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
Email: nir.piterman@wisdom.weizmann.ac.il, URL: http://www.wisdom.weizmann.ac.il/~nirp
[2] 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. 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 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. 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 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 [Bur97b]. See also [BE96,BEM97,Bur97a,FWW97,BS99,BCMS00].[3] Some of this theory has also been reduced to practice. Pushdown model-checkers such as Mops [CW02], Moped [ES01,Sch02], 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 [BR01,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

[3] Recently, it was shown that the monadic second-order theory of *high-order pushdown graphs* is decidable [KNU03]. This was adapted to solve μ-calculus model-checking over such graphs, but the complexity of model-checking μ-calculus on a high order pushdown graph of level n is a stack of n exponentials [Cac03].

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. For example, in [CKV01,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 [Sha00,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] (see exposition in [Cac02a]). The automata-theoretic approach uses the theory of automata as a unifying paradigm for system specification, verification, and synthesis [WVS83,EJ91,Kur94,VW94,KVW00]. 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 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 [BEM97,EHRS00,EKS01,KPV02,Cac02b]. 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

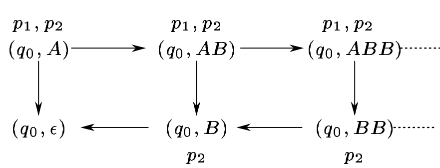
2.1 Labeled Rewrite Systems

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*.

Example 1. The pushdown system $\langle 2^{\{p_1,p_2\}}, \{A, B\}, \{q_0\}, L, T \rangle$, with $T = \{ \langle q_0, A, AB, q_0 \rangle, \langle q_0, A, \varepsilon, q_0 \rangle, \langle q_0, B, \varepsilon, q_0 \rangle \}$, and $L(q_0, A) = \{p_1, p_2\}\}$, $L(q_0, B) = \{p_2\}$, and $L(q_0, \epsilon) = \emptyset$ when starting from (q_0, A) induces the labeled transition graph on the right.



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 states 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. Our results match the previously known upper bounds [EHRS00,EKS01,Cac02b].[4]

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.*
- *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.*

2.2 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 $\upsilon \cdot x \in \pi$. Note that our definitions here dualize the standard definitions (e.g., when $\Upsilon = \{0, 1\}$, the successors of the node 0 are 00 and 10, rather than 00 and 01)[5].

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. We say that an $((\Upsilon \cup \{\perp\}) \times \Sigma)$-labeled Υ-tree $\langle T, V\rangle$ is Υ*-exhaustive* if for every node $x \in T$, we have $V(x) \in \{dir(x)\} \times \Sigma$.

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

[4] 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].

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

$\mathcal{D} = \langle \Upsilon, \Sigma, Q, q_0, \eta, L \rangle$, such that for every $x \in \Upsilon^*$, we have $\tau(x) = L(\eta(x))$. We then say that the size of $\langle \Upsilon^*, \tau \rangle$, 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 $\mathbf{true}$ and $\mathbf{false}$, 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 $\mathbf{true}$ 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 $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F \rangle$, 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 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 by (x, q), describes a copy of the automaton that is in the state q and reads the node x of Υ^*. 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 $\delta(q, V(x)) = \theta$. 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:
   – 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 *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 $inf(\pi) \subseteq Q$ be such that $q \in inf(\pi)$ if and only if there are infinitely many $y \in \pi$ for which $r(y) \in \Upsilon^* \times \{q\}$. That is, $inf(\pi)$ 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 $inf(\pi) \cap F_i \neq \emptyset$ and $inf(\pi) \cap F_{i-1} = \emptyset$. 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 acceptance condition [Büc62] is a private case of parity of index 3. The Büchi condition $F \subseteq Q$ is equivalent to the parity condition $\langle \emptyset, F, Q \rangle$. A path π satisfies the Büchi condition F iff $inf(\pi) \cap F \neq \emptyset$.

We say that $\mathcal{A}$ is one-way if δ is restricted to formulas in $B^+(\Upsilon \times Q)$. 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 $\delta : Q \times \Sigma \to 2^{Q^{|\Upsilon|}}$. In the case that $|\Upsilon| = 1$, $\mathcal{A}$ is a word automaton.

Theorem 2. *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 satisfying Condition 2 above where the root of the run tree is labeled by (w, q)). 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 of the automaton: 1 for 1-way automata (we often omit the 1) and 2 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 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.

2.3 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] [6] 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 $\delta : Q \times \Sigma \to \mathcal{B}^+(\Delta \times Q)$ 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 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 labeled tree in which every node is labeled by an element of $S \times Q$. A node labeled by (s, q), describes a copy of the automaton that is in the state q of $\mathcal{S}$ and reads the state s of G. Formally, a run is a Σ_r-labeled Γ-tree $\langle T_r, r \rangle$, where Γ is some set of directions, $\Sigma_r = S \times Q$, and $\langle T_r, r \rangle$ satisfies the following:

1. $\varepsilon \in T_r$ and $r(\varepsilon) = (s_0, q_0)$.
2. 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:
   - If $c = \varepsilon$, then there is $\gamma \in \Gamma$ such that $\gamma \cdot y \in T_r$ and $r(\gamma \cdot y) = (s, q')$.
   - If $c = \Box$, then for every successor s' of s, there is $\gamma \in \Gamma$ such that $\gamma \cdot y \in T_r$ and $r(\gamma \cdot y) = (s', q')$.
   - 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. We say that 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,

Theorem 3. [Wil99] *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.*

We use NBW as our linear time specification language. We say that 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)[7]. We are especially interested in cases where $\Sigma = 2^{AP}$, for some set AP of

[6] See related formalism in [JW95].

[7] 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 it if all traces satisfy it.

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.

Theorem 4. [VW94] *For every LTL formula φ, there exists an NBW N_φ with $2^{O(|\varphi|)}$ states such that $L(N_\varphi) = L(\varphi)$.*

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 used.

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 two 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 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 $w \in \Upsilon^*$ 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 5. *Consider a 2APT $\mathcal{A} = \langle \Sigma, S, s_0, \rho, \alpha \rangle$ and a regular tree $T = \langle \Upsilon^*, \tau \rangle$. We can construct an NFW $N = \langle \Upsilon, R' \cup S, r_0, \Delta, S \rangle$ 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.*

Proof. Let $S_+ = S \cup \{\perp\}$ and $\Upsilon = \{\upsilon_1, \ldots, \upsilon_k\}$. Consider the 2APT $\mathcal{A}' = \langle \Sigma \times S_+, S', s_0', \rho', \alpha \rangle$ where $S' = S \cup \{s_0'\}$, s_0' is a new initial state and ρ' is defined as follows.

$$\rho'(s, (\sigma, t)) = \begin{cases} \rho(s, \sigma) & s \neq s_0' \\ \bigvee_{\upsilon \in \Upsilon} (\upsilon, s_0') & s = s_0' \text{ and } t = \perp \\ \bigvee_{\upsilon \in \Upsilon} (\upsilon, s_0') \vee (\varepsilon, s') & s = s_0' \text{ and } t = s' \end{cases}$$

Clearly, $\mathcal{A}'$ accepts a $(\Sigma \times S_+)$-labeled tree T' iff there is a node x in T' labeled by (σ, s) for some $(\sigma, s) \in \Sigma \times S$ and $\mathcal{A}$ accepts the projection of T' on Σ when it starts its run from node x in state s. Let $\mathcal{P} = \langle \Sigma \times \mathcal{S}_+, P, p_0, \rho_1, \alpha_1 \rangle$ 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 $\mathcal{D} = \langle \Upsilon, \Sigma, Q, q_0, \eta, L \rangle$ be the transducer inducing the labeling τ of T. We construct an NBT $\mathcal{D}'$ that accepts $(\Sigma \times S_+)$-labeled trees whose projection on Σ is τ and have exactly one node marked by a state in S. Consider the NBT $\mathcal{D}' = \langle \Sigma \times S_+, Q \times \{\perp, \top\}, (q_0, \perp), \eta', Q \times \{\top\} \rangle$ where η' is defined as follows. For $q \in Q$ let $pend_i(q) = \langle (\eta(q, \upsilon_1), \top), \ldots, (\eta(q, \upsilon_i), \perp), \ldots, (\eta(q, \upsilon_k), \top) \rangle$ 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 $\perp$. Intuitively, a state $(q, \top)$ accepts a subtree all of whose nodes are marked by $\perp$. A state $(q, \perp)$ 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 $\upsilon_i \in \Upsilon$.

