An Automata-Theoretic Approach to Reasoning about Infinite-State Systems

Orna Kupferman[1] and Moshe Y. Vardi[2*]

[1] Hebrew University, The institute of Computer Science, Jerusalem 91904, Israel
Email: `orna@cs.huji.ac.il`, URL: `http://www.cs.huji.ac.il/~orna`
[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 develop an automata-theoretic framework for reasoning about infinite-state sequential systems. 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 an alternating two-way tree automaton that navigates through the tree. As has been the case with finite-state systems, the automata-theoretic framework is quite versatile. We demonstrate it by solving several versions of the model-checking problem for μ-calculus specifications and prefix-recognizable systems, and by solving the realizability and synthesis problems for μ-calculus specifications with respect to prefix-recognizable environments.

1 Introduction

One of the most significant developments in the area of formal design verification is the discovery of algorithmic methods for verifying temporal-logic properties of *finite-state* systems [CES86,LP85,QS81,VW86]. In temporal-logic *model checking*, we verify the correctness of a finite-state system with respect to a desired behavior by checking whether a labeled state-transition graph that models the system satisfies a temporal logic formula that specifies this behavior (for a survey, see [CGP99]). Symbolic methods that enable model checking of very large state spaces, and the great ease of use of fully algorithmic methods, led to industrial acceptance of temporal model checking [BBG+94].

An important research topic over the past decade has been the application of model checking to infinite-state systems. Notable successes in this area has been the application of model checking to real-time and hybrid systems (cf. [HHWT95,LPY97]). Another active thrust of research 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

* Supported in part by NSF grant CCR-9700061, and by a grant from the Intel Corporation.

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* [BS99a,Wal96], *regular graphs* [BQ96], and *prefix-recognizable graphs* [Cau96]. The most powerful result so far is an exponential-time algorithm by Burkart for model checking formulas of the μ-calculus with respect to prefix-recognizable graphs [Bur97b]. See also [BCMS00,BE96,BEM97,BS99b,Bur97a,FWW97].

In this paper we develop an automata-theoretic framework for reasoning about infinite-state sequential systems. 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 enables the separation of the logical and the algorithmic aspects of reasoning about systems, yielding clean and asymptotically optimal algorithms. The automata-theoretic framework for reasoning about finite-state systems has proven to be very versatile. Automata are the key to techniques such as on-the-fly verification [GPVW95], and they are useful also for modular verification [KV98], partial-order verification [GW94,WW96], verification of real-time and hybrid systems [HKV96,DW99], and verification of open systems [AHK97,KV99]. Many decision and synthesis problems have automata-based solutions and no other solution for them is known [EJ88,PR89,KV00]. Automata-based methods have been implemented in industrial automated-verification tools (c.f., COSPAN [HHK96] and SPIN [Hol97,VB99]).

The automata-theoretic approach, however, has long been thought to be inapplicable for effective reasoning about infinite-state systems. The reason, essentially, lies in the fact that the automata-theoretic techniques involve constructions in which the state space of the system directly influences the state space of the automaton (e.g., when we take the product of a specification automaton with the graph that models the system). On the other hand, the automata we know to handle have finitely many states. The key insight, which enables us to overcome this difficulty, and which is implicit in all previous decidability results in the area of infinite-state sequential systems, is that in spite of the somewhat misleading terminology (e.g., "context-free graphs" and "pushdown graphs"), the classes of infinite-state graphs for which decidability is known can be described by finite-state automata. This is explained by the fact the the states of the graphs that model these systems can be viewed as nodes in an infinite tree and transitions between states can be expressed by finite-state automata. As a result, automata-theoretic techniques can be used to reason about such systems. In particular, we show that various problems related to the analysis of such systems can be reduced to the emptiness problem for *alternating two-way tree automata*, which was recently shown to be decidable in exponential time [Var98].

We first show how the automata-theoretic framework can be used to solve the μ-calculus model-checking problem with respect to context-free and prefix-recognizable systems. While our framework does not establish new complexity results for model checking of infinite-state sequential systems, it appears to be, like the automata-theoretic framework for finite-state systems, very versatile, and it has further potential applications. We demonstrate it by showing how the μ-calculus model-checking algorithm can be extended to graphs with *regular state properties*, to graphs with *regular fairness* *constraints*, to μ-calculus with *backwards modalities*, and to checking *realizability* of μ-calculus formulas with respect to infinite-state sequential environments. In each of these problems all we have to demonstrate is a (fairly simple) reduction to the emptiness problem for alternating two-way tree automata; the (exponentially) hard work is then done by the emptiness-checking algorithm.

2 Preliminaries

2.1 Labeled rewrite systems

A *labeled transition graph* is quadruple $G = \langle S, Act, \rho, s_0 \rangle$, where S is a (possibly infinite) set of states, Act is a finite set of actions, $\rho \subseteq S \times Act \times S$ is a labeled transition relation, and $s_0 \in S_0$ is an initial state. When $\rho(s, a, s')$, we say that s' is an a-*successor* of s, and s is an a-*predecessor* of s'. For a state $s \in S$, we denote by $G^s = \langle S, Act, \rho, s \rangle$, the graph G with s as its initial state. A *rewrite system* is a quadruple $\mathcal{R} = \langle V, Act, R, x_0 \rangle$, where V is a finite alphabet, Act is a finite set of actions, R maps each action a to a finite set of rewrite rules, to be defined below, and $x_0 \in V^*$ is an initial word. Intuitively, $R(a)$ describes the possible rules that can be applied by taking the action a. We consider here two types of rewrite systems. In a *context-free* rewrite system, each rewrite rule is a pair $\langle A, x \rangle \in V \times V^*$. In a *prefix-recognizable* rewrite system, each rewrite rule is a triple $\langle \alpha, \beta, \gamma \rangle$ of regular expressions over V, each defining a subset of V^*. We refer to rewrite rules in $R(a)$ as a-*rules*.

