Model Checking Linear Properties of Prefix-Recognizable Systems

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

[1] Hebrew University, School of Engineering and Computer Science, Jerusalem 91904, Israel
Email: orna@cs.huji.ac.il, URL: http://www.cs.huji.ac.il/~orna

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

[3] Rice University, Department of Computer Science, Houston, TX 77251-1892, U.S.A.
Email: vardi@cs.rice.edu, URL: http://www.cs.rice.edu/~vardi

Abstract. We develop an automata-theoretic framework for reasoning about *linear properties of 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 automaton that navigates through the tree. We introduce *path automata on trees*. The input to a path automaton is a tree, but the automaton cannot split to copies and it can read only a single path of the tree. In particular, *two-way* nondeterministic path automata enable exactly the type of navigation that is required in order to check linear properties of infinite-state sequential systems.

We demonstrate the versatility of the automata-theoretic approach by solving several versions of the model-checking problem for LTL specifications and prefix-recognizable systems. Our algorithm is exponential in both the size of (the description of) the system and the size of the LTL specification, and we prove a matching lower bound. This is the first optimal algorithm for solving the LTL model-checking problem for prefix recognizable systems. Our framework also handles systems with regular labeling.

1 Introduction

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]). 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

* Supported in part by BSF grant 9800096.

** Supported in part by the European Commission (FET project ADVANCE, contract No IST-1999-29082). This work was carried out at the John von Neumann Minerva Center.

*** Supported in part by NSF grants CCR-9988322, IIS-9908435, IIS-9978135, and EIA-0086264, by BSF grant 9800096, and by a grant from the Intel Corporation.

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]. 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,BS99,Bur97a,FWW97].

In [KV00], Kupferman and Vardi 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 enable the separation of the logical and the algorithmic aspects of reasoning about systems, yielding clean and asymptotically optimal algorithms. Kupferman and Vardi use two-way alternating tree automata in order to reason about branching 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 branching temporal property can then be done by an alternating two-way automaton. The two-way alternating automaton starts checking the input tree from the root. It then spawns several copies of itself that may go in different directions in the tree. Each new copy can spawn other new copies and so on. The automaton accepts the input tree if all spawned copies agree on acceptance. Thus, copies of the alternating automaton navigate through the tree and check the branching temporal property. The method in [KV00] handles prefix-recognizable systems, and properties specified in the μ-calculus. The method appears to be very versatile, and it has further applications: the μ-calculus model-checking algorithm can be easily extended to graphs with *regular labeling* (that is, graphs in which each atomic proposition p has a regular expression describing the set of states in which p holds) and *regular fairness constraints*, to μ-calculus with *backward modalities*, to checking *realizability* of μ-calculus formulas with respect to infinite-state sequential environments, and to computing the set pre^* ($post^*$) of predecessors (successors) of a regular set of states. All the above are achieved using a reduction to the emptiness problem for alternating two-way tree automata where the location of the alternating automaton on the infinite tree indicates the contents of the pushdown store.

The μ-calculus is sufficiently strong to express all properties expressible in the linear temporal logic LTL (and in fact, all properties expressible by an ω-regular language) [Dam94]. Thus, the framework in [KV00] can be used in order to solve the problem of LTL model-checking for prefix-recognizable systems. The solution, however, is not optimal. This has to do both with the fact that the translation of LTL to the μ-calculus is exponential, as well as the fact that the framework in [KV00] is based on tree automata. A tree automaton splits into several copies when it runs on a tree. While splitting is essential for reasoning about branching properties, it has a computational price. For linear properties, it is sufficient to follow a single computation of the system, and tree automata seem too strong for this task. For example, while the application of the framework in [KV00] to pushdown systems and LTL properties results in a doubly-exponential algorithm, the problem is known to be EXPTIME-complete [BEM97].

In this paper, we develop an automata-theoretic framework to reason about linear properties of infinite-state sequential systems. We introduce *path automata on trees*. The input to a path automaton is a tree, but the automaton cannot split to copies and it can read only a single path of the tree. In particular, *two-way* nondeterministic path automata enable exactly the type of navigation that is required in order to check linear properties of infinite-state sequential systems. We study the expressive power and the complexity of the decision problems for (two way) path automata. The fact that path automata follow a single path in the tree makes them very similar to two-way nondeterministic automata on infinite words. This enables us to reduce the membership problem (whether an automaton accepts the tree obtained by unwinding a given finite labeled graph) of two-way nondeterministic path automata to the emptiness problem of one-way alternating weak automata on infinite words, which was studied in [KVW00]. This leads to a quadratic upper bound for the membership problem for two-way nondeterministic path automata.