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

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

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

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

Every tree T' accepted by $\mathcal{A}''$ has a unique node x labeled by a state s of $\mathcal{A}$ and 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 such that all states in the tuple 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 below.

Consider a state $((q, \perp), p) \in R'$. Its transition in $\mathcal{A}''$ is of the form

$$\delta\left(((q,\perp),p),(L(q),\perp)\right) = \left\{ \langle ((q_1,\top),p_1),\ldots,((q_i,\perp),p_i),\ldots,((q_k,\top),p_k)\rangle \;\middle|\; \begin{array}{c} q_j = \eta(q,\upsilon_j) \text{ and} \\ \langle p_1,\ldots,p_k\rangle \in \rho_1(p,(L(q),\perp)) \end{array} \right\}$$

$$\delta\left(((q,\perp),p),(L(q),s)\right) = \left\{ \langle ((q_1,\top),p_1),\ldots,((q_k,\top),p_k)\rangle \;\middle|\; \begin{array}{c} q_j = \eta(q,\Upsilon_j) \text{ and} \\ \langle p_1,\ldots,p_k\rangle \in \rho_1(p,(L(q),\perp)) \end{array} \right\}$$

For every tuple $\langle ((q_1, \top), p_1), \ldots, ((q_i, \perp), p_i), \ldots, ((q_k, \top), p_k) \rangle$, we add $((q_i, \perp), p_i)$ to $\Delta(((q, \perp), p), \upsilon_i)$. For every tuple $\langle ((q_1, \top), p_1), \ldots, ((q_k, \top), p_k) \rangle$, we add the letter s used in the transition to $\Delta(((q, \perp), p), \epsilon)$.

Lemma 1. *A word $w \in \Upsilon^*$ is accepted by N in a state $s \in S$ iff $\mathcal{A}$ accepts T from (w, s).*

The proof of the Lemma is in Appendix A.

4 Global Model Checking of Branching Time Properties

In this section we solve the global model-checking for branching time specifications by a reduction to the global membership problem for 2APT. The construction is somewhat different from the construction in [KV00] as we use the global-membership of 2APT instead of the emptiness of 2APT.

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 $\tau(x) = dir(x)$.

Theorem 6. *Given a pushdown system $R = \langle \Sigma, V, Q, L, T\rangle$ and a graph automaton $\mathcal{W} = \langle \Sigma, W, w_0, \delta, F\rangle$, there is 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 $\mathcal{W}$.*

States of the automaton $\mathcal{A}$ have three components: a state of $\mathcal{W}$, a state of R, and navigation information. States are partitioned into *action* states and *navigation* states. An action state that includes state w of $\mathcal{W}$ and state q of R and reads node x (in an accepting run of $\mathcal{A}$) means that $\mathcal{W}$ starting in w accepts $G_R^{(q,x)}$. A navigation state, contains the information on how to navigate to a new node in the tree. From an action state, in order to check that the requirements imposed by state w of $\mathcal{W}$ on the graph $G_R^{(q,x)}$, the transition $\Diamond w'$ is simulated by $\mathcal{A}$ by sending a copy that navigates to some successor of confi guration (q, x) and from there applies new actions. A transition $\Box w'$ is simulated by $\mathcal{A}$ by sending copies that navigate to all the successors of confi guration (q, x). The full proof of Theorem 6 is in Appendix B.1.

We extend the above construction to prefi x-recognizable systems. Again the two-way automaton navigates through the full V-tree and simulates transitions of the rewrite system. In order to apply a rewrite rule $\langle q, \alpha_i, \beta_i, \gamma_i, q'\rangle$, the automaton goes up the tree along a word in α_i, it checks that the suffi x is in β_i by sending a separate copy to the root, and moves downwards along a word in γ_i.

Theorem 7. *Given a prefix-recognizable system $R = \langle \Sigma, V, Q, L, T\rangle$ and a graph automaton $\mathcal{W} = \langle \Sigma, W, w_0, \delta, F\rangle$, there is 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 $\mathcal{W}$.*

As in the case of pushdown systems, states of the automaton $\mathcal{A}$ contain a state of $\mathcal{W}$, a state of R, and navigation information. The navigation information, relating to a rewrite rule $\langle q, \alpha_i, \beta_i, \gamma_i, q'\rangle$, is a state of an automaton $\mathcal{U}_\lambda$ for $\lambda \in \{\alpha_i, \beta_i, \gamma_i\}$. Again, states are partitioned into action and navigation states, this time navigation states are either universal or existential (indicating whether they are part of a $\Diamond$ transition of $\mathcal{W}$ or a $\Box$ transition of $\mathcal{W}$). In order to simulate a transition $\Diamond w'$, existential navigation states are used. The automaton $\mathcal{A}$ guesses a transition $\langle q, \alpha_i, \beta_i, \gamma_i, q'\rangle \in T$, it spawns an existential navigation state that contains the initial state of $\mathcal{U}_{\alpha_i}$. The navigation phase continues by emulating the run of $\mathcal{U}_{\alpha_i}$ while going up the tree (towards the root). Once an accepting state of $\mathcal{U}_{\alpha_i}$ is reached, $\mathcal{A}$ spawns an extra navigation process that is in charge of going to the root and ensuring that the current location is a member in β_i (that is, spawn a navigation state with a state of $\mathcal{U}_{\beta_i}$). Simultaneously, $\mathcal{A}$ proceeds with a navigation state that contains a state of $\mathcal{U}_{\gamma_i}$. It emulates a run of $\mathcal{U}_{\gamma_i}$ backwards and guesses a word in γ_i. In order to simulate a transition $\Box w'$, universal navigation states are used. In order to check all possible successors of the confi guration (q, x), the behavior of universal navigation states is dual. The full proof of Theorem 7 is in Appendix B.2.

The constructions in Theorems 6 and 7 reduce the global model-checking problem to the global membership problem of a 2APT. By Theorem 5, we then have the following.

Theorem 8. *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 $\mathcal{W}$.*

Together with Theorem 3, 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 prefi x-recognizable rewrite systems stems from the different defi nition of $\|T\|$ in the two cases.

5 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[8]. 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 measure the size of a 2NBP by two parameters, the number of states and the size, $|\delta| = \Sigma_{p \in P} \Sigma_{a \in \Sigma} |\delta(s, a)|$, of the transition function.

We studied in [KPV02] the emptiness and membership problems for 2NBP. Here, we consider the global membership problem of 2NBP. We show that the reduction used in [KPV02] from the membership problem of 2NBP to the emptiness problem of ABW, can be used to construct an NFW N that accepts the 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 sink of N) iff $\mathcal{S}$ accepts $\langle \Upsilon^*, \tau \rangle$ from (q, w).

Theorem 9. *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 $O(|P|^2 \cdot \|\tau\|)$.*

The first thing that we do is slightly modify the 2NBP. We add an 'idle' state, in which the automaton starts its run from the root. In this idle state, the automaton navigates to some arbitrary node of the tree. Then, the automaton transitions to an arbitrary state and starts a 'normal' run. The 'idle' state masks the fact that we would like to identify all the pairs (q, w) from which the tree is accepted. Thus, the new automaton $\mathcal{S}'$ navigates to the node w in the idle state and then transitions into state q. If (q, w) is accepted the sequence leading to the idle state to w can be prolonged into a full accepting run.

We showed in [KPV02] how to construct an ABW A that is not empty iff $\mathcal{S}'$ accepts the tree T. In the proof, we translate an accepting run of A on a^ω into an accepting run of $\mathcal{S}'$ on T and vice versa. Thus, there is a 1-1 and onto correspondence between runs of A on a^ω and runs of $\mathcal{S}'$ on T. We extract from the emptiness information on A the pairs (q, w) such that $\mathcal{S}^q$ accepts the tree from node w. The full proof of Theorem 9, which is rather involved, is in Appendix D.