The rewrite system $\mathcal{R}$ induces the labeled transition graph $G_{\mathcal{R}} = \langle V^*, Act, \rho_{\mathcal{R}}, x_0 \rangle$, where $\langle x, a, y \rangle \in \rho_{\mathcal{R}}$ if there is a rewrite rule in $R(a)$ whose application on x results in y. Formally, if $\mathcal{R}$ is a context-free rewrite system, then $\rho_{\mathcal{R}}(A \cdot y, a, x \cdot y)$ if $\langle A, x \rangle \in R(a)$. If $\mathcal{R}$ is a prefix-recognizable system, then $\rho_{\mathcal{R}}(z \cdot y, a, x \cdot y)$ if there are regular expressions α, β, and γ such that $z \in \alpha$, $y \in \beta$, $x \in \gamma$, and $\langle \alpha, \beta, \gamma \rangle \in R(a)$. A labeled transition graph that is induced by a context-free rewrite system is called a *context-free graph*. A labeled transition system that is induced by a prefix-recognizable rewrite system is called a *prefix-recognizable graph*. Note that in order to apply an a-transition in state x of a context-free graph, we only need to match the first latter of x with the first element of an a-rule. On the other hand, in an application of an a-transition in a prefix-recognizable graph, we should find an a-rule and a partition of x to a prefix that belongs to the first element of the rule and a suffix that belongs to the second element.

Example 1. The context-free rewrite system $\langle \{A, B\}, \{a, b\}, R, A \rangle$, with $R(a) = \{\langle A, AB \rangle\}$ and $R(b) = \{\langle A, \varepsilon \rangle, \langle B, \varepsilon \rangle\}$, induces the labeled transition graph on the right.

$$\begin{tikzcd}
A \arrow[r, "a"] \arrow[d, "b"'] & AB \arrow[r, "a"] \arrow[d, "b"'] & ABB \arrow[r, "a"] \arrow[d, "b"'] & ABBB \cdots \arrow[d, "b"'] \\
\varepsilon & B \arrow[l, "b"] & BB \arrow[l, "b"] & BBB \cdots \arrow[l, "b"]
\end{tikzcd}$$

We define the *size* $|R|$ of R as the space required in order to encode the rewrite rules in R. Thus, in the case of a context-free rewrite system, $|R| = \sum_{a \in Act} \sum_{\langle A, x \rangle \in R(a)} |x|$, and in a prefix-recognizable rewrite system, $|R| = \sum_{a \in Act} \sum_{\langle \alpha, \beta, \gamma \rangle \in R(a)} |\mathcal{U}_\alpha| + |\mathcal{U}_\beta| + |\mathcal{U}_\gamma|$, where $|\mathcal{U}_r|$ is the size of a nondeterministic automaton provided for the regular expression r.

2.2 μ-calculus

The *μ-calculus* is a modal logic augmented with least and greatest fixpoint operators [Koz83]. Given a finite set Act of actions and a finite set Var of variables, a μ-calculus formula (in a positive normal form) over Act and Var is one of the following:

- $\mathbf{true}, \mathbf{false}$, or y for all $y \in Var$;
- $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, for μ-calculus formulas φ_1 and φ_2;
- $[a]\varphi$ or $\langle a \rangle \varphi$, for $a \in Act$ and a μ-calculus formula φ;
- $\mu y.\varphi$ or $\nu y.\varphi$, for $y \in Var$ and a μ-calculus formula φ.

A *sentence* is a formula that contains no free variables from Var (that is, all the variables are in a scope of some fixed-point operator). We define the semantics of μ-calculus with respect to a labeled transition graph $G = \langle S, Act, \rho, s_0 \rangle$ and a valuation $\mathcal{V} : Var \rightarrow 2^S$ for its free variables. Each formula ψ and valuation $\mathcal{V}$ then define a set $\psi^G(\mathcal{V})$ of states of G that satisfy the formula. For a valuation $\mathcal{V}$, a variable $y \in Var$, and a set $S' \subseteq S$, we denote by $\mathcal{V}[y \leftarrow S']$ the valuation obtained from $\mathcal{V}$ by assigning S' to y. The mapping ψ^G is defined inductively as follows:

- $\mathbf{true}^G(\mathcal{V}) = S$ and $\mathbf{false}^G(\mathcal{V}) = \emptyset$;
- For $y \in Var$, we have $y^G(\mathcal{V}) = \mathcal{V}(y)$;
- $(\psi_1 \wedge \psi_2)^G(\mathcal{V}) = \psi_1^G(\mathcal{V}) \cap \psi_2^G(\mathcal{V})$;
- $(\psi_1 \vee \psi_2)^G(\mathcal{V}) = \psi_1^G(\mathcal{V}) \cup \psi_2^G(\mathcal{V})$;
- $([a]\psi)^G(\mathcal{V}) = \{s \in S :$ for all s' such that $R(s,a,s'),$ we have $s' \in \psi^G(\mathcal{V})\}$;
- $(\langle a \rangle \psi)^G(\mathcal{V}) = \{s \in S :$ there is s' such that $R(s,a,s')$ and $s' \in \psi^G(\mathcal{V})\}$;
- $(\mu y.\psi)^G(\mathcal{V}) = \bigcap\{S' \subseteq S : \psi^G(\mathcal{V}[y \leftarrow S']) \subseteq S'\}$;
- $(\nu y.\psi)^G(\mathcal{V}) = \bigcup\{S' \subseteq S : S' \subseteq \psi^G(\mathcal{V}[y \leftarrow S'])\}$.

Note that ψ^G cares only about the valuation of free variables in ψ. In particular, no valuation is required for a sentence. For a state $s \in S$ and a sentence ψ, we say that ψ holds at s in G, denoted $G, s \models \psi$ iff $s \in \psi^G$. Also, $G \models \psi$ iff $G, s_0 \models \psi$.

2.3 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 reverse the standard definitions (e.g., when $\Upsilon = \{0, 1\}$, the successors of the node 0 are 00 and 10 (rather than 00 and 01)[1].

Given two finite sets Υ and Σ, a *Σ-labeled Υ-tree* is a pair $\langle T, V \rangle$ where T is an Υ-tree and $V : T \rightarrow \Sigma$ maps each node of T to a letter in Σ. When Υ and Σ are

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

not important or clear from the context, we call $\langle T, V\rangle$ a labeled tree. We say that an $((\Upsilon \cup \{\bot\}) \times \Sigma)$-labeled Υ-tree $\langle T, V\rangle$ is Υ*-exhaustive* if for every node $x \in T$, we have $V(x) \in \{dir(x)\} \times \Sigma$.

