Model Checking Linear Properties of Prefix-Recognizable Systems

Orna Kupferman*
Hebrew University

Nir Piterman†
Weizmann Institute of Science

Moshe Y. Vardi‡
Rice University

January 27, 2002

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. For branching properties, the framework is known and the two-way alternating automaton is a tree automaton. Applying the framework for linear properties results in algorithms that are not optimal. Indeed, the fact that a tree automaton can split to copies and simultaneously read all the paths of the tree has a computational price and is irrelevant for linear properties. 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.

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 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, and in fact we show that LTL model checking with respect to pushdown systems with regular labeling is intereducible with LTL model checking with respect to prefix-recognizable systems with simple labeling.

*Address: School of Computer Science and Engineering, Hebrew University, Jerusalem 91904, Israel. Email: orna@cs.huji.ac.il

†Department of Computer Science and Applied Mathematics, Weizmann institute, Rehovot 76100, Israel. Email: nirp@wisdom.weizmann.ac.il

‡Address: Department of Computer Science, Rice University, Houston TX 77005-1892, U.S.A. Email: vardi@cs.rice.edu

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 [BLM01, CFF$^+$01].

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 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 [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 match 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 where 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 transitions of a prefix-recognizable system are very similar. This leads us to the understanding that regular labeling and prefix recognizability have exactly the same power. Formally, we prove that LTL model checking in a prefix-recognizable system can be reduced to LTL model checking in a pushdown system with regular labeling, and vice versa. Since the latter problem is known to 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.

[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 discssion regarding past-time connectives in branching temporal logics, see [KP95].

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.

2.1 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, $\delta : Q \times \Sigma \to 2^Q$ 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 $|\delta(q, a)| = 1$ 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 $p_{j+1} \in \delta(p_j, w_j)$. 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 2.1 [VW94] *For every LTL formula φ, there exists an NBW N_φ with $2^{O(|\varphi|)}$ states, such that $L(N_\varphi) = L(\varphi)$.*

2.2 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 2.1 has to be modified so that it can recognize also finite words. Our results are easily extended to consider also finite paths).

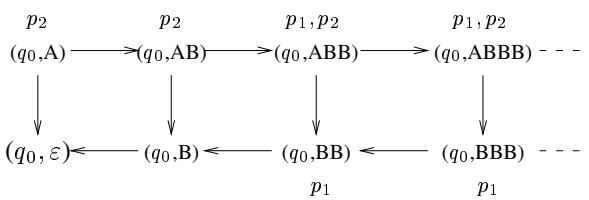
A *rewrite system* is $R = \langle \Sigma, V, Q, L, T, q_0, x_0 \rangle$, where Σ is a finite set of labels, V is a finite alphabet, Q is a finite set of states, $L : Q \times V^* \to \Sigma$ is a labeling function, T is a finite set of rewrite rules, to be defined below, q_0 is an initial state, and $x_0 \in V^*$ 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. We use prefix-recognizable systems with control states for the sake of uniform notation.

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 w 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 $\langle q, \alpha, \beta, \gamma, q' \rangle \in T$. Note that in order to apply a rewrite rule in state $(q, z) \in Q \times V^*$ of a pushdown graph, we only need to match the state q and the first letter of z with the second element of a rule. On the other hand, in an application of a rewrite rule in a prefix-recognizable graph, we have to match the state q and we should find a partition of z to a prefix that belongs to the second element of the rule and a suffix that belongs to the third element. A labeled transition graph that is induced by a pushdown system is called a *pushdown graph*. A labeled transition system that is induced by a prefix-recognizable system is called a *prefix-recognizable graph*. We say that a rewrite system R satisfies an LTL formula φ if $G_R \models \varphi$. [2]

Example 2.2 *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 on the right.*



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

[2] Some work on verification of infinite-state system (e.g., [EHRS00]), consider properties given by nondeterministic Büchi word automata, rather than LTL formulas. Since we anyway translate LTL formulas to automata, we can easily handle also properties given by automata.

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 2.3 *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].

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 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[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 then say that the size of the regular tree $\langle \Upsilon^*, \tau \rangle$, denoted $\|\tau\|$, is $|Q|$, the number 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

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

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 $x_{i+1} = z$.