As usual, the automata-theoretic framework proves to be very helpful. We are able to solve the problem of LTL model checking with respect to pushdown systems by a reduction to the membership problem of two-way nondeterministic path automata. This is in contrast to [KV00], where the emptiness problem for two-way alternating tree automata is being used. We note that both simplifications, to the membership problem vs. the emptiness problem, and to path automata vs. tree automata are crucial: as we prove, the emptiness problem for two-way nondeterministic Büchi path automata is EXPTIME-complete, and the membership problem for two-way alternating Büchi automata is also EXPTIME-complete[1]. Our automata-theoretic technique matches the known upper bound for model checking LTL properties on pushdown systems [BEM97,EHRS00]. In addition, the automata-theoretic approach provides the first solution for the case the system is prefix recognizable. Specifically, we show that we can solve the model-checking problem of an LTL formula φ with respect to a prefix-recognizable system R of size n in time and space $2^{O(n+|\varphi|)}$. We also prove a matching EXPTIME lower bound.

Our framework also handles regular labeling (in both pushdown and prefix-recognizable systems). The complexity is exponential in the nondeterministic automata that describe the labeling, matching the known bound for pushdown systems [EKS01]. The automata-theoretic techniques for handling regular labeling and for handling the regular transi-

[1] In contrast, the membership problem for one-way alternating Büchi tree automata can be solved in quadratic time. Indeed, the problem can be reduced to the emptiness problem of the 1-letter alternating word automaton obtained by taking the product of the labeled graph that models the tree with the one-way alternating tree automaton [KVW00]. This technique cannot be applied to two-way automata, since they can distinguish between a graph and its unwinding. For a related discussion regarding past-time connectives in branching temporal logics, see [KP95].

tions of a prefix-recognizable system are very similar. In both settings, the system has to be able to check the membership of the word in the store in a regular expression. This leads us to the understanding that regular labeling and prefix recognizability have exactly the same power. In the full version, we prove that LTL model checking in a prefix recognizable system and LTL model checking in a pushdown system with regular labeling are intereducible. Since the latter problem is known be EXPTIME-complete [EKS01], our reductions suggest an alternative proof of the exponential upper and lower bounds for the problem of LTL model checking in prefix-recognizable systems.

2 Preliminaries

We consider finite or infinite sequences of symbols from some finite alphabet Σ. Given a *word* $w = w_0 w_1 w_2 \cdots \in \Sigma^* \cup \Sigma^\omega$, we denote by by $w_{\geq i}$ the suffix of w starting at w_i hence $w_{\geq i} = w_i w_{i+1} w_{i+2} \cdots$. The *length* of w is denoted by $|w|$ and is defined to be ω for infinite words.

Nondeterministic automata. A *nondeterministic automaton on words* is $N = \langle \Sigma, Q, q_0, \eta, F \rangle$, where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $\eta : Q \times \Sigma \to 2^Q$ is a transition function, and $F \subseteq Q$ is a set of accepting states. We can run N either on finite words (*nondeterministic finite automaton* or *NFA* for short) or on infinite words (*nondeterministic Büchi automaton* or *NBW* for short). A *deterministic* automaton is an automaton for which $|\eta(q, a)| = 1$ for all $q \in Q$ and $a \in \Sigma$. We denote by N^q the automaton N with initial state q. A *run* of N on a finite word $w = w_0, \ldots, w_{l-1}$ is a finite sequence of states $p_0, p_1, \ldots, p_l \in Q^{l+1}$ such that $p_0 = q_0$ and for all $0 \leq j < l$, we have $p_{j+1} \in \eta(p_j, w_j)$. A run is *accepting* if $p_l \in F$. A *run* of N on an infinite word $w = w_0, w_1, \ldots$ is defined similarly as an infinite sequence. For a run $r = p_0, p_1, \ldots$, let $inf(r) = \{q \in Q \mid q = p_i \text{ for infinitely many } i\text{'s}\}$ be the set of all states occurring infinitely often in the run. A run r of an NBW is *accepting* if it visits the set F infinitely often, thus $inf(r) \cap F \neq \emptyset$. A word w is *accepted* by N if N has an accepting run on w. The *language* of N, denoted $L(N)$, is the set of words accepted by N. The size $|N|$ of a nondeterministic automaton N is the size of its transition function, thus $|N| = \Sigma_{q \in Q} \Sigma_{\sigma \in \Sigma} |\eta(q, \sigma)|$.