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 $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F\rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+(ext(\Upsilon) \times Q)$ is the transition function, $q_0 \in Q$ is an initial state, and F 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^*. Note that many nodes of T_r can correspond to the same node of Υ^*; there is no one-to-one correspondence between the nodes of the run and the nodes of the tree. The labels of a node and its successors have to satisfy the transition function. Formally, a run $\langle T_r, r\rangle$ is a Σ_r-labeled Γ-tree, for some set Γ of directions, where $\Sigma_r = \Upsilon^* \times Q$ and $\langle T_r, r\rangle$ satisfies the following:

1. $\varepsilon \in T_r$ and $r(\varepsilon) = (\varepsilon, q_0)$.
2. Consider $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \theta$. 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, $inf(\pi)$ contains exactly all the 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 $\mathcal{L}(\mathcal{A}) \neq \emptyset$.

Theorem 1. *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].

2.4 Alternating automata on labeled transition graphs

Consider a labeled transition graph $G = \langle S, Act, \rho, s_0 \rangle$. For the set Act of actions, let $next(Act) = \{\varepsilon\} \cup \bigcup_{a \in Act} \{[a], \langle a \rangle\}$. An alternating automaton on labeled transition graphs (*graph automaton*, for short) [JW95][2] is a tuple $\mathcal{S} = \langle Act, Q, \delta, q_0, F \rangle$, where Q, q_0, and F are as in alternating two-way automata, Act is a set of actions, and $\delta : Q \to \mathcal{B}^+(next(Act) \times Q)$ is the transition function. Intuitively, when $\mathcal{S}$ is in state q and it reads a state s of G, fulfilling an atom $\langle \langle a \rangle, t \rangle$ (or $\langle a \rangle t$, for short) requires $\mathcal{S}$ to send a copy in state t to some a-successor of s. Similarly, fulfilling an atom $[a]t$ requires $\mathcal{S}$ to send copies in state t to all the a-successors of s. Thus, like symmetric automata [DW99,Wil99], graph automata cannot distinguish between the various a-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 $G = \langle S, Act, \rho, s_0 \rangle$ 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 an arbitrary 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 $\delta(q) = \theta$. Then there is a (possibly empty) set $S \subseteq next(Act) \times Q$, such that S satisfies θ, and for all $\langle c, q' \rangle \in S$, the following hold:
   - 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 $c = [a]$, then for every a-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 $c = \langle a \rangle$, then there is an a-successor s' of s and $\gamma \in \Gamma$ such that $\gamma \cdot y \in T_r$ and $r(\gamma \cdot y) = (s', q')$.

A run $\langle T_r, r \rangle$ is *accepting* if all its infinite paths satisfy the acceptance condition. 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. We denote by $\mathcal{S}^q = \langle Act, Q, \delta, q, F \rangle$ the automaton $\mathcal{S}$ with q as its initial state.

We use graph automata as our 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. It is shown in [JW95] that graph automata are as expressive as μ-calculus. In particular, we have the following.

Theorem 2. *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.*

[2] The graph automata in [JW95] are different than these defined here, but this is only a technical difference.

3 Model Checking of Context-free Graphs

In this section we present an automata-theoretic approach to model-checking of context-free transition systems. Consider a labeled transition graph $G = \langle V^*, Act, \rho_{\mathcal{R}}, v_0\rangle$, induced by a rewrite system $\mathcal{R} = \langle V, Act, R, x_0\rangle$. Since the state space of G is the full V-tree, we can think of each transition $\langle z, a, z'\rangle \in \rho_{\mathcal{R}}$ as a "jump" that is activated by the action a from the node z of the V-tree to the node z'. Thus, if $\mathcal{R}$ is a context-free rewrite system and we are at node $A \cdot y$ of the V-tree, an application of the action a takes us to nodes $x \cdot y$, for $\langle A, x\rangle \in R(a)$. Technically, this means that we first move up to the parent y of $A \cdot y$, and then move down along x. Such a navigation through the V-tree can be easily performed by two-way automata.

Theorem 3. *Given a context-free rewrite system $\mathcal{R} = \langle V, Act, R, v_0\rangle$ and a graph automaton $\mathcal{S} = \langle Act, Q, \delta, q_0, F\rangle$, we can construct an alternating two-way parity automaton $\mathcal{A}$ over $(V \cup \{\bot\})$-labeled V-trees such that $\mathcal{L}(\mathcal{A})$ is not empty iff $G_{\mathcal{R}}$ satisfies $\mathcal{S}$. The automaton $\mathcal{A}$ has $O(|Q| \cdot |R| \cdot |V|)$ states, and has the same index as $\mathcal{S}$.*

Proof: The automaton $\mathcal{A}$ checks that the input tree is V-exhaustive (that is, each node is labeled by its direction). As such, $\mathcal{A}$ can learn from labels it reads the state in V^* that each node corresponds to. The transition function of $\mathcal{A}$ then consults the rewrite rules in R in order to transform an atom in $next(Act) \times Q$ to a chain of transitions that spread copies of $\mathcal{A}$ to the corresponding nodes of the full V-tree.

We define $\mathcal{A} = \langle V \cup \{\bot\}, Q', \eta, q'_0, F'\rangle$ as follows.