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. We now turn to study these issues. One-way nondeterministic path automata can read a single path of the tree, so it is easy to see that they accept exactly all languages $\mathcal{T}$ of trees such that there is an ω-regular language L of words and $\mathcal{T}$ contains exactly all trees that have a path labeled by a word in L. For two-way path automata, the expressive power is less clear, as by going up and down the tree, the automaton can traverse several paths. Still, a path automaton cannot traverse all the nodes of the tree. To see that, we prove that a 2NBP cannot recognize even very simple properties that refer to all the branches of the tree (*universal* properties for short).

Theorem 3.1 *There are no 2NBP $\mathcal{S}_1$ and $\mathcal{S}_2$ over the alphabet $\{0, 1\}$ such that*

- $L(\mathcal{S}_1) = \{\langle \Upsilon^*, \tau \rangle : \tau(x) = 0$ *for all* $x \in T\}$.
- $L(\mathcal{S}_2) = \{\langle \Upsilon^*, \tau \rangle :$ *for every path* $\pi \subseteq T$*, there is* $x \in \pi$ *with* $\tau(x) = 0\}$.

The proof of Theorem 3.1 follows from the fact that for every 2NBP $\mathcal{S}$ and an accepting run $(x_0, p_0), (x_1, p_1), \ldots$ of $\mathcal{S}$, there exist $0 \leq i < j$ such that x_j is a descendant of x_i in T, $p_i = p_j$, and there is $i \leq k \leq j$ such that $p_k \in F$. We can construct an alternative accepting run that repeats the movement of the 2NBP from (x_i, p_i) to (x_{i+1}, p_{i+1}) and from (x_{i+1}, p_{i+1}) to (x_{i+2}, p_{i+2}) and so on until (x_j, p_j) and iterate ad infinitum. This alternative run does not traverse all the nodes of the input tree, implying that the 2NBP accepts a tree that is not in the language yet agrees with a tree on the language on some of its nodes.

There are, however, universal properties that a 2NBP can recognize. Consider a language $L \subseteq \Sigma^\omega$ of infinite words over the alphabet Σ. A finite word $x \in \Sigma^*$ is a *bad prefix* for L iff for all $y \in \Sigma^\omega$, we have $x \cdot y \notin L$. Thus, a bad prefix is a finite word that cannot be extended to an infinite word in L. A language L is a *safety* language iff every $w \notin L$ has a finite bad prefix. A language $L \subseteq \Sigma^\omega$ is *clopen* if both L and its complement are safety languages, or, equivalently, L corresponds to a set that is both closed and open in Cantor space. It is known that a clopen language is bounded: there is an integer k such that after reading a prefix of length k of a word $w \in \Sigma^\omega$, one can determine whether w is in L [KV01a]. A 2NBT can then traverse all the paths of the input tree up to level k (given L, its bound k can be calculated), hence the following theorem.

Theorem 3.2 *Let $L \subseteq \Sigma^\omega$ be a clopen language. There is a 2NBP $\mathcal{S}$ such that $L(\mathcal{S}) = \{\langle \Upsilon^*, \tau \rangle$: for all paths $\pi \subseteq \Upsilon^*$, we have $\tau(\pi) \in L\}$.*

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 very 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 Appendix A, we give the exact definition of 1AWW and show a reduction from the membership problem of 2NBP to the emptiness problem of 1AWW with a 1-letter alphabet. The reduction is a generalization of a construction that translates two-way nondeterministic Büchi automata on infinite words to 1AWW [PV01a, PV01b, Pit00]. The emptiness of 1AWW with a 1-letter alphabet is solvable in linear time and space [KVW00]. In the full version we also prove that the emptiness problem of 2NBP is EXPTIME-complete. Formally, we have the following.

Theorem 3.3 *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 4.1 *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 2.1, 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 $\Sigma = V \times \Pi_{1 \le i \le n} D_{\beta_i}$. For a letter $\sigma \in \Sigma$, let $\sigma[i]$, for $i \in \{0, \ldots n\}$, denote the i-th element in σ (that is, $\sigma[0] \in V$ and $\sigma[i] \in D_{\beta_i}$ for $i > 0$). Let $\langle V^*, \tau_\beta\rangle$ denote the Σ-labeled V-tree such that $\tau_\beta(\epsilon) = \langle \bot, q^0_{\beta_1}, \ldots, q^0_{\beta_n}\rangle$, 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 if $\tau_\beta(x)[i] \in F_{\beta_i}$ 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 the regular expressions $\beta_1, \ldots, \beta_n$.