6 Global Linear Time Model Checking

In this section we solve the global model-checking for 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 simply reduce linear time model-checking to branching time model-checking. We have to develop methods specifically for linear time. We solve the global model-checking problem by a reduction to the global membership problem of 2NBP. Again, the main difference from the construction in [KPV02] is the usage of the global-membership problem of 2NPT.

As in the previous section, 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. As before,

[8] There is a similar type of automata called *Tree Walking Automata*. These are automata that read finite trees and expect the nodes of the tree to be labeled by the direction and by the set of successors of the node. Tree walking automata are used in XML queries. See [EHvB99,Nev02].

for pushdown systems it is sufficient to label a node in the tree by its direction. For prefix-recognizable systems the label is more complex and reflects the membership of x in the regular expressions that are used in the transition rules.

In order to handle pushdown systems we use again the tree $\langle V^*, \tau_V \rangle$. We construct a 2NBP $\mathcal{S}$ that reads $\langle V^*, \tau_V \rangle$. The state space of $\mathcal{S}$ contains a component that memorizes the current state of the rewrite system. The location of the reading head in $\langle V^*, \tau_V \rangle$ represents the store of the current configuration. Thus, in order to know which rewrite rules can be applied, $\mathcal{S}$ consults its current state and the label of the node it reads.

Theorem 10. *Given a pushdown system $R = \langle \Sigma, V, Q, L, T \rangle$ and an NBW $N = \langle \Sigma, W, w_0, \eta, F \rangle$, there is 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 $\langle V^*, \tau_V \rangle$ from (s, x) iff $G_R^{(f(s),x)} \models N$. The automaton $\mathcal{S}$ has $O(|Q| \cdot \|T\| \cdot |N|)$ states and the size of its transition function is $O(\|T\| \cdot |N|)$.*

The full proof is in Appendix C.1.

We now turn to consider prefix-recognizable systems. Again the configuration of a prefix-recognizable system $R = \langle \Sigma, V, Q, L, T \rangle$ consists of a state in Q and a word in V^*. So, the store content is still a node in the tree V^*. However, in order to apply a rewrite rule it is not enough to know the direction of the node. Recall that in order to represent the configuration $(q, x) \in Q \times V^*$, our 2NBP memorizes the state q as part of its state space and it reads the node $x \in V^*$. In order to apply the rewrite rule $t_i = \langle q, \alpha_i, \beta_i, \gamma_i, q' \rangle$, the 2NBP has to go up the tree along a word $y \in \alpha_i$. Then, if $x = y \cdot z$, it has to check that $z \in \beta_i$, and finally guess a word $y' \in \gamma_i$ and go downwards y' to $y' \cdot z$. Finding a prefix y of x such that $y \in \alpha_i$, and a new word $y' \in \gamma_i$ is done as in the case of branching time by emulating the automata $\mathcal{U}_{\alpha_i}$ and $\mathcal{U}_{\gamma_i}$. How can the 2NBP know that $z \in \beta_i$? Instead of labeling each node $x \in V^*$ only by its direction, we can label it also by the regular expressions β for which $x \in \beta$. Thus, when the 2NBP runs $\mathcal{U}_{\alpha_i}$ up the tree, it can tell, in every node it visits, whether z is a member of β_i or not. If $z \in \beta_i$, the 2NBP may guess that time has come to guess a word in γ_i and run $\mathcal{U}_{\gamma_i}$ down the guessed word.

Thus, in the case of prefix-recognizable systems, the nodes of the tree whose membership is checked are labeled by both their directions and information about the regular expressions β. We denote this tree by $\langle V^*, \tau_\beta \rangle$ and give its full definition in Appendix C.2.

Theorem 11. *Given a prefix-recognizable system $R = \langle \Sigma, V, Q, L, T \rangle$ and an NBW $N = \langle \Sigma, W, w_0, \eta, F \rangle$, there is 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 $\langle V^*, \tau_\beta \rangle$ from (s, x) iff $G_R^{(f(s),x)} \models N$. The automaton $\mathcal{S}$ has $O(|Q| \cdot (|Q_\alpha| + |Q_\gamma|) \cdot |T| \cdot |N|)$ states and the size of its transition function is $O(\|T\| \cdot |N|)$.*

The proof is in Appendix C.2. Combining Theorems 9, 10 and 11, we get the following.

Theorem 12. *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.*
- *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.

7 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]. We match the complexity bounds of previous algorithms for global model-checking [EHRS00,EKS01,KPV02,Cac02b] 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. Recently, Alur et al. suggested the logic CARET, that can specify non-regular properties [AEM04]. Our algorithm generalizes to CARET specifications as well.

References

[AEM04] R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *10th TACAS*, LNCS. Springer. 2004, to appear.

[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.

[BCMS00] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. Unpublished, 2000.

[BE96] O. Burkart and J. Esparza. More infinite results. *ENTCS*, 6, 1996.

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

[BQ96] O. Burkart and Y.-M. Quemener. Model checking of infinite graphs defined by graph grammers. 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.

[BR01] T. Ball and S. Rajamani. The SLAM toolkit. In *13th CAV*, *LNCS* 2102, 260–264, 2001.

[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. Comut.*, 2:89–125, 1995.

[BS99] O. Burkart and B. Steffen. Model checking the full modal μ-calculus for infinite sequential processes. *TCS*, 221:251–270, 1999.

[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.

[Bur97a] O. Burkart. Automatic verification of sequential infinite-state processes. In *LNCS* 1354. Springer, 1997.

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

[Cac02a] T. Cachat. Two-way tree automata solving pushdown games. In *Automata Logics, and Infinite Games*, *LNCS* 2500, 303–317. Springer, 2002.

[Cac02b] 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):244–263, 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.

[CKV01] H. Chockler, O. Kupferman, and M.Y. Vardi. Coverage metrics for temporal logic model checking. In *7th TACAS*, *LNCS* 2031, 528 – 542. 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.

[EHvB99] J. Engelfriet, H.J. Hoggeboom, and J.-P van Best. Trips on trees. *Acta Cybernetica*, 14:51–64, 1999.

[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.

[FWW97] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown automata. In *2nd Infi nity*, 1997.

[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 Verifi cation 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.

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

[Nev02] F. Neven. Automata, logic, and XML. In *16th CSL*, *LNCS* 2471, 2–26, 2002. Springer.

[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.

[Sch02] S. Schwoon. *Model-checking pushdown systems*. PhD thesis, TU M¨unchen, 2002.

[Sha00] N. Shankar. Combining theorem proving and model checking through symbolic analysis. In *11th Concur*, *LNCS* 1877, 1–16, 2000. Springer.

[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*, *LNCS* 1443, 628–641. Springer, 1998.

[VW86] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *JCSS*, 32(2):182–221, 1986.

[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*, *LNCS* 1102, 62–74. Springer, 1996.

[Wil99] T. Wilke. CTL$^+$ is exponentially more succinct than CTL. In *19th FSTTCS*, *LNCS* 1738, 110–121. Springer, 1999.

[WVS83] P. Wolper, M.Y. Vardi, and A.P. Sistla. Reasoning about infinite computation paths. In *24th FOCS*, 185–194, 1983.

A Proof of Lemma 1

Lemma 1. *A word $w \in \Upsilon^*$ is accepted by N in a state $s \in S$ iff $\mathcal{A}$ accepts T from (w, s).*

Proof. Given a node $w \in \Upsilon^*$ and a state $s \in S$ let the tree T^s_w be the unique $(\Sigma \times S_+)$-labeled tree whose projection on Σ is T and its unique node labeled by a state in S is w that is labeled by s.