- $Q' = Q \times tails(\mathcal{R}) \times (V \cup \{\bot, \#\})$, where $tails(\mathcal{R}) \subseteq V^*$ is the set of all suffixes of words $x \in V^*$ for which there are $a \in Act$ and $A \in V$ such that $\langle A, x\rangle \in R(a)$. Intuitively, when $\mathcal{A}$ visits a node $x \in V^*$ in state $\langle q, y, A\rangle$, it checks that $G_{\mathcal{R}}$ with initial state $y \cdot x$ is accepted by $\mathcal{S}^q$. In particular, when $y = \varepsilon$, then $G_{\mathcal{R}}$ with initial state x (the node currently being visited) needs to be accepted by $\mathcal{S}^q$. In addition, if $A \neq \#$, then $\mathcal{A}$ also checks that $dir(x) = A$. States of the form $\langle q, \varepsilon, A\rangle$ are called *action states*. From these states $\mathcal{A}$ consults δ and R in order to impose new requirements on the exhaustive V-tree. States of the form $\langle q, y, A\rangle$, for $y \in V^+$, are called *navigation states*. From these states $\mathcal{A}$ only navigates downwards y to reach new action states. On its way, $\mathcal{A}$ also checks the V-exhaustiveness of the input tree.
- In order to define $\eta : Q' \times (V \cup \{\bot\}) \to \mathcal{B}^+(ext(V) \times Q')$, we first define the function $apply_R : next(Act) \times Q \times (V \cup \{\bot\}) \to \mathcal{B}^+(ext(V) \times Q')$. Intuitively, $apply_R$ transforms atoms participating in δ, together with a letter $A \in V \cup \{\bot\}$, which stands for the direction of the current node, to a formula that describes the requirements on $G_{\mathcal{R}}$ when the rewrite rules in R are applied to words of the form $A \cdot V^*$. For $c \in next(Act)$, $q \in Q$, and $A \in V \cup \{\bot\}$, we define

$$apply_R(c, q, A) = \begin{bmatrix} \langle \varepsilon, (q, \varepsilon, A)\rangle & \text{If } c = \varepsilon. \\ \bigwedge_{\langle A, y\rangle \in R(a)} \langle \uparrow, (q, y, \#)\rangle & \text{If } c = [a]. \\ \bigvee_{\langle A, y\rangle \in R(a)} \langle \uparrow, (q, y, \#)\rangle & \text{If } c = \langle a\rangle. \end{bmatrix}$$

  Note that $R(a)$ may contain no pairs in $\{A\} \times V^*$ (that is, the transition relation of $G_{\mathcal{R}}$ may not be total). In particular, this happens when $A = \bot$ (that is, the state

ε of $G_{\mathcal{R}}$ has no successors). Then, we take empty conjunctions as $\mathbf{true}$, and take empty disjunctions as $\mathbf{false}$.

In order to understand the function $apply_{\mathcal{R}}$, consider the case $c = [a]$. When $\mathcal{S}$ reads the state $A \cdot x$ of the input graph, fulfilling the atom $[a]q$ requires $\mathcal{S}$ to send copies in state q to all the a-successors of $A \cdot x$. The automaton $\mathcal{A}$ then sends to the node x copies that check whether all the states $y \cdot x$, with $\rho_{\mathcal{R}}(A \cdot x, a, y \cdot x)$, are accepted by $\mathcal{S}$ with initial state q.

Now, for a formula $\theta \in \mathcal{B}^+(next(Act) \times Q)$, the formula $apply_R(\theta, A) \in \mathcal{B}^+(ext(V) \times Q')$ is obtained from θ by replacing an atom $\langle c, q \rangle$ by the atom $apply_R(c, q, A)$. We can now define η for all $A \in V \cup \{\perp\}$ as follow.

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

Thus, in action states, $\mathcal{A}$ reads the direction of the current node and applies the rewrite rules of $\mathcal{R}$ in order to impose new requirements according to δ. In navigation states, $\mathcal{A}$ needs to go downwards $B \cdot y$ and check that the nodes it comes across on its way are labeled by their direction. For that, $\mathcal{A}$ proceeds only with the direction of the current node (maintained as the third element of the state), and sends to direction B a state whose third element is B. Note that since we reach states with $\#$ only with upward transitions, $\mathcal{A}$ visits these states only when it reads nodes x that have already been read by a copy of $\mathcal{A}$ that does check whether x is labeled by its direction.

– $q'_0 = \langle q_0, x_0, \perp \rangle$. Thus, in its initial state $\mathcal{A}$ checks that $G_{\mathcal{R}}$ with initial state x_0 is accepted by $\mathcal{S}$ with initial state q_0. It also checks that the root of the input tree is labeled with $\perp$.

– F' is obtained from F by replacing each set F_i by the set $F_i \times tails(R) \times (V \cup \{\#\})$.

$\square$

Context-free rewrite systems can be viewed as a special case of prefix-recognizable rewrite systems. In the next section we describe how to extend the construction described above to prefix-recognizable graphs, and we also analyze the complexity of the model-checking algorithm that follows for the two types of systems.

4 Model Checking of Prefix-Recognizable Graphs

In this section we extend the construction described in Section 3 to prefix-recognizable transition systems. The idea is similar: two-way automata can navigate through the full V-tree and simulate transitions in a system induced by a rewrite system by a chain of transitions in the tree. While in context-free transition systems the application of rewrite rules involved one move up the tree and then a chain of moves down, here things are a bit more involved. In order to apply a rewrite rule $\langle \alpha, \beta, \gamma \rangle$, the automaton has to move upwards along a word in α, check that the remaining word leading to the root is in β, and move downwards along a word in γ. As we explain below, $\mathcal{A}$ does so by simulating automata for the regular expressions participating in R.

Theorem 4. *Given a prefix-recognizable rewrite system $\mathcal{R} = \langle V, Act, R, v_0 \rangle$ and a graph automaton $\mathcal{S} = \langle Act, Q, \delta, q_0, F \rangle$, we can construct an alternating two-way* *parity automaton $\mathcal{A}$ over $(V \cup \{\perp\})$-labeled V-trees such that $\mathcal{L}(\mathcal{A})$ is not empty iff $G_{\mathcal{R}}$ satisfies $\mathcal{S}$. The automaton $\mathcal{A}$ has $O(|Q| \cdot |R| \cdot |V|)$ states, and has the same index as $\mathcal{S}$.*

Proof: For a regular expression α on V, let $\mathcal{U}_\alpha = \langle V, S_\alpha, M_\alpha, S^0_\alpha, F_\alpha\rangle$ be a nondeterministic word automaton with $\mathcal{L}(\mathcal{U}_\alpha) = \alpha$. Let $\Omega = \{\langle\alpha,\beta,\gamma\rangle :$ there is $a \in Act$ such that $\langle\alpha,\beta,\gamma\rangle \in R(a)\}$ be the set of all triples in $R(a)$, for some $a \in Act$, and let $S_\Omega = \bigcup_{\langle\alpha,\beta,\gamma\rangle\in\Omega} S_\alpha \cup S_\beta \cup S_\gamma$ be the union of all the state spaces of the automata associated with regular expressions that participate in R.