Theorem 4.2 *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. The full details can be found in Appendix C.

Regular labeling. Handling regular labels for either 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.

Formally, let $\{R_1, \ldots, R_n\}$ denote the set of regular expressions R_i such that there exist some state $q \in Q$ and proposition $p \in AP$ with $R_i = R_{q,p}$. Let $\mathcal{D}_{R_i} = \langle V, D_{R_i}, q^0_{R_i}, \eta_{R_i}, F_{R_i} \rangle$ be the deterministic automaton for the language of R_i. For a word $x \in V^*$, we denote by $\eta_{R_i}(x)$ the unique state that $\mathcal{D}_{R_i}$ reaches after reading the word x. Let $\Sigma = V \times \Pi_{1 \le i \le n} D_{R_i}$. For a letter $\sigma \in \Sigma$ let $\sigma[i]$, for $i \in \{0, \ldots, n\}$, denote the i-th element of σ. Let $\langle V^*, \tau_L \rangle$ be the Σ-labeled V-tree such that $\tau_L(\epsilon) = \langle \bot, q^0_{R_1}, \ldots, q^0_{R_n} \rangle$ and for every node $A \cdot x \in V^+$ we have $\tau_L(A \cdot x) = \langle A, \eta_{R_1}(A \cdot x), \ldots, \eta_{R_n}(A \cdot x) \rangle$. The 2NBP $\mathcal{S}$ reads $\langle V^*, \tau_L \rangle$. Note that if the state space of $\mathcal{S}$ indicates that the current state of the rewrite system is q and $\mathcal{S}$ reads the node x, then for every atomic proposition p, we have that $p \in L(q, x)$ iff $\tau_L(x)[i] \in F_{R_i}$, where i is such that $R_i = R_{q,p}$. In action states, $\mathcal{S}$ needs to updates the state of $\mathcal{M}_{\neg\varphi}$, which reads the label of the current configuration. Based on its current state and τ_L, the 2NBP $\mathcal{S}$ knows the letter with which $\mathcal{M}_{\neg\varphi}$ proceeds. Note that the way we handle regular labeling is very similar to the way we handle prefix recognizability. We will get back this point in Section 5.

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 $\langle V^*, \tau_{\beta,L} \rangle$ be the composition of $\langle V^*, \tau_\beta \rangle$ with $\langle V^*, \tau_L \rangle$. Again note that $\langle V^*, \tau_{\beta,L} \rangle$ is a regular tree of exponential size.

Theorem 4.3 *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 4.3 differs from Theorem 4.2 only in the labeled tree whose membership is checked. Also, all the three labled 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 4.1, 4.2, 4.3, and 3.3, we get the following.

Theorem 4.4 *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 Appendix B, 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 2.3. 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.

5 Relating regular labeling with prefix-recognizability

Recall that the way we handle regular labeling is very similar to the way we handle prefix recognizability. In both settings, the system has to be able to check the membership of the word in the store in a regular expression. In prefix-recognizable systems, the check is done when the system follows a transition rule. In systems with regular labeling, the check is done when the system needs to evaluate the labeling of the current configuration. In this section we show that these checks are intereducible. We describe a reduction from the LTL model-checking problem of a prefix-recognizable system with a simple labeling function to the LTL model-checking problem of a pushdown system with a regular labeling function, and a reduction from the LTL model-checking problem of a pushdown system with a regular labeling function to the LTL model-checking problem of a prefix-recognizable system with a simple labeling function. We note that we cannot just replace one system by another, but we also have to adjust the LTL formula. We start with the first direction.