Suppose that N accepts w with the run $r = ((q_0, \perp), p_0), \ldots, ((q_n, \perp), p_n), s$ that ends in s. We construct an accepting run tree $r' : \Upsilon^* \to R$ of $\mathcal{A}''$ on T^s_w. Let $r'(\epsilon) = ((q_0, \perp), p_0)$. Clearly, $r'(\epsilon) = r_0$. Continue by induction the run r' from a node $x \in \Upsilon^*$ labeled by $((q_i, \perp), p_i)$. From the definition of N it follows that for every two adjacent states in r, $((q_i, \perp), p_i), ((q_{i+1}, \perp), p_{i+1})$ the transition $\delta(((q_i, \perp), p_i), (\sigma, \perp))$ of $\mathcal{A}''$ contains a tuple $\langle$ ($(q^{i+1}_1, \top)$, p^{i+1}_1), ..., $((q^{i+1}_j, \perp), p^{i+1}_j)$, ..., $((q^{i+1}_k, \top), p^{i+1}_k)$ $\rangle$ such that $\sigma = L(q_i)$, $q^{i+1}_j = q_{i+1}$, $p^{i+1}_j = p_{i+1}$ and for every l we have that the language of $((q^{i+1}_l, \alpha), p^{i+1}_l)$ is not empty. For $l \neq j$ we add some accepting run tree of $((q^{i+1}_l, \top), p^{i+1}_l)$ under $x \cdot \upsilon_l$. We label $x \cdot \upsilon_j$ by $((q_{i+1}, \perp), p_{i+1})$. Similarly, when we reach the end of the run of N, the transition $\delta(((q_n, \perp), p_n), (L(q), s))$ contains a tuple $\langle$ $((q^{n+1}_1, \top), p^{n+1}_1)$, ..., $((q^{n+1}_k, \top), p^{n+1}_k)$ $\rangle$ such that for every state in the tuple its language is not empty. We now add a complete accepting run tree below every successor of the node x and complete the accepting run r' of $\mathcal{A}''$. It follows from the definition of $\mathcal{A}''$ that $\mathcal{A}$ accepts T from (s, w).

Suppose that $\mathcal{A}$ accepts T from (s, w) then we conclude that $\mathcal{A}''$ accepts T^s_w and from the accepting run of $\mathcal{A}''$ we construct an accepting run of N on w that ends in state s.

B Reductions from Branching Time Model Checking

B.1 Pushdown Systems

Theorem 6. *Given a pushdown system $R = \langle \Sigma, V, Q, L, T\rangle$ and a graph automaton $\mathcal{W} = \langle \Sigma, W, w_0, \delta, F\rangle$, there is 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 $G^{(f(p),x)}_R \models \mathcal{W}$. The automaton $\mathcal{A}$ has $O(|Q| \cdot |R| \cdot |V|)$ states, and has the same index as $\mathcal{W}$.*

Proof. Let $V = \{A_1, \ldots, A_n\}$ and $\Sigma = V \cup \{\perp\}$. Recall that in order to apply a rewrite rule of a pushdown system from configuration (q, x), it is sufficient to know q and the first letter of x. Let $\langle V^*, \tau_V\rangle$ be the V-labeled V-tree such that for every $x \in V^*$ we have $\tau_V(x) = dir(x)$. Note that $\langle V^*, \tau_V\rangle$ is a regular tree of size $|V| + 1$. We define $\mathcal{A} = \langle V, P, \eta, p_0, \alpha\rangle$ as follows.

- $P = (W \times Q \times \mathit{tails}(T))$, where $\mathit{tails}(T) \subseteq V^*$ is the set of all suffixes of words $x \in V^*$ for which there are states $q, q' \in Q$ and $A \in V$ such that $\langle q, A, x, q'\rangle \in T$. Intuitively, when $\mathcal{A}$ visits a node $x \in V^*$ in state $\langle w, q, y\rangle$, it checks that G_R with initial state $(q, y \cdot x)$ is accepted by $\mathcal{W}^w$. In particular, when $y = \varepsilon$, then G_R with initial state (q, x) (the node currently being visited) needs to be accepted by $\mathcal{W}^s$. States of the form $\langle w, q, \varepsilon\rangle$ are called *action states*. From these states $\mathcal{A}$ consults δ and T in order to impose new requirements on the exhaustive V-tree. States of the form $\langle w, q, y\rangle$, for $y \in V^+$, are called *navigation states*. From these states $\mathcal{A}$ only navigates downwards y to reach new action states.
- In order to define $\eta : P \times \Sigma \to \mathcal{B}^+(ext(V) \times P)$, we first define the function $apply_T$: $\Delta \times W \times Q \times V \to \mathcal{B}^+(ext(V) \times P)$. Intuitively, $apply_T$ transforms atoms participating in δ to a formula that describes the requirements on G_R when the rewrite rules in T are applied to words of the form $A \cdot V^*$. For $c \in \Delta$, $w \in W$, $q \in Q$, and $A \in V$ we define

$$apply_R(c, w, q, A) = \begin{cases} \langle \varepsilon, (w, q, \varepsilon)\rangle & \text{If } c = \varepsilon \\ \bigwedge_{\langle q,A,y,q'\rangle \in T} \langle \uparrow, (w, q', y)\rangle & \text{If } c = \Box \\ \bigvee_{\langle q,A,y,q'\rangle \in T} \langle \uparrow, (w, q', y)\rangle & \text{If } c = \Diamond \end{cases}$$

Note that T may contain no tuples in $\{q\} \times \{A\} \times V^* \times Q$ (that is, the transition relation of G_R may not be total). In particular, this happens when $A = \bot$ (that is, for every state $q \in Q$ the configuration the state (q, ε) of G_R has no successors). Then, we take empty conjunctions as $\mathbf{true}$, and take empty disjunctions as $\mathbf{false}$.
In order to understand the function $apply_R$, consider the case $c = \Box$. When $\mathcal{W}$ reads the configuration $(q, A \cdot x)$ of the input graph, fulfilling the atom $\Box s$ requires $\mathcal{S}$ to send copies in state w to all the successors of $(q, A \cdot x)$. The automaton $\mathcal{A}$ then sends to the node x copies that check whether all the configuration $(q', y \cdot x)$, with $\rho_R((q, A \cdot x), (q', y \cdot x))$, are accepted by $\mathcal{W}$ with initial state w.
Now, for a formula $\theta \in \mathcal{B}^+(\Delta \times W)$, the formula $apply_R(\theta, q, A) \in \mathcal{B}^+(ext(V) \times P)$ is obtained from θ by replacing an atom $\langle c, w\rangle$ by the atom $apply_R(c, w, q, A)$. We can now define η for all $A \in V \cup \{\bot\}$ as follow.

- $\eta(\langle w, q, \varepsilon\rangle, A) = apply_R(\delta(w, L(q, A)), q, A)$.
- $\eta(\langle w, q, B \cdot y\rangle, A) = (B, \langle w, q, y\rangle)$.

Thus, in action states, $\mathcal{A}$ reads the direction of the current node and applies the rewrite rules of R in order to impose new requirements according to δ. In navigation states, $\mathcal{A}$ needs to go downwards $B \cdot y$.

– F' is obtained from F by replacing each set F_i by the set $F_i \times Q \times tails(R)$.

The function f associates with state (w, q, ϵ) the state q of R. For other states, f is undefined.

B.2 Prefix-Recognizable Systems

Theorem 7. *Given a prefix-recognizable system $R = \langle \Sigma, V, Q, L, T\rangle$ and a graph automaton $\mathcal{W} = \langle \Sigma, W, w_0, \delta, F\rangle$, there is 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 $O(|Q| \cdot |R| \cdot |V|)$ states, and has the same index as $\mathcal{W}$.*

Proof. Let $Q_\Omega = Q_\alpha \cup Q_\beta \cup Q_\gamma$ be the union of all the state spaces of the automata associated with regular expressions that participate in T.