We are especially interested in cases where $\Sigma = 2^{AP}$, for some set AP of atomic propositions AP, and in languages $L \subseteq (2^{AP})^\omega$ definable by NBW or formulas of the linear temporal logic LTL [Pnu77]. For an LTL formula φ, the *language* of φ, denoted $L(\varphi)$, is the set of infinite words that satisfy φ.

Theorem 1. [VW94] *For every LTL formula φ, there exists an NBW N_φ with $2^{O(|\varphi|)}$ states, such that $L(N_\varphi) = L(\varphi)$.*

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* $L(s_0) \cdot L(s_1) \cdots$.

The set $\mathcal{T}_s$ is the set of all s-traces. We say that s satisfies an LTL formula φ, denoted $(G, s) \models \varphi$, iff $\mathcal{T}_s \subseteq \mathcal{L}(\varphi)$. A graph G satisfies an LTL formula φ, denoted $G \models \varphi$, iff its initial state satisfies it; that is $(G, s_0) \models \varphi$. The *model-checking problem* for a labeled transition graph G and an LTL formula φ is to determine whether G satisfies φ. Note that the transition relation need not be total. There may be finite paths but satisfaction is determined only with respect to infinite paths. In particular, if the graph has only finite paths, its set of traces is empty and the graph satisfies every LTL formula (It is also possible to consider finite paths. In this case, the NBW in Theorem 1 has to be modified so that it can recognize also finite words. Our results are easily extended to consider also finite paths).

A *rewrite system* is $R = \langle \Sigma, V, Q, L, T, q_0, x_0 \rangle$, 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, T is a finite set of rewrite rules, to be defined below, q_0 is an initial state, and $x_0 \in V^*$ is an initial word. The set of *configurations* of the system is $Q \times V^*$. Intuitively, the system has finitely many control states and 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*. A configuration $(q, x) \in Q \times V^*$ indicates that the system is in control state q with store x. 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. For uniformity, we use prefix-recognizable systems with control states.

We consider two types of labeling functions, *simple* and *regular*. The labeling function associates with a configuration $(q, x) \in Q \times V^*$ a symbol from Σ. A simple labeling function 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. A regular labeling function considers the entire word x but can only refer to its membership in some regular set. Formally, for every state q there is a partition of V^* to $|\Sigma|$ regular languages $R_1, \ldots R_{|\Sigma|}$, and $L(q, x)$ depends on the regular set that x belongs to. We are especially interested in the cases where the alphabet Σ is the powerset 2^{AP} of the set of atomic propositions. In this case, we associate with every state q and proposition p a regular language $R_{q,p}$ that contains all the words x for which the proposition p is true in configuration (q, x). Thus $p \in L(q, x)$ iff $x \in R_{q,p}$.

The rewrite system R induces the labeled transition graph $G_R = \langle \Sigma, Q \times V^*, L', \rho_R, (q_0, x_0) \rangle$. 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 $\langle q, \alpha, \beta, \gamma, q' \rangle \in T$. 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 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*. We say that a rewrite system R satisfies an LTL formula φ if $G_R \models \varphi$. [2]

Example 1. The pushdown system $\langle 2^{\{p_1,p_2\}}, \{A,B\}, \{q_0\}, L, T, q_0, A\rangle$, 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 defined by $R_{q_0,p_1} = \{A,B\}^* \cdot B \cdot B \cdot \{A,B\}^*$ and $R_{q_0,p_2} = A \cdot \{A,B\}^*$, induces the labeled transition graph below.