Theorem 5.1 *Given a prefix-recognizable system* $R = \langle 2^{AP}, V, Q, L, T, q_0, x_0 \rangle$ *and an LTL formula* φ*, there is a pushdown system* $R' = \langle 2^{AP'}, V, Q', L', T', q'_0, x_0 \rangle$ *with a regular labeling function and an LTL formula* φ'*, such that* $R \models \varphi$ *iff* $R' \models \varphi'$*. Furthermore,* $|Q'| = |Q| \times |T| \times (|Q_\alpha| + |Q_\gamma|)$*,* $\|T'\| = O(\|T\|)$*, and* $\|L\| = |Q_\beta|$*. The reduction is computable in logarithmic space.*

The idea is to add to the configurations of R labels that would enable the pushdown system to simulates transitions of the prefix-recognizable system. Recall that in order to apply the rewrite rule $\langle q, \alpha, \beta, \gamma, q' \rangle$ from configuration (q, x), the prefix-recognizable system has to find a partition $y \cdot z$ of x such that the prefix y is a word in α and the suffix z is a word in β. It then replaces y by a word $y' \in \gamma$. The pushdwon system can remove the prefix y letter by letter, guess whether the remaining suffix z is a word in β, and add y' letter by letter. In order to check the validity of guesses, the system marks every configuration where it guesses that the remaining suffix is a word in β. It then consults the regular labeling function in order to single out traces in which a wrong guess is made. For that, we add a new proposition, not_wrong, which holds in a configuration iff it is not the case that pushdown system guesses that the suffix z is in the language of some regular expression r and the guess turns out to be incorrect. The pushdown system also marks the configurations where it finishes handling some rewrite rule. For that, we add a new proposition, $ch\text{-}rule$, which is true only when the system finishes handling some rewrite rule and starts handling another.

The pushdown system R' has four modes of operation when it simulates a transition that follows a rewrite rule $\langle q, \alpha, \beta, \gamma, q' \rangle$. In *delete* mode, R' deletes letters from the store x while emulating a run of $\mathcal{U}_{\alpha_i}$ backward. Delete mode starts from a final state of $\mathcal{U}_{\alpha_i}$, from which R' proceeds backward until it reaches the initial

state of $\mathcal{U}_{\alpha_i}$. Once the initial state of $\mathcal{U}_{\alpha_i}$ is reaches, R' transitions to *change-direction* mode, where it does not change the store and just moves to the initial state of $\mathcal{U}_{\gamma_i}$, and transitions to *write* mode. In write mode, R' guesses letters in V and emulates the run of $\mathcal{U}_{\gamma_i}$ on them, while adding them to the store. From a final state of $\mathcal{U}_{\gamma_i}$ the pushdown system R' transitions to *change-rule* mode, where it chooses a new rewrite rule $\langle q', \alpha_{i'}, \beta_{i'}, \gamma_{i'}, q'' \rangle$ and transitions to delete mode. Note that if delete mode starts in configuration (q, x) it cannot last indefinitely. Indeed, the pushdown system can remove only finitely many letters from the store. On the other hand, since the store is unbounded, write mode can last foever. Hence, traces along which $ch\text{-}rule$ occurs only finitely often should be singled out.

Singling out of traces is done by the formula φ' which restricts attention to traces in which not_wrong is always asserted and $ch\text{-}rule$ is asserted infinitely often. Formally, R' has the following components

- $AP' = AP \cup \{not_wrong, ch\text{-}rule\}$.
- $Q' = Q \times T \times (\{ch\text{-}dir, ch\text{-}rule\} \cup Q_\alpha \cup Q_\gamma)$. A state $\langle q, t, s \rangle \in Q'$ maintains the state $q \in Q$ and the rewrite rule t currently being applied. the third element s indicates the mode of R'. Change-direction and change-rule modes are indicated by a marker. In delete and write modes, R' also maintains the current state of $\mathcal{U}_\alpha$ and $\mathcal{U}_\gamma$.
- For every proposition $p \in AP$, we have $p \in L'(q, x)$ iff $p \in L(q, x)$. We now describe the regular expression for the propositions $ch\text{-}rule$ and not_wrong. The proposition $ch\text{-}rule$ holds in all the configuration in which the system is in change-rule mode. Thus, for every $q \in Q$ and $t \in T$, we have $R_{\langle q,t,ch\text{-}rule\rangle, ch\text{-}rule} = V^*$ and $R_{\langle q,t,\zeta\rangle, ch\text{-}rule} = \emptyset$ for $\zeta \neq ch\text{-}rule$. The proposition not_wrong holds in configurations in which we are not in change-direction mode, or configuration in which we are in change-direction mode and the store is in β, thus changing direction is possible in the configuration. Formally, for every $q \in Q$ and $t = \langle q', \alpha, \beta, \gamma, q \rangle \in T$, we have $R_{\langle q,t,ch\text{-}dir\rangle, not_wrong} = \beta$ and $R_{\langle q,t,\zeta\rangle, not_wrong} = V^*$ for $\zeta \neq ch\text{-}dir$.
- $q'_0 = \langle q_0, t, ch\text{-}rule \rangle$ for some arbitrary rewrite rule t.