As in the case of pushdown systems, $\mathcal{A}$ uses the labels of the tree to learn the state in V^* that each node corresponds to. As there, $\mathcal{A}$ applies to the transition function δ of $\mathcal{W}$ the rewrite rules of R. Here, however, the application of the rewrite rules on atoms of the form $\Diamond w$ and $\Box w$ is more involved, and we describe it below. Assume that $\mathcal{A}$ wants to check whether $\mathcal{W}^w$ accepts $G_R^{(q,x)}$, and it wants to proceed with an atom $\Diamond w'$ in $\delta(w)$. The automaton $\mathcal{A}$ needs to check whether $\mathcal{W}^{w'}$ accepts $G_R^{(q',y)}$ for some configuration (q', y) reachable from (q, x). That is, a configuration (q', y) for which there is $\langle q, \alpha_i, \beta_i, \gamma_i, q'\rangle \in T$ and partitions $x' \cdot z$ and $y' \cdot z$, of x and y, respectively, such that x' is accepted by $\mathcal{U}_{\alpha_i}$, z is accepted by $\mathcal{U}_{\beta_i}$, and is y' accepted by $\mathcal{U}_{\gamma_i}$. The way $\mathcal{A}$ detects such a configuration (q, y) is the following. From the node x, the automaton $\mathcal{A}$ simulates the automaton $\mathcal{U}_{\alpha_i}$ upwards (that is, $\mathcal{A}$ guesses a run of $\mathcal{U}_{\alpha_i}$ on the word it reads as it proceeds on direction $\uparrow$ from x towards the root of the V-tree). Suppose that on its way up to the root, $\mathcal{A}$ gets to a state in F_{α_i} as it reads the node $z \in V^*$. This means that the word read so far is in α_i, and can serve as the prefix x' above. If this is indeed the case, then it is left to check that the word z is accepted by $\mathcal{U}_{\beta_i}$, and that there is a state that is obtained from z by prefixing it with a word $y' \in \gamma_i$ that is accepted by $\mathcal{S}^{s'}$. To check the first condition, $\mathcal{A}$ sends a copy in direction $\uparrow$ that simulates a run of $\mathcal{U}_{\beta_i}$, hoping to reach a state in F_{β_i} as it reaches the root (that is, $\mathcal{A}$ guesses a run of $\mathcal{U}_{\beta_i}$ on the word it reads as it proceeds from z up to the root of the V-tree). To check the second condition, $\mathcal{A}$ simulates the automaton $\mathcal{U}_{\gamma_i}$ backwards down the tree. A node $y' \cdot z \in V^*$ that $\mathcal{A}$ reads as it encounters the initial state $q^0_{\gamma_i}$ can serve as the node y we are after. The case for an atom $\Box w'$ is similar, only that here $\mathcal{A}$ needs to check whether $\mathcal{W}^s$ accepts $G_R^{(q,y)}$ for all configurations (q', y) reachable from x, and thus the choices made by $\mathcal{A}$ for guessing the partition $x' \cdot z$ of x and the prefix y' of y are dual.

In order to follow the above application of rewrite rules, the state space of $\mathcal{A}$ is $P = W \times Q \times T \times Q_\Omega \times \{\forall, \exists\}$. Thus, a state is a 5-tuple $p = \langle w, q, \langle q', \alpha_i, \beta_i, \gamma_i, q \rangle, s, b \rangle$, where $b \in \{\forall, \exists\}$ is the simulation mode (depending on whether we are applying R to an $\Diamond$ or an $\Box$ atom), $\langle q', \alpha_i, \beta_i, \gamma_i, q \rangle$ is the rewrite rule in T we are applying, and $s \in Q_{\alpha_i} \cup Q_{\beta_i} \cup Q_{\gamma_i}$ is the current state of the simulated automaton[9]. A state where $s = q^0_{\gamma_i}$ is an action state,where we apply R on the transitions in δ. Other states are navigation states. The formal definition of the transition function of $\mathcal{A}$ follows quite straightforwardly from the definition of the state space and the explanation above.

The acceptance condition of $\mathcal{A}$ is the adjustment of F to the new state space. That is, it is obtained from F by replacing each set F_i by the set $F_i \times Q \times T \times Q^0_\gamma \times \{\forall, \exists\}$. We add $W \times Q \times T \times (Q_\Omega \setminus Q^0_\gamma) \times \{\forall\}$ as the maximal even set and $W \times Q \times T \times (Q_\Omega \setminus Q^0_\gamma) \times \{\exists\}$ as the maximal odd set. This way, in existential mode we exclude runs in which the simulation phase continues forever while allowing them in universal mode. Indeed, as we assumed that initial states have no incoming arrows, as long as $\mathcal{A}$ does not reach the initial state of $\mathcal{U}_{\gamma_i}$ it cannot visit lower sets in the acceptance condition.

C Reductions from Linear Time Model Checking

C.1 Pushdown Systems

Theorem 10. *Given a pushdown system $R = \langle \Sigma, V, Q, L, T \rangle$ and an NBW $N = \langle \Sigma, W, w_0, \eta, F \rangle$, there is 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 $\langle V^*, \tau_V \rangle$ from (s, x) iff $G_R^{(f(s),x)} \models N$. The automaton $\mathcal{S}$ has $O(|Q| \cdot \|T\| \cdot |N|)$ states and the size of its transition function is $O(\|T\| \cdot |N|)$.*

Proof. We define $\mathcal{S} = \langle V, P, p_0, \delta, F' \rangle$, where

– $P = W \times Q \times tails(T)$. Intuitively, when $\mathcal{S}$ visits a node $x \in V^*$ in state $\langle w, q, y \rangle$, it checks that R with initial configuration $(q, y \cdot x)$ is accepted by N^w. In particular, when $y = \varepsilon$, then R with initial configuration (q, x) needs to be accepted by N^w. As before, states of the form $\langle w, q, \varepsilon \rangle$ are *action states* where $\mathcal{S}$ imposes new requirement on $\langle V^*, \tau_V \rangle$. States of the form $\langle w, q, y \rangle$, for $y \in V^+$, are *navigation states*.
– The transition function δ is defined for every state in $\langle w, q, x \rangle \in W \times Q \times tails(T)$ and letter $A \in V$ as follows.
  • $\delta(\langle w, q, \epsilon \rangle, A) = \{(\langle w', q', y \rangle, \uparrow) \ : \ w' \in \eta(w, L(q, A))$ and $\langle q, A, y, q' \rangle \in T\}$.
  • $\delta(\langle w, q, B \cdot y \rangle, A) = \{(\langle w, q, y \rangle, B)\}$.

  Thus, in action states, $\mathcal{S}$ reads the direction of the current node and applies the rewrite rules of R in order to impose new requirements according to η. In navigation states, $\mathcal{S}$ needs to go downwards $B \cdot y$, so it continues in direction B.
– $F' = \{\langle w, q, \epsilon \rangle \ : \ w \in F$ and $q \in Q\}$. Note that only action states can be accepting states of $\mathcal{S}$.

The function f associates with state (w_0, q, ϵ) of $\mathcal{S}$ the state q of R. For other states f is undefined.

Assume first that $\mathcal{S}$ accepts $\langle V^*, \tau_V \rangle$ when starting its run in state (w_0, q, ϵ) from node x. Then, there exists an accepting run $r = ((w_0, q, \epsilon), x), ((w_1, q_1, \alpha_1), x_1), \ldots$ of $\mathcal{S}$ on $\langle V^*, \tau_V \rangle$. Extract from r the subsequence $((w_0, q, \epsilon), x), ((w_{i_1}, q_{i_1}, \epsilon), x_{i_1}), \ldots$ of action states. As the run is accepting and only action states are accepting states we know that this subsequence is infinite. By the definition of δ, the sequence $(q_{i_1}, x_{i_1}), (q_{i_2}, x_{i_2}), \ldots$ corresponds to an infinite path in the graph G_R. Also, by the definition of F', the run

[9] Note that a straightforward representation of P results in $O(|Q| \cdot |T| \cdot |R| \cdot |V|)$ states. Since, however, the states of the automata for the regular expressions are disjoint, we can assume that the tuple in T that each automaton corresponds to is uniquely defined from it.

$w_0, w_{i_1}, w_{i_2}, \ldots$ is an accepting run of N on the trace of this path. Hence, G_R contains an (x, q)-trace that is accepted by N, thus $(x, q) \models N$.