As in the case of context-free rewrite systems, $\mathcal{A}$ checks that the input tree is the V-exhaustive tree and then uses its labels in order to learn the state in V^* that each node corresponds to. As there, $\mathcal{A}$ applies to the transition function δ of $\mathcal{S}$ the rewrite rules of $\mathcal{R}$. Here, however, the application of the rewrite rules on atoms of the form $\langle a\rangle q$ and $[a]q$ is more involved, and we describe it below. Assume that $\mathcal{A}$ wants to check whether $\mathcal{S}^q$ accepts $G^x_{\mathcal{R}}$, and it wants to proceed with an atom $\langle a\rangle q$ in $\delta(t)$. The automaton $\mathcal{A}$ needs to check whether $\mathcal{S}^q$ accepts $G^y_{\mathcal{R}}$ for some state y reachable from x by applying an a-rule. That is, a state y for which there is $\langle\alpha,\beta,\gamma\rangle \in R(a)$ and partitions $x'\cdot z$ and $y'\cdot z$, of x and y, respectively, such that x' is accepted by $\mathcal{U}_\alpha$, z is accepted by $\mathcal{U}_\beta$, and is y' accepted by $\mathcal{U}_\gamma$. The way $\mathcal{A}$ detects such a state y is the following. From the node x, the automaton $\mathcal{A}$ simulates the automaton $\mathcal{U}_\alpha$ upwards (that is, $\mathcal{A}$ guesses a run of $\mathcal{U}_\alpha$ 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, $\mathcal{A}$ encounters a state in F_α as it reads the node $z \in V^*$. This means that the word read so far is in α, and can serve as the prefix x' above. If this is indeed the case (and $\mathcal{A}$ may also continue as if a state in F_α has not been encountered; thus guess that the word read so far is not x'), then it is left to check that the word z is accepted by $\mathcal{U}_\beta$, and that there is a state that is obtained from z by prefixing it with a word $y' \in \gamma$ that is accepted by $\mathcal{S}^q$. To check the first condition, $\mathcal{A}$ sends a copy in direction $\uparrow$ that simulates a run of $\mathcal{U}_\beta$, hoping to reach a state in F_β as it reaches the root (that is, $\mathcal{A}$ guesses a run of $\mathcal{U}_\beta$ 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 $\mathcal{U}_\gamma$ downwards. A node $y'\cdot z \in V^*$ that $\mathcal{A}$ reads as it encounters a state in F_γ can serve as the state y we are after. The case for an atom $[a]q$ is similar, only that here $\mathcal{A}$ needs to check whether $\mathcal{S}^q$ accepts $G^y_{\mathcal{R}}$ for all states y reachable from x by applying an a-rule, and thus the choices made by $\mathcal{A}$ for guessing the partition $x'\cdot z$ of x and the prefix y' of y are now treated dually.

In order to follow the above application of rewrite rules, the state space of $\mathcal{A}$ is $Q' = Q \times \Omega \times S_\Omega \times \{0,1,2,3\} \times \{\forall,\exists\} \times (V \cup \{\perp,\#\})$. Thus, a state is a 6-tuple $q' = \langle q, \langle\alpha,\beta,\gamma\rangle, s, i, b, A\rangle$, where A is the expected direction of the current node (needed in order to check the V-exhaustiveness), $i \in \{0,1,2,3\}$ is the current simulation mode (states in mode 0 are action states, where we apply $\mathcal{R}$ on the transitions in δ, and states in modes $1, 2$ and 3 are states where we simulate automata for α, β, and γ, respectively), $b \in \{\forall,\exists\}$ is the simulating mode (depending on whether we are applying $\mathcal{R}$ to an $\langle a\rangle$ or an $[a]$ atom), $\langle\alpha,\beta,\gamma\rangle$ is the rewrite rule in $R(a)$ we are applying, and s is the current state of the simulated automaton[3]. 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 $F_i \times \Omega \times S_\Omega \times \{0\} \times \{\forall, \exists\} \times (V \cup \{\bot, \#\})$. Considering only action states excludes runs in which the simulation of the automata for the regular expressions continues forever. Indeed, as long as a copy of $\mathcal{A}$ simulates an automaton $\mathcal{U}_\alpha$, $\mathcal{U}_\beta$, or $\mathcal{U}_\gamma$, it stays in simulation mode $1, 2$, or 3, respectively. □

The constructions described in Theorems 3 and 4 reduce the model-checking problem to the nonemptiness problem of an alternating two-way parity tree automaton. By Theorem 1, we then have the following.

Theorem 5. *The model-checking problem for a context-free or a prefix recognizable rewrite system $\mathcal{R} = \langle V, Act, R, v_0 \rangle$ and a graph automaton $\mathcal{S} = \langle Act, Q, \delta, q_0, F \rangle$, can be solved in time exponential in nk, where $n = |Q| \cdot |R| \cdot |V|$ and k is the index of $\mathcal{S}$.*

Together with Theorem 2, we can conclude with an EXPTIME bound also for the model-checking problem of μ-calculus formulas matching the lower bound in [Wal96]. Note that the fact the same complexity bound holds for both context-free and prefix-recognizable rewrite systems stems from the different definition of $|R|$ in the two cases.

5 Extensions

The automata-theoretic approach offers several extensions to the model-checking setting. We describe some of these extensions below.

5.1 Regular state properties

The systems we want to reason about often have, in addition to a set of actions, also a set P of *state properties*. In the case of finite-state systems, these are described by a mapping $L : S \to P$ that associates with each state of the labeled transition graph that models the system, the property that is true in it (for simplicity, we assume that exactly one property holds in each state). In our case, of infinite-state graphs induced by rewrite systems, we consider *regular state properties*, where each property $p \in P$ is associated with a regular expression $[p]$ over V, describing the set of states (words in V^*) in which p holds. Again, we assume that for each $x \in V^*$, there is a single $p \in P$ such that $x \in [p]$.