$$\begin{tikzcd}
p_2 & p_2 & p_1, p_2 & p_1, p_2 & \\
(q_0,\mathrm{A}) \arrow[r] \arrow[d] & (q_0,\mathrm{AB}) \arrow[r] \arrow[d] & (q_0,\mathrm{ABB}) \arrow[r] \arrow[d] & (q_0,\mathrm{ABBB}) \arrow[d] & - - - \\
(q_0,\varepsilon) & (q_0,\mathrm{B}) \arrow[l] & (q_0,\mathrm{BB}) \arrow[l] & (q_0,\mathrm{BBB}) \arrow[l] & - - - \\
 & & p_1 & p_1 &
\end{tikzcd}$$

Consider a prefix-recognizable system $R = \langle \Sigma, V, Q, L, T, q_0, x_0\rangle$. For a rewrite rule $t_i = \langle s, \alpha_i, \beta_i, \gamma_i, s'\rangle \in T$, let $\mathcal{U}_\lambda = \langle V, Q_\lambda, q^0_\lambda, \eta_\lambda, F_\lambda\rangle$, 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 assume that we have an automaton whose language is $\{x_0\}$. We denote the initial state of this automaton by x_0 and add all its states to Q_γ. Finally, for a regular labeling function L, a state $q \in Q$, and a proposition $p \in AP$, let $\mathcal{U}_{q,p} = \langle V, Q_{p,q}, q^0_{p,q}, \rho_{p,q}, F_{p,q}\rangle$ be the nondeterministic automaton for the language of $R_{q,p}$.

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, $\|T\| = \sum_{\langle q,\alpha,\beta,\gamma,q'\rangle \in T} |\mathcal{U}_\alpha| + |\mathcal{U}_\beta| + |\mathcal{U}_\gamma|$. In the case of a regular labeling function, we also measure the labeling function $\|L\| = \sum_{q \in Q} \sum_{p \in AP} |\mathcal{U}_{q,p}|$.

Theorem 2. *The model-checking problem for a pushdown system R and an LTL formula φ is solvable*

- *in time $O(\|T\|^3) \cdot 2^{O(|\varphi|)}$ and space $O(\|T\|^2) \cdot 2^{O(|\varphi|)}$ in the case that L is a simple labeling function* [EHRS00].
- *in time $O(\|T\|^3) \cdot 2^{O(\|L\|+|\varphi|)}$ and space $O(\|T\|^2) \cdot 2^{O(\|L\|+|\varphi|)}$ in the case that L is a regular labeling function. The problem is EXPTIME-hard in $\|L\|$ even for a fixed formula* [EKS01].

[2] Some work on verification of infinite-state system (e.g., [EHRS00]), consider properties given by nondeterministic Büchi word automata, rather than LTL formulas. Our algorithm actually handles properties given by automata. We translate an LTL formula to an automaton and use the automaton in our algorithm.

3 Two-way path automata on trees

Given a finite set Υ of directions, an *Υ-tree* is a set $T \subseteq \Upsilon^*$ such that if $\upsilon \cdot x \in T$, where $\upsilon \in \Upsilon$ and $x \in \Upsilon^*$, then also $x \in T$. The elements of T are called *nodes*, and the empty word ε is the *root* of T. For every $\upsilon \in \Upsilon$ and $x \in T$, the node x is the *parent* of $\upsilon \cdot x$. If $z = x \cdot y \in T$ then z is a descendant of y. Each node $x \neq \varepsilon$ of T has a *direction* in Υ. The direction of the root is the symbol $\perp$ (we assume that $\perp \notin \Upsilon$). The direction of a node $\upsilon \cdot x$ is υ. We denote by $dir(x)$ the direction of the node x. An Υ-tree T is a *full infinite tree* if $T = \Upsilon^*$. A *path* π of a tree T is an infinite 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[3].

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

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. For a set Υ of directions, the *extension* of Υ is the set $ext(\Upsilon) = \Upsilon \cup \{\varepsilon, \uparrow\}$ (we assume that $\Upsilon \cap \{\varepsilon, \uparrow\} = \emptyset$). A *two-way nondeterministic Büchi path automaton* (2NBP, for short) on Σ-labeled Υ-trees is $\mathcal{S} = \langle \Sigma, P, \delta, p_0, F \rangle$, 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 $(\Delta, p') \in \delta(p, \tau(x))$, and then follows direction Δ and moves to state p'.