Assume now that $(q, x) \models N$. Then, there exists a path $(q, x), (q_1, x_1), \ldots$ in G_R whose trace does not satisfy φ. There exists an accepting run $w_0, w_1, \ldots$ of $\mathcal{M}_{\neg\varphi}$ on this trace. The combination of the two sequence serves as the subsequence of the action states in an accepting run of $\mathcal{S}$. It is not hard to extend this subsequence to an accepting run of $\mathcal{S}$ on $\langle V^*, \tau_V \rangle$ from $((w_0, q, \epsilon), x)$.

C.2 Prefix-Recognizable Systems

Let $\{\beta_1, \ldots, \beta_n\}$ be the set of regular expressions β_i such that there is a rewrite rule $\langle q, \alpha_i, \beta_i, \gamma_i, q' \rangle \in T$. Let $\mathcal{D}_{\beta_i} = \langle V, D_{\beta_i}, q^0_{\beta_i}, \eta_{\beta_i}, F_{\beta_i} \rangle$ be the deterministic automaton for the language of β_i^R (where L^R is the reversed language of L). For a word $x \in V^*$, we denote by $\eta_{\beta_i}(x)$ the unique state that $\mathcal{D}_{\beta_i}$ reaches after reading the word x^R. Let $\Sigma = V \times \Pi_{1 \le i \le n} D_{\beta_i}$. For a letter $\sigma \in \Sigma$, let $\sigma[i]$, for $i \in \{0, \ldots n\}$, denote the i-th element in σ (that is, $\sigma[0] \in V$ and $\sigma[i] \in D_{\beta_i}$ for $i > 0$). Let $\langle V^*, \tau_\beta \rangle$ denote the Σ-labeled V-tree such that $\tau_\beta(\epsilon) = \langle \perp, q^0_{\beta_1}, \ldots, q^0_{\beta_n} \rangle$, and for every node $A \cdot x \in V^+$, we have $\tau_\beta(A \cdot x) = \langle A, \eta_{\beta_1}(A \cdot x), \ldots, \eta_{\beta_n}(A \cdot x) \rangle$. Thus, every node x is labeled by $dir(x)$ and the vector of states that each of the deterministic automata reach after reading x^R. Note that $\tau_\beta(x)[i] \in F_{\beta_i}$ iff $x^R \in \beta_i^R$ i.e. $x \in \beta_i$. Note also that $\langle V^*, \tau_\beta \rangle$ is a regular tree whose size is exponential in the sum of the lengths of the regular expressions $\beta_1, \ldots, \beta_n$.

Theorem 11. *Given a prefix-recognizable system $R = \langle \Sigma, V, Q, L, T \rangle$ and an NBW $N = \langle \Sigma, W, w_0, \eta, F \rangle$, there is 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 $\langle V^*, \tau_\beta \rangle$ from (s, x) iff $G_R^{(f(s),x)} \models N$. The automaton $\mathcal{S}$ has $O(|Q| \cdot (|Q_\alpha| + |Q_\gamma|) \cdot |T| \cdot |N|)$ states and the size of its transition function is $O(\|T\| \cdot |N|)$.*

The proof resembles the proof for pushdown systems. This time, the application of a rewrite rule $t_i = \langle q, \alpha_i, \beta_i, \gamma_i, q' \rangle$ involves an emulation of the automata $\mathcal{U}_{\alpha_i}$ (upwards) and $\mathcal{U}_{\gamma_i}$ (downwards). Accordingly, one of the components of the states of the 2NBP is a state of either $\mathcal{U}_{\alpha_i}$ or $\mathcal{U}_{\gamma_i}$. Action states are states in which this component is the initial state of $\mathcal{U}_{\gamma_i}$. From action states, the 2NBP chooses a new rewrite rule $t_{i'} = \langle q', \alpha_{i'}, \beta_{i'}, \gamma_{i'}, q'' \rangle$, and it applies it as follows. First, it enters the initial state of $\mathcal{U}_{\alpha_{i'}}$, and runs $\mathcal{U}_{\alpha_{i'}}$ up the tree until it reaches a final state. It then verifies that the current node is in the language of $\beta_{i'}$, in which case it moves to a final state of $\mathcal{U}_{\gamma_{i'}}$ and runs it backward down the tree until it reaches a new action state.

Proof. We define $\mathcal{S} = \langle \Sigma, P, p_0, \delta, F' \rangle$ as follows.

- $\Sigma = V \times \Pi_{i=1}^n D_{\beta_i}$.
- $P = \{\langle w, q, s, t_i \rangle \mid w \in W,\ q \in Q,\ t_i = \langle q', \alpha_i, \beta_i, \gamma_i, q \rangle \in T,\ \text{ and } s \in Q_{\alpha_i} \cup Q_{\gamma_i}\}$
  Thus, S holds in its state a state of N, a state in Q, the current state in Q_α or Q_γ, and the current rewrite rule being applied. A state $\langle w, q, s, \langle q', \alpha_i, \beta_i, \gamma_i, q \rangle \rangle$ is an action state if s is the initial state of $\mathcal{U}_{\gamma_i}$, that is $s = q^0_{\gamma_i}$. In action states, $\mathcal{S}$ chooses a new rewrite rule $t_{i'} = \langle q, \alpha_{i'}, \beta_{i'}, \gamma_{i'}, q' \rangle$. Then $\mathcal{S}$ updates the N component according to the current location in the tree and moves to the state $q^0_{\alpha_{i'}}$, the initial state of $\mathcal{U}_{\alpha_{i'}}$. Other states are navigation states. If $s \in Q_{\gamma_i}$ is a state in $\mathcal{U}_{\gamma_i}$ (that is not initial), then $\mathcal{S}$ chooses a direction in the tree, a predecessor of the state in Q_{γ_i} reading the chosen direction, and moves in the chosen direction. If $s \in Q_{\alpha_i}$ is a state of $\mathcal{U}_{\alpha_i}$ then $\mathcal{S}$ moves up the tree (towards the root) while updating the state of $\mathcal{U}_{\alpha_i}$. If $s \in F_{\alpha_i}$ is an accepting state of $\mathcal{U}_{\alpha_i}$ and $\tau(x)[i] \in F_{\beta_i}$ marks the current node x as a member of the language of β_i then $\mathcal{S}$ moves to an accepting state $s \in F_{\gamma_i}$ of $\mathcal{U}_{\gamma_i}$ (recall that initial states and accepting states have no incoming / outgoing edges respectively).

– The transition function δ is defined for every state in P and letter in $\Sigma = V \times \Pi_{i=1}^{n} D_{\beta_i}$ as follows.

$$\delta(\langle w,q,s,t_i\rangle,\sigma) = \begin{cases} \left\{ (\langle w,q,s',t_i\rangle,\uparrow) \;\middle|\; \begin{array}{l} t_i = \langle q',\alpha_i,\beta_i,\gamma_i,q\rangle \\ s' \in \eta_{\alpha_i}(s,\sigma[0]) \end{array} \right\} \cup \\ \left\{ (\langle w,q,s',t_i\rangle,\epsilon) \;\middle|\; \begin{array}{l} t_i = \langle q',\alpha_i,\beta_i,\gamma_i,q\rangle, \\ s \in F_{\alpha_i},\ s' \in F_{\gamma_i}, \\ \text{and } \sigma[i] \in F_{\beta_i} \end{array} \right\} & s \in Q_\alpha \\ \\ \left\{ (\langle w,q,s',t_i\rangle,B) \;\middle|\; \begin{array}{l} t_i = \langle q',\alpha_i,\beta_i,\gamma_i,q\rangle \\ s \in \eta_{\gamma_i}(s',B) \text{ and } B \in V \end{array} \right\} \cup \\ \left\{ (\langle w',q'',s',t_{i'}\rangle,\epsilon) \;\middle|\; \begin{array}{l} t_i = \langle q',\alpha_i,\beta_i,\gamma_i,q\rangle, \\ t_{i'} = \langle q,\alpha_{i'},\beta_{i'},\gamma_{i'},q''\rangle, \\ w' \in \eta(w,L(q,\sigma[0])), \\ s = q^0_{\gamma_i} \text{ and } s' = q^0_{\alpha_{i'}} \end{array} \right\} & s \in Q_\gamma \end{cases}$$