In order to specify behaviors of labeled transition graphs with regular state properties in P, we consider an extension of graph automata with the alphabet P. The transition function of an extended automaton $\mathcal{S} = \langle P, Act, Q, \delta, q_0, F \rangle$, is $\delta : Q \times P \to$

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

$\mathcal{B}^+(next(Act)\times Q)$; thus it reads from the input graph both the state properties, in order to know with which transition to proceed, and the actions, in order to know to which successors to proceed. The formal definition of a run of an extended graph automaton on a labeled transition graph with state properties is the straightforward extension of the definition given in Section 2.4 for the graph automata described there. Alternatively, one can consider a μ-calculus with both state properties and actions [Koz83]. Theorem 2 holds also for formulas in such a μ-calculus.

Having our solution to the model-checking problem based on two-way automata, it is simple to extend it to graphs and specifications with state properties. Indeed, whenever the automaton $\mathcal{A}$ from Theorems 3 and 4 reads the state $x \in V^*$ and takes a transition from an action state, it should now also guess the property p that holds in x and proceed according to the transition function of the specification automaton with input letter p. In order to check that the guess $x \in [p]$ is correct, the automaton simulates the word automaton $\mathcal{U}_{[p]}$ upwards, hoping to visit an accepting state when the root is reached. The complexity of the model-checking algorithm stays the same.

5.2 Fairness

The systems we want to reason about are often augmented with *fairness constraints*. Like state properties, we can define a *regular fairness constraint* by a regular expression α, where a computation of the labeled transition graph is fair iff it contains infinitely many states in α (this corresponds to weak fairness; other types of fairness can be defined similarly). It is easy to extend our model-checking algorithm to handle fairness (that is, let the path quantification in the specification range only on fair paths[4]): the automaton $\mathcal{A}$ can guess whether the state currently visited is in α, and then simulate the word automaton $\mathcal{U}_\alpha$ upwards, hoping to visit an accepting state when the root is reached. When $\mathcal{A}$ checks an existential property, it has to make sure that the property is satisfied along a fair path, and it is therefore required to visit infinitely many states in α. When $\mathcal{A}$ checks a universal property, it may guess that a path it follows is not fair, in which case $\mathcal{A}$ eventually always send copies that simulate the automaton for $\neg\alpha$. The complexity of the model-checking algorithm stays the same.

5.3 Backward modalities

Another extension is the treatment of specifications with *backwards modalities*. While forward modalities express weakest precondition, backward modalities express strongest postcondition, and they are very useful for reasoning about the past [LPZ85]. In order to adjust graph automata to backward reasoning, we add to $next(Act)$ the "directions" $\langle a^-\rangle$ and $[a^-]$. This enables the graph automata to move to a-predecessors of the current state. More formally, if a graph automaton reads a state x of the input graph, then fulfilling an atom $\langle a^-\rangle t$ requires $\mathcal{S}$ to send a copy in state t to some a-predecessor of x, and dually for $[a^-]t$. Theorem 2 can then be extended to μ-calculus formulas and graph automata with both forward and backward modalities [Var98].

[4] The exact semantics of *fair graph automata* as well as *fair μ-calculus* is not straightforward, as they enable cycles in which we switch between existential and universal modalities. To make our point here, it is simpler to assume, say, graph automata that correspond to CTL* formulas.

Extending our solution to graph automata with backward modalities is simple. Consider a node $x \in V^*$ in a prefix-recognizable graph. The a-predecessors of x are states y for which there is a rule $\langle \alpha, \beta, \gamma \rangle \in R(a)$ and partitions $x' \cdot z$ and $y' \cdot z$, of x and y, respectively, such that x' is accepted by $\mathcal{U}_\gamma$, z is accepted by $\mathcal{U}_\beta$, and y' is accepted by $\mathcal{U}_\alpha$. Hence, we can define a mapping R^- such that $\langle \gamma, \beta, \alpha \rangle \in R^-(a)$ iff $\langle \alpha, \beta, \gamma \rangle \in R(a)$, and handle atoms $\langle a^- \rangle t$ and $[a^-]t$ exactly as we handle $\langle a \rangle t$ and $[a]t$, only that for them we apply the rewrite rules in R^- rather than these in R. The complexity of the model-checking algorithm stays the same. Note that the simple solution relies on the fact that the structure of the rewrite rules in a prefix-recognizable rewrite system is symmetric (that is, switching α and γ results in a well-structured rule), which is not the case for context-free rewrite systems[5].

5.4 Global model checking

In the full paper we show that in addition to checking whether a system $\mathcal{R}$ satisfies a specification $\mathcal{S}$, we can compute the regular languages of all states satisfying $\mathcal{S}$, thus we solve the *global* model-checking problem. For a rewrite system $\mathcal{R}$ and a regular language L, let $post(L)$, $post^*(L)$, $pre(L)$, and $pre^*(L)$ be the sets of states in $G_\mathcal{R}$ that are immediate sucessors of the states in L, sucessors of the states in L, immediate predecessors of the states in L, and predecessors of the states in L, respectively. The predicates above can be viewed as specifications. Indeed, $post(L) = \langle + \rangle L$, $post^*(L) = \mu y.L \vee \langle + \rangle y$, $pre(L) = \langle - \rangle L$, and $pre^*(L) = \mu y.L \vee \langle - \rangle y$ (in a μ-calculus with state predicates, where $\langle + \rangle$ and $\langle - \rangle$ are the "next" and "previously" modalities). Hence, the algorithm can be used to compute successors and predecessors of regular state sets, and can be viewed as the automata-theoretic approach to the algorithms in [BEM97].

This observation is related to the work in [LS98], where bottom-up automata on finite trees are used in order to recognize sets of terms in Process Algebra. Given a term t, [LS98] shows that it is possible to define $post^*(t)$ as the solution of a regular equation. They conclude that $post * (t)$ is a regular tree language, and similarly for $post(t)$, $pre(t)$, and $pre^*(t)$.

