] and show that a uniform solution exists in the automata-theoretic framework.

We believe that our algorithms generalize also to *micro-macro stack systems* [PV03] and to high order pushdown systems [KNU03,Cac03] as the algorithms for local model-checking over these types of systems are also automata-theoretic.

References

[ATM03] R. Alur, S. La Torre, and P. Madhusudan. Modular strategies for infinite games on recursive game graphs. In *15th CAV*, *LNCS* 2725, 67–79, Springer-Verlag, 2003.

[BCM$^+$92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *IC*, 98(2):142–170, 1992.

[BQ96] O. Burkart and Y.-M. Quemener. Model checking of infinite graphs defined by graph grammars. In *1st Infinity*, *ENTCS* 6, 1996.

[BR00] T. Ball and S. Rajamani. Bebop: A symbolic model checker for boolean programs. In *7th SPIN Workshop*, *LNCS* 1885, 113–130, 2000. Springer.

[BS92] O. Burkart and B. Steffen. Model checking for context-free processes. In *3rd Concur*, *LNCS* 630, 123–137. Springer, 1992.

[BS95] O. Burkart and B. Steffen. Composition, decomposition and model checking of pushdown processes. *Nordic J. Comput.*, 2:89–125, 1995.

[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, 1962.

[Bur97] O. Burkart. Model checking rationally restricted right closures of recognizable graphs. In *2nd Infinity*, 1997.

[Cac02] T. Cachat. Uniform solution of parity games on prefix-recognizable graphs. In *4th Infinity*, *ENTCS* 68(6), 2002.

[Cac03] T. Cachat. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In *30th ICALP*, *LNCS* 2719, 556–569, 2003. Springer.

[Cau96] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *23rd ICALP*, volume 1099 of *LNCS*, 194–205. Springer, 1996.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *TOPLAS*, 8(2), 1986.

[CKKV01] H. Chockler, O. Kupferman, R.P. Kurshan, and M.Y. Vardi. A practical approach to coverage in model checking. In *13th CAV*, *LNCS* 2102, 66–78. Springer, 2001.

[CVWY92] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *FMSD*, 1:275–288, 1992.

[CW02] H. Chen and D. Wagner. Mops: an infrastructure for examining security properties of software. In *9th CCS*, 235–244, 2002. ACM.

[EHRS00] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *12th CAV*, *LNCS* 1855, 232–247, 2000. Springer.

[EJ91] E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *32nd FOCS*, 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, 385–396, 1993. Springer.

[EKS01] J. Esparza, A. Kucera, and S. Schwoon. Model-checking LTL with regular valuations for pushdown systems. In *4th STACS*, *LNCS* 2215, 316–339, 2001. Springer.

[Eme97] E.A. Emerson. Model checking and the μ-calculus. In *Descriptive Complexity and Finite Models*, 185–214. AMS, 1997.

[ES01] J. Esparza and S. Schwoon. A BDD-based model checker for recursive programs. In *13th CAV*, *LNCS* 2102, 324–336, 2001. Springer.

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

[KNU03] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *5th FOSSACS*, *LNCS* 2303, 205–222, 2003. Springer.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, 1983.

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

[Kur94] R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. 1994.

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

[LBBO01] Y. Lakhnech, S. Bensalem, S. Berezin, and S. Owre. Incremental verification by abstraction. In *7th TACAS*, *LNCS* 2031, 98–112, 2001. Springer.

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

[MS87] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *TCS*, 54, 1987.

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

[PRZ01] A. Pnueli, S. Ruah, and L. Zuck. Automatic deductive verification with invisible invariants. In *7th TACAS*, *LNCS* 2031, 82–97, 2001. Springer.

[PV03] N. Piterman and M.Y. Vardi. Micro-macro stack systems: A new frontier of decidability for sequential systems. In *18th LICS*, 381–390, 2003. IEEE.

[Rab72] M.O. Rabin. Automata on infinite objects and Church's problem. *AMS*, 1972.

[SW91] C. Stirling and D. Walker. Local model checking in the modal μ-calculus. *TCS*, 89(1):161–177, 1991.

[Var98] M.Y. Vardi. Reasoning about the past with two-way automata. In *25th ICALP*, 1998.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *IC*, 115(1), 1994.

[Wal96] I. Walukiewicz. Pushdown processes: games and model checking. In *8th CAV*, 1996.

[Wil99] T. Wilke. CTL^{+} is exponentially more succinct than CTL. In *19th FSTTCS*, 1999.