Formally, a *run* of a 2NBP $\mathcal{S}$ on a labeled tree $\langle \Upsilon^*, \tau \rangle$ is a sequence of pairs $r = (x_0, p_0), (x_1, p_1), \ldots$ where for all $i \geq 0$, $x_i \in \Upsilon^*$ is a node of the tree and $p_i \in P$ is a state. The pair (x, p) describes a copy of the automaton that reads the node x of Υ^* and is in the state p. Note that many pairs in r may correspond to the same node of Υ^*; Thus, $\mathcal{S}$ may visit a node several times. The run has to satisfy the transition function. Formally, $(x_0, p_0) = (\varepsilon, q_0)$ and for all $i \geq 0$ there is $\Delta \in ext(\Upsilon)$ such that $(\Delta, p_{i+1}) \in \delta(p_i, \tau(x_i))$ and

- If $\Delta \in \Upsilon$, then $x_{i+1} = \Delta \cdot x_i$.
- If $\Delta = \varepsilon$, then $x_{i+1} = x_i$.
- If $\Delta = \uparrow$, then $x_i = \upsilon \cdot z$, for some $\upsilon \in \Upsilon$ and $z \in \Upsilon^*$, and $x_{i+1} = z$.

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

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 $\Delta =\uparrow$, we require that $x_i \neq \varepsilon$. A run r is *accepting* if it visits $\Upsilon^* \times F$ infinitely often. 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 $\mathcal{L}(\mathcal{A}) \neq \emptyset$. 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.

Readers familiar with tree automata know that the run of a tree automaton starts in a single copy of the automaton reading the root of the tree, and then the copy splits to the successors of the root and so on, thus the run simultaneously follows many paths in the input tree. In contrast, a path automaton has a single copy at all times. It starts from the root and it always chooses a single direction to go to. In two-way path automata, the direction may be "up", so the automaton can read many paths of the tree, but it cannot read them simultaneously. The fact that a 2NBP has a single copy influences its expressive power and the complexity of its nonemptiness and membership problems. In the full version we study the expressive power of 2NBP. We show that a 2NBP cannot recognize even very simple properties that refer to all the branches of the tree. However, when a universal property considers only a bounded prefix of the branch, it can be recognized by a 2NBP. We now turn to study the emptiness and membership problems of 2NBP.

Given a 2NBP $\mathcal{S}$, the *emptiness problem* is to determine whether $\mathcal{S}$ accepts some tree, or equivalently whether $\mathcal{L}(\mathcal{S}) = \emptyset$. The *membership problem* of $\mathcal{S}$ and a regular tree $\langle \Upsilon^*, \tau\rangle$ is to determine whether $\mathcal{S}$ accepts $\langle \Upsilon^*, \tau\rangle$, or equivalently $\langle \Upsilon^*, \tau\rangle \in \mathcal{L}(\mathcal{S})$. The fact that 2NBP cannot spawn new copies makes them similar to word automata. Thus, the membership problem for 2NBP can be reduced to the emptiness problem of one-way weak alternating automata on infinite words (1AWW) over a 1-letter alphabet (cf. [KVW00]). The reduction yields a polynomial time algorithm for solving the membership problem. In contrast, the emptiness problem of 2NBP is EXPTIME-complete.

In the full version, we show that we can reduce the membership problem of 2NBP to the emptiness problem of alternating word automata. The reduction generalizes the construction in [PV01a,PV01b,Pit00]. We combine this reduction with an algorithm for checking the emptiness of alteranting word automata [KVW00]. Formally, we have the following.

Theorem 3. *Consider a 2NBP $\mathcal{S} = \langle \Sigma, P, p_0, \delta, F\rangle$.*

- *The membership problem of the regular tree $\langle \Upsilon^*, \tau\rangle$ in the language of $\mathcal{S}$ is solvable in time $O(|P|^2 \cdot |\delta| \cdot \|\tau\|)$ and space $O(|P|^2 \cdot \|\tau\|)$.*
- *The emptiness problem of $\mathcal{S}$ is EXPTIME-complete.*

We note that the membership problem for 2-way alternating Büchi automata on trees (2ABT) is EXPTIME-complete. Indeed, CTL model-checking of pushdown systems, proven to be EXPTIME-hard in [Wal00], can be reduced to the membership problem of a regular tree in a 2ABT. The size of the regular tree is linear in the size of the alphabet of the pushdown system and the size of the 2ABT is linear in the size of the

CTL formula. Thus, path automata capture the computational difference between linear and branching specifications.