6 Realizability and Synthesis

Given a rewrite system $\mathcal{R} = \langle V, Act, R, v_0 \rangle$, a *strategy* of $\mathcal{R}$ is a function $f : V^* \to Act$. The function f restricts the graph $G_\mathcal{R}$ so that from a state $x \in V^*$, only $f(x)$-actions are taken. Formally, $\mathcal{R}$ and f together define the graph $G_{\mathcal{R},f} = \langle V^*, Act, \rho, v_0 \rangle$, where $\rho(x, a, y)$ iff $f(x) = a$ and $\rho_\mathcal{R}(x, a, y)$. Given $\mathcal{R}$ and a graph automaton $\mathcal{S} = \langle Act, Q, \delta, q_0, F \rangle$, we say that a strategy f of $\mathcal{R}$ is *winning* for $\mathcal{S}$ iff $G_{\mathcal{R},f}$ satisfies $\mathcal{S}$. Given $\mathcal{R}$ and $\mathcal{S}$, the problem of *realizability* is to determine whether there is a winning strategy of $\mathcal{R}$ for $\mathcal{S}$. The problem of *synthesis* is then to construct such a strategy.

[5] Note that this does not mean we cannot model check specifications with backwards modalities in context-free rewrite systems. It just mean that doing so invovles rewrite rules that are no longer context free. Indeed, a rule $\langle A, x \rangle \in R(a)$ in a context-free system corresponds to the rule $\langle A, V^*, x \rangle \in R(a)$ in a prefix recognizable system, inducing the rule $\langle x, V^*, A \rangle \in R^{-1}(a)$.

The setting described here corresponds to the case where the system needs to satisfy a specification with respect to environments modeled by a rewrite system. Then, at each state, the system chooses the action to proceed with and the environment provides the rules that determine the successors of the state. Branching-time realizability of finite-state systems can be viewed as a special case of our setting here, where for all actions $a \in Act$, we have $R(a) = \{\langle \varepsilon, V^*, A\rangle\}$. Thus, from each state $x \in V^*$, we can apply an a-transitions to all the states $A \cdot x$, for $A \in V$.

The automaton $\mathcal{A}$ from Theorem 4 can be modified to solve the realizability problem and to generate winning strategies. The idea is simple: a strategy $f : V^* \to Act$ can be viewed as an Act-labeled V-tree. Thus, the realizability problem can be viewed as the problem of determining whether we can augment the labels of the V-labeled V-exhaustive tree by elements in Act, and accept the augmented tree in a run of $\mathcal{A}$ in which whenever $\mathcal{A}$ reads an action $a \in Act$, it applies to the transition function of the specification graph automaton only rewrite rules in $R(a)$. Hence the following theorem.

Theorem 6. *Given a prefix-recognizable rewrite system $\mathcal{R} = \langle V, Act, R, v_0\rangle$ and a graph automaton $\mathcal{S} = \langle Act, Q, \delta, q_0, F\rangle$, we can construct an alternating two-way parity automaton $\mathcal{A}$ over $((V \cup \{\bot\}) \times Act)$-labeled V-trees such that $\mathcal{L}(\mathcal{A})$ contains exactly all the V-exhaustive trees whose projection on Act is a winning strategy of $\mathcal{R}$ for $\mathcal{S}$. The automaton $\mathcal{A}$ has $O(|Q| \cdot |R| \cdot |V|)$ states, and has the same index as $\mathcal{S}$.*

Proof: Exactly as in Theorem 4, only that from an action state we proceed with the rules in $R(a)$, where a is the Act-element of the letter we read. For example, in the case of a context-free rewrite system, we would have, for $c \in next(Act)$, $q \in Q$, $A \in V$, and $a \in Act$ (the new parameter to $apply_{\mathcal{R}}$, which is read from the input tree),

$$apply_R(c, q, A, a) = \begin{cases} \langle \varepsilon, (q, \varepsilon, A)\rangle & \text{If } c = \varepsilon. \\ \bigwedge_{\langle A,y\rangle \in R(a)} \langle \uparrow, (q, y, \#)\rangle & \text{If } c = [a]. \\ \mathbf{true} & \text{If } c = [b], \text{for } b \neq a. \\ \bigvee_{\langle A,y\rangle \in R(a)} \langle \uparrow, (q, y, \#)\rangle & \text{If } c = \langle a\rangle. \\ \mathbf{false} & \text{If } c = \langle b\rangle, \text{for } b \neq a. \end{cases}$$

□

Let $n = |Q| \cdot |R| \cdot |V|$, let k be the index of $\mathcal{S}$, and let $\Sigma = (V \cup \{\bot\}) \times Act$. By Theorem 1, we can transform $\mathcal{A}$ to a nondeterministic one-way parity tree automaton $\mathcal{A}'$ with $2^{O(nk)}$ states and index $O(nk)$. By [Rab69,Eme85], if $\mathcal{A}'$ is nonempty, there exists a Σ-labeled V-tree $\langle V^*, f\rangle$ such that for all $\sigma \in \Sigma$, the set X_σ of nodes $x \in V^*$ for which $f(x) = \sigma$ is a regular set. Moreover, the nonemptiness algorithm of $\mathcal{A}'$, which runs in time exponential in nk, can be easily extended to construct, within the same complexity, a deterministic word automaton $\mathcal{U}_{\mathcal{A}}$ over V such that each state of $\mathcal{U}_{\mathcal{A}}$ is labeled by a letter $\sigma \in \Sigma$, and for all $x \in V^*$, we have $f(x) = \sigma$ iff the state of $\mathcal{U}_{\mathcal{A}}$ that is reached by following the word x is labeled by σ. The automaton $\mathcal{U}_{\mathcal{A}}$ is then the answer to the synthesis problem.

The construction described in Theorems 3 and 4 implies that the realizability and synthesis problem is in EXPTIME. Thus, it is not harder than in the satisfiability problem for the μ-calculus, and it matches the known lower bound [FL79]. Formally, we have the following.

Theorem 7. *The realizability and synthesis problems for a context-free or a prefix recognizable rewrite system* $\mathcal{R} = \langle V, Act, R, v_0 \rangle$ *and a graph automaton* $\mathcal{S} = \langle Act, Q, \delta, q_0, F \rangle$*, can be solved in time exponential in* nk*, where* $n = |Q| \cdot |R| \cdot |V|$*, and* k *is the index of* $\mathcal{S}$.

By Theorem 2, if the specification is given by a μ-calculus formula ψ, the bound is the same, with $n = |\psi| \cdot |R| \cdot |V|$, and k being the alternation depth of ψ.