Thus, when $s \in Q_\alpha$ the 2NBP $\mathcal{S}$ either chooses a successor s' of s and goes up the tree or in case s is an accepting state of $\mathcal{U}_{\alpha_i}$ and $\sigma[i] \in F_{\beta_i}$ then $\mathcal{S}$ chooses an accepting state of $\mathcal{U}_{\gamma_i}$.
When $s \in Q_\gamma$ the 2NBP $\mathcal{S}$ either guesses a direction B and chooses a B-predecessor s' of s or in case $s = q^0_{\gamma_i}$ is the initial state of $\mathcal{U}_{\gamma_i}$, the automaton $\mathcal{S}$ updates the state of N, chooses a new rewrite rule $t_{i'} = \langle q,\alpha_{i'},\beta_{i'},\gamma_{i'},q''\rangle$ and moves to the initial state $q^0_{\alpha_{i'}}$ of $\mathcal{U}_{\alpha_{i'}}$.
– $F' = \{\langle w,q,s,t_i\rangle \mid w \in F,\ q \in Q,\ t_i = \langle q',\alpha_i,\beta_i,\gamma_i,q\rangle, \text{ and } s = q^0_{\gamma_i}\}$
Only action states may be accepting. As initial states (of $\mathcal{U}_{\gamma_i}$) have no incoming edges, in an accepting run, no navigation stage can last indefinitely.

The function f associates with state $(w_0,q,q^0_{\gamma_i},\langle q',\alpha_i,\beta_i,\gamma_i,q\rangle)$ the state q of R. For other states, f is undefined.

As before we can show that a (s,x) trace that satisfies N and the rewrite rules used to create this trace can be used to produce a run of $\mathcal{S}$ on $\langle V^*,\tau_\beta\rangle$ starting from node x in state (w_0,q,s,t_i) where $t_i = \langle q',\alpha_i,\beta_i,\gamma_i,q\rangle$ and $s = q^0_{\gamma_i}$.

Similarly, an accepting run of $\mathcal{S}$ on $\langle V^*,\tau_\beta\rangle$ starting from node x in state (w_0,q,s,t_i) where $t_i = \langle q',\alpha_i,\beta_i,\gamma_i,q\rangle$ and $s = q^0_{\gamma_i}$ is used to find a (q,x)-trace in G_R that is accepted by N.

Notice that there is some redundancy in the states of $\mathcal{S}$. If we assume that a transition $\langle q,\alpha_i,\beta_i,\gamma_i,q'\rangle$ is recognized by the states in $Q_{\alpha_i} \cup Q_{\gamma_i}$, then we can remove the T component from $\mathcal{P}$.

D The reduction from 2NBP to 1AWW

D.1 Definition of Alternating Automata on infinite words

An *alternating Büchi automaton on words* (*ABW* for short) is $A = \langle \Sigma,Q,q_0,\eta,F\rangle$ where Σ, Q, q_0, and F are as in NBW and $\eta : Q \times \Sigma \to B^+(\{0,1\} \times Q)$ is the transition function. A *run* of A on an infinite word $w = w_0 w_1 \ldots$ is a labeled $\mathbb{N}$-tree (T,r) where $r : T \to \mathbb{N} \times Q$. A node x labeled by (i,q) describes a copy of the automaton in state q reading letter w_i. The labels of a node and its successors have to satisfy the transition function η. Formally, $\epsilon \in T$ and $r(\epsilon) = (0,q_0)$ and for all nodes x with $r(x) = (i,q)$ and $\eta(q,w_i) = \theta$ there is a (possibly empty) set $\{(\Delta_1,q_1),\ldots,(\Delta_n,q_n)\} \models \theta$ such that $\{x\cdot 1,\ldots,x\cdot n\} \subseteq T$ and for every $1 \le c \le n$ we have , $r(x\cdot c) = (i+\Delta_c,q_c)$. Thus, a 0-transition leaves the automaton reading the same letter. Note that for 2NBP we call transitions that leave the automaton in the same location ϵ-transitions and for ABW we call them 0-transitions.

A run of an ABW is *accepting* if every infinite path visits the accepting set infinitely often. As before, a word w is *accepted* by A if A has an accepting run on the word. We similarly define the language $L(A)$ of A.

Again, the size of the 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 $B^+(\{0,1\}\times Q)$ we define $|(\Delta,q)| = |\mathbf{true}| = |\mathbf{false}| = 1$ and $|\theta_1 \vee \theta_2| = |\theta_1 \wedge \theta_2| = |\theta_1| + |\theta_2| + 1$.

Theorem 12. [VW86] *Given an ABW over 1-letter alphabet $A = \langle\{a\}, Q, q_0, \eta, F\rangle$ we can check whether $L(A)$ is empty in time $O(|\eta|)$ and space $O(|Q|)$.*

The emptiness algorithm can also produce a table $T : Q \to \{0,1\}$ such that $T(q) = 1$ iff $L(A^q) \neq \emptyset$. A simple extension of the algorithm can produce for a state q such that $L(A^q) \neq \emptyset$ an accepting (ultimately periodic) run of A^q on a^ω.

D.2 The proof

Theorem 9. *Consider a 2NBP $\mathcal{S} = \langle\Sigma, P, p, \delta, F\rangle$ and a regular tree $T = \langle\Upsilon^*, \tau\rangle$. 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 $\mathcal{D}_\tau = \langle\Upsilon, \Sigma, D_\tau, d^0_\tau, \rho_\tau, L_\tau\rangle$. 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 $\mathcal{A} = \langle\{a\}, Q, q_0, \eta, F'\rangle$ as follows.

- $Q = (P' \cup (P' \times P')) \times D_\tau \times \{\bot, \top\}$.
- $q_0 = \langle p_0, d^0_\tau, \bot\rangle$.
- $F' = (F \times D_\tau \times \{\bot\}) \cup (P' \times D_\tau \times \{\top\})$.

In order to define the transition function we have the following definitions. Two functions $f_\alpha : P' \times P' \to \{\bot, \top\}$ where $\alpha \in \{\bot, \top\}$, and for every state $p \in P'$ and alphabet letter $\sigma \in \Sigma$ the set C^σ_p 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. Formally

$$f_\bot(p,q) = \bot$$

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

$$C^\sigma_p = \left\{ p' \;\middle|\; \begin{array}{l} \exists s_0, s_1, \ldots, s_n \in (P')^+ \text{ such that} \\ s_0 = p',\ s_n = p, \\ \forall 0 < i < n,\ \langle\epsilon, s_i\rangle \in \delta'(s_{i-1}, \sigma), \text{ and} \\ \langle\uparrow, s_n\rangle \in \delta(s_{n-1}, \sigma) \end{array} \right\}$$

Now η is defined for every state in Q as follows.

$$\eta(p,d,\alpha) = \bigvee \begin{array}{l} \bigvee_{p'\in P'} \bigvee_{\beta\in\{\bot,\top\}} (\langle p,p',d,\beta\rangle,0) \wedge (\langle p',d,\beta\rangle,0) \\ \bigvee_{\upsilon\in\Upsilon} \bigvee_{\langle \upsilon,p'\rangle\in\delta'(p,L_\tau(d))} (\langle p',\rho_\tau(d,\upsilon),\bot\rangle,1) \\ \bigvee_{\langle \epsilon,p'\rangle\in\delta'(p,L_\tau(d))} (\langle p',d,\bot\rangle,0) \end{array}$$

$$\eta(p_1,p_2,d,\alpha) = \bigvee \begin{array}{l} \bigvee_{\langle \epsilon,p'\rangle\in\delta'(p_1,L_\tau(d))} (\langle p',p_2,d,f_\alpha(p',p_2)\rangle,0) \\ \bigvee_{p'\in P'} \bigvee_{\beta_1+\beta_2=\alpha} \begin{pmatrix} (\langle p_1,p',d,f_{\beta_1}(p_1,p')\rangle,0)\wedge \\ (\langle p',p_2,d,f_{\beta_2}(p',p_2)\rangle,0) \end{pmatrix} \\ \bigvee_{\upsilon\in\Upsilon,\langle \upsilon,p'\rangle\in\delta'(p_1,L_\tau(d))} \bigvee_{p''\in C_{p_2}^{L_\tau(d)}} (\langle p',p'',\rho_\tau(d,\upsilon),f_\alpha(p',p'')\rangle,1) \end{array}$$

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

The following claim establishes the connection between $\mathcal{A}$ and $\mathcal{S}'$.