4 LTL model checking

In this section we solve the LTL model-checking problem by a reduction to the membership problem of 2NBP. We start by demonstrating our technique on LTL model-checking for pushdown systems. Then we show how to extend it to prefix-recognizable systems and to systems with regular labeling. For an LTL formula φ, we construct a 2NBP that navigates through the full infinite V-tree and simulates a computation of the rewrite system that does not satisfy φ. Thus, our 2NBP accepts the V-tree iff the rewrite system does not satisfy the specification. Then, we use the results in Section 3: we check whether the given V-tree is in the language of the 2NBP and conclude whether the system satisfies the property.

Consider a rewrite system $R = \langle \Sigma, V, Q, L, T, q_0, x_0 \rangle$. 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, and results in trees that are not regular. We show that it is possible to label V^* with a regular labeling that is sufficiently informative to provide the 2NBP with the information it needs in order to simulate the transitions of the rewrite system. For pushdown systems with a simple labeling function, we show that it is enough to label a node x by its direction. For prefix-recognizable systems or systems with regular labeling, the label is more complex and reflects the membership of x in the regular expressions that are used in the transition rules and the regular labeling.

Pushdown systems. 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 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 (note that $dir(x)$ is the first letter of x). Formally, we have the following.

Theorem 4. *Given a pushdown system $R = \langle 2^{AP}, V, Q, L, T, q_0, x_0 \rangle$ and an LTL formula φ, there is a 2NBP $\mathcal{S}$ on V-trees such that $\mathcal{S}$ accepts $\langle V^*, \tau_V \rangle$ iff $G_R \not\models \varphi$. The automaton $\mathcal{S}$ has $O(|Q| \cdot \|T\|) \cdot 2^{O(|\varphi|)}$ states and the size of its transition function is $O(\|T\|) \cdot 2^{O(|\varphi|)}$.*

Proof. According to Theorem 1, there is an NBW $\mathcal{M}_{\neg\varphi} = \langle 2^{AP}, W, \eta_{\neg\varphi}, w_0, F \rangle$ such that $\mathcal{L}(\mathcal{M}_{\neg\varphi}) = (2^{AP})^\omega \setminus \mathcal{L}(\varphi)$. The 2NBP $\mathcal{S}$ tries to find a trace in G_R that satisfies $\neg\varphi$. The 2NBP $\mathcal{S}$ runs $\mathcal{M}_{\neg\varphi}$ on a guessed (q_0, x_0)-computation in R. Thus, $\mathcal{S}$ accepts $\langle V^*, \tau_V \rangle$ iff there exists an (q_0, x_0)-trace in G_R accepted by $\mathcal{M}_{\neg\varphi}$. Such a (q_0, x_0)-trace does not satisfy φ, and it exists iff $R \not\models \varphi$. We define $\mathcal{S} = \langle V, P, p_0, \delta, F' \rangle$, where

- $P = W \times Q \times tails(T)$, where $tails(T) \subseteq V^*$ 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 $\mathcal{S}$ 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 $\mathcal{M}^w_{\neg\varphi}$. In particular, when $y = \varepsilon$, then R with initial configuration (q, x) needs to be accepted by $\mathcal{M}^w_{\neg\varphi}$. States of the form $\langle w, q, \varepsilon \rangle$ are called *action states*. From these states $\mathcal{S}$ consults $\eta_{\neg\varphi}$ and T in order to impose new requirements on $\langle V^*, \tau_V \rangle$. States of the form $\langle w, q, y \rangle$, for $y \in V^+$, are called *navigation states*. From these states $\mathcal{S}$ only navigates downwards y to reach new action states.
- $p_0 = \langle w_0, q_0, x_0 \rangle$. Thus, in its initial state $\mathcal{S}$ checks that R with initial configuration (q_0, x_0) contains a trace that is accepted by $\mathcal{M}$ with initial state w_0.
- The transition function δ is defined for every state in $\langle w, q, x \rangle \in W \times Q \times tails(T)$ and letter in $A \in V$ as follows.
  • $\delta(\langle w, q, \epsilon \rangle, A) = \{(\langle w', q', y \rangle, \uparrow) : w' \in \eta_{\neg\varphi}(w, L(q, A))$ 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 $\eta_{\neg\varphi}$. 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 $\mathcal{S}$.