The transition function of R' includes four types of transitons according to the four operation modes. In change-direction mode, in configuration $(\langle q, t, ch\text{-}dir \rangle, x)$ that applies the rewrite rule $t = \langle q', \alpha_i, \beta_i, \gamma_i, q \rangle$, the system R' does not change x, and moves to the initial state $q^0_{\gamma_i}$ of $\mathcal{U}_{\gamma_i}$. In change rule mode, in configuration $(\langle q, t, ch\text{-}rule \rangle, x)$, the system R' does not change x, it chooses a new rewrite rule $t' = \langle q, \alpha_{i'}, \beta_{i'}, \gamma_{i'}, q' \rangle$, changes the Q component to q', and moves to the initial state $q^0_{\alpha_{i'}}$ of $\mathcal{U}_{\alpha_{i'}}$. In delete mode, in configuration $(\langle q, t, s \rangle, x)$, for $t = \langle q', \alpha_i, \beta_i, \gamma_i, q \rangle$ and $s \in Q_{\alpha_i}$, the system R' proceeds by either removing one letter from x and continuing the run of $\mathcal{U}_{\alpha_i}$ backward, or if $s = q^0_{\alpha_i}$ is the initial state of $\mathcal{U}_{\alpha_i}$ then R' may also leave x unchanged, and changes s to $ch\text{-}dir$. In write mode, in configuration $(\langle q, t, s \rangle, x)$, for $t = \langle q', \alpha_i, \beta_i, \gamma_i, q \rangle$ and $s \in Q_{\gamma_i}$, the system R' proceeds by either extending x with a guessed symbol from V and continuing the run of $\mathcal{U}_{\gamma_i}$ using the guessed symbol, or if $s \in F_{\gamma_i}$, then R' may also not change x and just replaces s by $ch\text{-}rule$. Formally, $T' = T'_{ch\text{-}rule} \cup T'_{ch\text{-}dir} \cup T'_\alpha \cup T'_\gamma$, where

- $T'_{ch\text{-}rule} = \{(\langle q, t, ch\text{-}rule \rangle, A, A, \langle q', t', s \rangle) \mid t' = \langle q, \alpha_i, \beta_i, \gamma_i, q' \rangle,\ s \in F_{\alpha_i} \textit{ and } A \in V\}$.
- $T'_{ch\text{-}dir} = \left\{(\langle q, t, ch\text{-}dir \rangle, A, A, \langle q, t, s \rangle) \;\middle|\; t = \langle q', \alpha_i, \beta_i, \gamma_i, q \rangle,\ s = q^0_{\gamma_i},\ \textit{and } A \in V\right\}$.

Note that the same letter A is removed from the store and added again. Thus, the store content of the configuration does not change.

- $T'_\alpha = \begin{array}{l} \{(\langle q,t,s\rangle, A, \epsilon, \langle q,t,s'\rangle) \mid t = \langle q', \alpha_i, \beta_i, \gamma_i, q\rangle,\ s \in Q_\alpha,\ s \in \rho_{\alpha_i}(s', A), \text{ and } A \in V\} \cup \\ \{(\langle q,t,s\rangle, A, A, \langle q,t,\textit{ch-dir}\rangle) \mid t = \langle q', \alpha_i, \beta_i, \gamma_i, q\rangle,\ s \in Q_\alpha,\ s = q^0_{\alpha_i},\ \textit{and } A \in V\}. \end{array}$

- $T'_\gamma = \begin{array}{l} \{(\langle q,t,s\rangle, A, AB, \langle q,t,s'\rangle) \mid t = \langle q', \alpha_i, \beta_i, \gamma_i, q\rangle,\ s \in Q_\gamma,\ s' \in \rho_{\gamma_i}(s, B), \text{ and } A, B \in V\} \cup \\ \{(\langle q,t,s\rangle, A, A, \langle q,t,\textit{ch-rule}\rangle) \mid t = \langle q', \alpha_i, \beta_i, \gamma_i, q\rangle,\ s \in Q_\gamma,\ s \in F_{\gamma_i} \textit{ and } A \in V\}. \end{array}$