Claim. [KPV02] $\mathcal{L}(\mathcal{A})\neq\emptyset$ iff $\langle\Upsilon^*,\tau\rangle\in\mathcal{L}(\mathcal{S})$

The proof in [KPV02] translates 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. It follows from the proof, that whenever the language of a state (p,d,α) is not empty, then there exists an accepting run of $\mathcal{S}'$ on the regular tree $\langle\Upsilon^*,\tau_d\rangle$ where τ_d is the labeling induced by the transducer $\mathcal{D}^d$. Similarly, whenever the language of a state (p_1,p_2,d,α) is not empty, then there exists a partial run of $\mathcal{S}'$ that starts and ends in the root of $\langle\Upsilon^*,\tau_d\rangle$. Furthermore, if $\alpha=\top$ then this partial run contains a state in F.

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 $O(|\delta|\cdot|P|^2\cdot\|\tau\|)$. As previously explained, from the emptiness algorithm we can get a table $T:Q\rightarrow\{0,1\}$ 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 are now ready to construct 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), $q_0=(p_0,d_\tau^0,\bot)$ is the initial state of $\mathcal{A}$, and Δ is defined as follows.

Consider a state $(p_0,d,\alpha)\in Q'$, its transition in $\mathcal{A}$ is

$$\eta(p_0,d,\alpha) = \bigvee \begin{array}{l} \bigvee_{p\in P} \bigvee_{\beta\in\{\bot,\top\}} (\langle p_0,p,d,\beta\rangle,0) \wedge (\langle p,d,\beta\rangle,0) \\ \bigvee_{\upsilon\in\Upsilon} (\langle p_0,\rho_\tau(d,\upsilon),\bot\rangle,1) \\ \bigvee_{p\in P} (\langle p,d,\bot\rangle,0) \end{array}$$

For every $\upsilon\in\Upsilon$ such that the language of$(p_0,\rho_\tau(d,\upsilon),\bot)$ is not empty, we add $(p_0,\rho_\tau(d,\upsilon),\bot)$ to $\Delta((p_0,d,\alpha),\upsilon)$. For every state p such that the language of (p_0,p,d,β) is not empty and the language of (p,d,β) is not empty, we add (p_0,p,d,β) to $\Delta((p_0,d,\alpha),\varepsilon)$. For every state $p\in P$ such that the language of $(p,d,\bot)$ is not empty, we add (the accepting state) p to $\Delta((p_0,d,\alpha),\varepsilon)$.

Consider a state $(p_0,p,d,\alpha)\in Q'$, its transition in $\mathcal{A}$ is

$$\eta(p_0,p,d,\alpha) = \bigvee \begin{array}{l} \bigvee_{p'\in P} (\langle p',p,d,f_\alpha(p',p)\rangle,0) \\ \bigvee_{p'\in P} \bigvee_{\beta_1+\beta_2=\alpha} \begin{pmatrix} (\langle p_0,p',d,f_{\beta_1}(p_0,p')\rangle,0)\wedge \\ (\langle p',p,d,f_{\beta_2}(p',p)\rangle,0) \end{pmatrix} \\ \bigvee_{\upsilon\in\Upsilon} \bigvee_{p'\in C_p^{L_\tau(d)}} (\langle p_0,p',\rho_\tau(d,\upsilon),f_\alpha(p_0,p')\rangle,1) \end{array}$$

For every $\upsilon \in \Upsilon$ and for every $p' \in C_p^{L_\tau(d)}$ such that the language of $(p_0, p', \rho_\tau(d, \upsilon), f_\alpha(p_0, p'))$ is not empty, we add $(p_0, p', \rho_\tau(d, \upsilon), f_\alpha(p_0, p'))$ to $\Delta((p_0, p', d, \alpha), \upsilon)$. For every state p' such that the language of $(p', p, d, f_\alpha(p', p))$ is not empty, we add p' to $\Delta((p_0, p, d, \alpha), \varepsilon)$. For every state p' such that the language of (p_0, p', d, β_1) is not empty and the language of (p', p, d, β_2) is not empty, we add (p_0, p', d, β_1) to $\Delta((p_0, p, d, \alpha), \varepsilon)$.

This completes the definition of the automaton. We have to show that for every word $w \in \Upsilon^*$ accepted by N in state $p \in P$ we have that $\langle \Upsilon^*, \tau\rangle$ is accepted by $\mathcal{S}$ from (s, w).

Lemma 2. *A word $w \in \Upsilon^*$ is accepted by N in a state $p \in P$ iff $\mathcal{S}$ accepts $\langle \Upsilon^*, \tau\rangle$ from (p, w).*

Proof. Consider some run $r = n_0, n_1, \ldots, n_l$ of N. Denote by $word(r, i)$ the sequence $\upsilon_1 \cdots \upsilon_m$ of letters read by N in the run $n_0, \ldots n_i$.

Suppose that N accepts w. There exists an accepting run r of N on w. The run r has the following form $r = (p_0, d_0, \alpha_0), \ldots, (p_0, d_n, \alpha_n), (p_0, p'_1, d'_1, \alpha'_1), \ldots (p_0, p'_k, d'_k, \alpha'_k), s$. It is simple to see that $w = word(r, n+k)$. We construct an accepting run of $\mathcal{S}$ on $\langle \Upsilon^*, \tau\rangle$ starting from (w, s). Consider the state $(p_0, p'_1, d'_1, \alpha'_1)$. From the definition of N it follows that the language of (p'_1, d'_1, α'_1) is not empty. Hence, there exists an accepting run tree of $\mathcal{S}$ starting from p' that accepts $\langle \Upsilon^*, \tau_{d'_1}\rangle$. We change this accepting run into an accepting run of $\mathcal{S}$ that starts from $word(r, n+1)$. This serves as the suffix of our run. Consider the transition from $(p_0, p'_i, d'_i, \alpha'_i)$ to $(p_0, p'_{i+1}, d'_{i+1}, \alpha'_{i+1})$. According to the definition of N it results from one of the following:

- The disjunct $(p_0, p'_{i+1}, d'_{i+1}, \alpha'_{i+1}) \wedge (p'_{i+1}, d'_{i+1}, p'_i, \beta)$ where $d_{i+1} = d_i$ and it is an ϵ transition.
- The disjunct $(p_0, p'_{i+1}, d'_{i+1}, \alpha'_{i+1})$ where $d'_{i+1} = \rho_\tau(d'_i, \upsilon)$, $word(r, n+i+1) = word(r, n+i) \cdot \upsilon$, $p'_{i+1} \in C_{p'_i}^{L_\tau(d)}$ and the transition reads the letter υ.

In the first case, there exists a run segment that connects p'_{i+1} to p'_i that starts and ends in the root of $\langle \Upsilon^*, \tau_{d_i}\rangle$. We change this run to start and end in $word(r, n+i)$ and add it before the current suffix of the run of $\mathcal{S}$. In the second case, we add the state p'_{i+1} reading $word(r, n+i+1)$ before the current suffix. By the fact that $p'_{i+1} \in C_{p'_i}^{L_\tau(d)}$ this is a valid transition of $\mathcal{S}$.

The last transition of r adds the initial state p before the current suffix and we are done.

In the other directions, suppose that $\mathcal{S}$ accepts T from (w, s). We construct an accepting run of $\mathcal{S}'$ that starts from the root of T by padding the run with a prefix of p_0 states. We translate this run of $\mathcal{S}'$ into an accepting run of $\mathcal{A}$ as in [KPV02]. The run of N follows the prefix of the run of $\mathcal{A}$ that contains p_0 and ends in s.