7 Discussion

The automata-theoretic approach has long been thought to be inapplicable for effective reasoning about infinite-state systems. We showed that infinite-state systems for which decidability is known can be described by finite-state automata, and therefore, the states and transitions of such systems can be viewed as nodes in an infinite tree and transitions between states can be expressed by finite-state automata. As a result, automata-theoretic techniques can be used to reason about such systems. In particular, we showed that various problems related to the analysis of such systems can be reduced to the emptiness problem for alternating two-way tree automata. Our framework achieves the same complexity bounds of known model-checking algorithms, and it enables several extensions, such as treatment of state properties, fairness constraints, backwards modalities, and global model checking. Our framework also provides a solution to the realizability problem.

An interesting open problem is the extension of our framework to the linear paradigm. Since LTL formulas can be translated to automata, a simple extension of our framework to handle specifications in LTL is possible. Nevertheless, since our algorithm involves a translation of a two-way alternating automaton to a nondeterministic automaton, we would end up in a complexity that is at least exponential in the system, which is worst than known polynomial algorithms [EHRS00].

References

[AHK97] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proc. 38th FOCS*, pp. 100–109, October 1997.

[BBG+94] I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, and M. Yoeli. Methodology and system for practical formal verification of reactive hardware. In *Proc. 6th CAV*, *LNCS* 818, pp. 182–193, June 1994.

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

[BE96] O. Burkart and J. Esparza. More infinite results. *Electronic Notes in Theoretical Computer Science*, 6, 1996.

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

[BQ96] O. Burkart and Y.-M. Quemener. Model checking of infinite graphs defined by graph grammers. In *Proc. 1st International workshop on verification of infinite states systems*, volume 6 of *ENTCS*, page 15. Elsevier, 1996.

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

[BS99a] O. Burkart and B. Steffen. Composition, decomposition and model checking of pushdown processes. *Nordic J. Comut.*, 2:89–125, 1999.

[BS99b] O. Burkart and B. Steffen. Model checking the full modal μ-calculus for infinite sequential processes. *Theoretical Computer Science*, 221:251–270, 1999.

[Bur97a] O. Burkart. Automatic Verification of Sequential Infinite-State Processes. *LNCS* 1354, 1997.

[Bur97b] O. Burkart. Model checking rationally restricted right closures of recognizable graphs. In *Proc. 2nd International Workshop on Verification of Infinite State Systems*, 1997.

[Cau96] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *Proc. 23st ICALP*, *LNCS* 1099, pp. 194–205, 1996.

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

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

[DW99] M. Dickhfer and T. Wilke. Timed alternating tree automata: the automata-theoretic solution to the TCTL model checking problem. In *Proc. 26th ICALP*, *LNCS* 1644, pp. 281–290, 1999.

[EHRS00] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proc. 12th CAV*, 2000.

[EJ88] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th FOCS*, pp. 328–337, October 1988.

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

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

[Eme85] E.A. Emerson. Automata, tableaux, and temporal logics. In *Proc. Workshop on Logic of Programs*, *LNCS* 193, pp. 79–87, 1985.

[FL79] M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and Systems Sciences*, 18:194–211, 1979.

[FWW97] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown automata. In *Proc. 2nd International Workshop on Verification of Infinite State Systems*, 1997.

[GPVW95] R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification, Testing, and Verification*, pp. 3–18. Chapman & Hall, August 1995.

[GW94] P. Godefroid and P. Wolper. A partial approach to model checking. *Information and Computation*, 110(2):305–326, May 1994.

[HHK96] R.H. Hardin, Z. Har'el, and R.P. Kurshan. COSPAN. In *Computer Aided Verification, Proc. 8th Int. Conference*, *LNCS* 1102, pp. 423–427, 1996.

[HHWT95] T.A. Henzinger, P.-H Ho, and H. Wong-Toi. A user guide to HYTECH. In *Proc. TACAS*, *LNCS* 1019, pp. 41–71, 1995.

[HKV96] T.A. Henzinger, O. Kupferman, and M.Y. Vardi. A space-efficient on-the-fly algorithm for real-time model checking. In *Proc. 7th CONCUR*, *LNCS* 1119, pp. 514–529, August 1996.

[Hol97] G.J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, May 1997. Special issue on Formal Methods in Software Practice.

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

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

[Kur94] R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton Univ. Press, 1994.

[KV98] O. Kupferman and M.Y. Vardi. Modular model checking. In *Proc. Compositionality Workshop*, *LNCS* 1536, pp. 381–401, 1998.

[KV99] O. Kupferman and M.Y. Vardi. Robust satisfaction. In *Proc. 10th CONCUR*, *LNCS* 1664, pp. 383–398, 1999.

[KV00] O. Kupferman and M.Y. Vardi. Synthesis with incomplete informatio. In *Advances in Temporal Logic*, pp. 109–127. Kluwer Academic Publishers, January 2000.

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

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

[LPY97] K. G. Larsen, P. Petterson, and W. Yi. UPPAAL: Status & developments. In *Proc. 9th CAV LNCS* 1254, pp. 456–459, 1997.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Logics of Programs*, *LNCS* 193, pp. 196–218, Brooklyn, June 1985.

[LS98] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. In *Proc. 9th CONCUR*, *LNCS* 1466, pp. 50–66, September 1998.

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

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

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pp. 179–190, January 1989.

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

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

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

[VB99] W. Visser and H. Barringer. CTL* model checking for SPIN. In *Software Tools for Technology Transfer*, *LNCS*, 1999.

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

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.

[Wal96] I. Walukiewicz. On completeness of μ-calculus. In *Proc. 8th CAV*, volume 1102 of *LNCS*, pp. 62–74, 1996.

[Wil99] T. Wilke. CTL$^+$ is exponentially more succinct than CTL. In *Proc. 19th Conference on Foundations of Software Technology and Theoretical Computer Science*, *LNCS* 1738, pp. 110–121, 1999.

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

[WW96] B. Willems and P. Wolper. Partial-order methods for model checking: From linear time to branching time. In *Proc. 11th Symp LICS*, pp. 294–303, July 1996.