We show that $\mathcal{S}$ accepts $\langle V^*, \tau_V \rangle$ iff $R \not\models \varphi$. Assume first that $\mathcal{S}$ accepts $\langle V^*, \tau_V \rangle$. Then, there exists an accepting run $(p_0, x_0), (p_1, x_1), \ldots$ of $\mathcal{S}$ on $\langle V^*, \tau_V \rangle$. Extract from this run the subsequence of action states $(p_{i_1}, x_{i_1}), (p_{i_2}, x_{i_2}), \ldots$. As the run is accepting and only action states are accepting states we know that this subsequence is infinite. Let $p_{i_j} = \langle w_{i_j}, q_{i_j}, \varepsilon \rangle$. 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 $w_{i_1}, w_{i_2}, \ldots$ is an accepting run of $\mathcal{M}_{\neg\varphi}$ on the trace of this path. Hence, G_R contains a trace that is accepted by $\mathcal{M}_{\neg\varphi}$, thus $R \not\models \varphi$.

Assume now that $R \not\models \varphi$. Then, there exists a path $(q_0, x_0), (q_1, x_1), \ldots$ 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 $\langle V^*, \tau_V \rangle$.

Prefix-recognizable systems. We now turn to consider prefix-recognizable systems. Again the configuration of a prefix-recognizable system $R = \langle \Sigma, V, Q, L, T, q_0, x_0 \rangle$ 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 not hard: the 2NBP can emulate the run of the automaton $\mathcal{U}_{\alpha_i}$ backwards while going up the tree and the run of the automaton $\mathcal{U}_{\gamma_i}$ while going down the guessed y'. 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 run $\mathcal{U}_{\alpha_i}$ 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. 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. 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. Let $\Sigma = V \times \Pi_{1 \leq i \leq n} D_{\beta_i}$. 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. Note that $\tau_\beta(x)[i] \in F_{\beta_i}$ iff x is in the language of β_i. Note also that $\langle V^*, \tau_\beta \rangle$ is a regular tree whose size is exponential in the sum of the lengths of $\beta_1, \ldots, \beta_n$.

Theorem 5. *Given a prefix-recognizable system $R = \langle \Sigma, V, Q, L, T, q_0, x_0 \rangle$ and an LTL formula φ, there is a 2NBP $\mathcal{S}$ such that $\mathcal{S}$ accepts $\langle V^*, \tau_\beta \rangle$ iff $R \not\models \varphi$. The automaton $\mathcal{S}$ has $O(|Q| \cdot (|Q_\alpha| + |Q_\gamma|) \cdot |T|) \cdot 2^{O(|\varphi|)}$ states and the size of its transition function is $O(\|T\|) \cdot 2^{O(|\varphi|)}$.*

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 a final 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 chooses a final state of $\mathcal{U}_{\alpha_{i'}}$, and run $\mathcal{U}_{\alpha_i}$ backwards up the tree until it reaches the initial state. It then verifies that the current node is in the language of β_i, in which case it moves to the initial state of $\mathcal{U}_{\gamma_i}$ and runs it forward down the tree until it reaches a new action state.

Regular labeling. Handling regular labels for pushdown systems or prefix-recognizable systems is similar to the above. We add to the label of every node in the tree V^* also the states of the deterministic automata that recognizes the languages of the regular expressions of the labels. The navigation through the V-tree proceeds as before, and whenever the 2NBP needs to know the label of the current configuration (that is, in action states, when it has to update the state of $\mathcal{M}_{\neg\varphi}$), it consults the labels of the tree.

If we want to handle a prefix-recognizable system with regular labeling we have to label the nodes of the tree V^* by both the deterministic automata for regular expressions β_i and the deterministic automata for regular expressions $R_{q,p}$. Let $\langle V^*, \tau_{\beta,L} \rangle$ denote this tree. Again note that $\langle V^*, \tau_{\beta,L} \rangle$ is a regular tree of exponential size.

Theorem 6. *Given a prefix-recognizable system $R = \langle \Sigma, V, Q, L, T, q_0, x_0 \rangle$ and an LTL formula φ, there is a 2NBP $\mathcal{S}$ such that $\mathcal{S}$ accepts $\langle V^*, \tau_{\beta,L} \rangle$ iff $R \not\models \varphi$. The automaton $\mathcal{S}$ has $O(|Q| \cdot (|Q_\alpha| + |Q_\gamma|) \cdot |T|) \cdot 2^{O(|\varphi|)}$ states and the size of its transition function is $O(\|T\|) \cdot 2^{O(|\varphi|)}$.*

Note that Theorem 6 differs from Theorem 5 only in the labeled tree whose membership is checked. Also, all the three labeled trees we use are regular, with $\|\tau_V\| = O(|V|)$, $\|\tau_\beta\| = 2^{O(|Q_\beta|)}$, and $\|\tau_{\beta,L}\| = 2^{O(|Q_\beta|+\|L\|)}$. Combining Theorems 4, 5, 6, and 3, we get the following.

Theorem 7. *The model-checking problem for a rewrite system R and an LTL formula φ is solvable*

- *in time $O(\|T\|^3) \cdot 2^{O(|\varphi|)}$ and space $O(\|T\|^2) \cdot 2^{O(|\varphi|)}$ when R is a pushdown system with simple labeling.*
- *in time $O(\|T\|^3) \cdot 2^{O(|\varphi|+|Q_\beta|)}$ and space $O(|T|^2) \cdot 2^{O(|\varphi|+|Q_\beta|)}$ when R is a prefix-recognizable system with simple labeling. The problem is EXPTIME-hard in $|Q_\beta|$ even for a fixed formula.*
- *in time $O(\|T\|^3) \cdot 2^{O(|\varphi|+|Q_\beta|+\|L\|)}$ and space $O(|T|^2) \cdot 2^{O(|\varphi|+|Q_\beta|+\|L\|)}$ when R is a prefix-recognizable system with regular labeling L.*

For pushdown systems with simple labeling (the first setting), our complexity coincides with the one in [EHRS00]. In the full version, we prove the EXPTIME lower bound in the second setting by a reduction from the membership problem of a linear space alternating Turing machine. An alternative proof is given in Theorem 2. This, together with the lower bound in [EKS01], implies EXPTIME-hardness in terms of $|Q_\beta|$ and $\|L\|$ in the the third setting. Thus, our upper bounds are tight.

References

[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 TCS*, 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.

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

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

[BS92] O. Burkart and B. Steffen. Model checking for context-free processes. In *3rd Concur*, LNCS 630, 123–137, 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. *Theoretical Computer Science*, 221:251–270, 1999.

[Bur97a] O. Burkart. Automatic verification of sequential infinite-state processes. LNCS 1354.

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

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

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

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

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *J. of ACM*, 28(1):114–133, January 1981.

[Dam94] M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *TCS*, 126:77–96.

[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.

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

[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.

[EL86] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *1st LICS*, 267–278, 1986.

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

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

[KP95] O. Kupferman and A. Pnueli. Once and for all. In *10th LICS*, 25–35, 1995.

[Kur94] R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton Univ. Press, 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, 2000.

[KV01a] O. Kupferman and M.Y. Vardi. On clopen specifications. In *8th LPAR*, LNCS 2250, 24–38, 2001.

[KV01b] O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. on Computational Logic*, 2001(2):408–429, July 2001.

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

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

[Lyn77] N. Lynch. Log space recognition and translation of parenthesis languages. *Journal ACM*, 24:583–590, 1977.

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

[Pit00] N. Piterman. Extending temporal logic with ω-automata. M.Sc. Thesis, The Weizmann Institute of Science, Israel, 2000. http://www.wisdom.weizmann.ac.il/home/nirp/public_html/publications/msc_thesis.ps.

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

[PV01a] N. Piterman and M. Vardi. From bidirectionality to alternation. In *26th MFCS*, LNCS 2136, 598–609, 2001.

[PV01b] N. Piterman and M. Vardi. From bidirectionality to alternation. *TCS*, 2001. to appear.

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

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

[Wal96] I. Walukiewicz. Pushdown processes: games and modal logic. In *8th CAV*, LNCS 1102, 62–74, 1996.

[Wal00] I. Walukiewicz. Model checking ctl properties of pushdown systems. In *20th FSTTCS*, LNCS 1974, 127–138, 2000.

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