Note that as initial states have no incoming edges, it is always the case that after a state $\langle q, \langle q', \alpha_i, \beta_i, \gamma_i, q\rangle, q^0_{\alpha_i}\rangle$ we visit the state $\langle q, t, \textit{ch-dir}\rangle$. Similarly, as final states have no outgoing edges, we always visit the state $\langle q, t, \textit{ch-rule}\rangle$ after visiting a state $\langle q, \langle q', \alpha_i, \beta_i, \gamma_i, q\rangle, s\rangle$ where $s \in F_{\gamma_i}$.

Finally, the formula φ' is the implication $\varphi'_1 \to \varphi'_2$ of two formulas. The formula φ'_1 holds in computations of R' that corresponds to real computations to R. Thus, $\varphi'_1 = \Box \textit{not_wrong} \wedge \Box\Diamond \textit{ch-rule}$. Then, φ'_2 adjusts φ to the fact that a single transition in R corresponds to multiple transitions in R'. Formally, $\varphi'_2 = f(\varphi)$, for the function f defined below.

- $f(p) = p$ for a proposition $p \in AP$
- $f(\neg a) = \neg f(a)$, $f(a \vee b) = f(a) \vee f(b)$, and $f(a \wedge b) = f(a) \wedge f(b)$.
- $f(a\mathcal{U}b) = (\textit{ch-rule} \to f(a))\mathcal{U}(f(b) \wedge \textit{ch-rule})$
- $f(\bigcirc a) = \bigcirc((\neg \textit{ch-rule})\mathcal{U}(f(a) \wedge \textit{ch-rule}))$

In Appendix C, we prove that $R \models \varphi$ iff $R' \models \varphi'$. If we use this construction in conjunction with Theorem 2.3, we get an algorithm whose complexity coincides with the one in Theorem 4.4.

We note that since we end up with a pushdown system with regular labeling, it is easy to extend the reduction to start with a prefix-regocnizable system with regular labeling. It is left to show the reduction in the other direction.

Theorem 5.2 *Given a pushdown system $R = \langle 2^{AP}, V, Q, T, L, q_0, x_0\rangle$ with a regular labeling function and an LTL formula φ, there is a prefix-recognizable system $R' = \langle 2^{AP'}, V, Q', T', L', q'_0, x_0\rangle$ with simple labeling and an LTL formula φ' such that $R \models \varphi$ iff $R' \models \varphi'$. Furthermore, $Q' = O(|Q| \cdot |AP|)$, $|Q'_\alpha| + |Q'_\gamma| = O(\|T\|)$, and $|Q'_\beta| = 2^{\|L\|}$ yet the automata for Q'_β are deterministic. The reduction is computable in polynomial space.*

Let $AP = \{p_1, \ldots, p_n\}$ be the set of atomic propositions. The idea behind the reduction is as follows. The state space of R' is $Q \times (\{\textit{start}\} \cup AP) \times \{\bot, \top\}$. We replace a configuration (q, x) in R by a sequence $\langle(\langle q, \textit{start}, \bot\rangle, x), (\langle q, p_1, \lambda_1\rangle, x), \ldots, (\langle q, p_n, \lambda_n\rangle, x)\rangle$ of $n+1$ configurations in R', where $\lambda_i = \bot$ if $x \notin R_{q,p_i}$ and $\lambda = \top$ if $x \in R_{q,p_i}$. Each of the last n states corresponds to one of the propositions in AP. The new rewrite rule checks that the marking of $\bot$ and $\top$ is indeed correct by matching the regular expression β of the transition with the regular expression of the proposition. For that, we use two types of transition rules. First, the transition rule $\langle\langle q, p_{i-1}, \zeta\rangle, \epsilon, R_{q,p_i}, \epsilon, \langle q, p_i, \top\rangle\rangle$ marks p_i as true and makes sure that $x \in R_{q,p_i}$. Second, $\langle\langle q, p_{i-1}, \zeta\rangle, \epsilon, \widetilde{R}_{q,p_i}, \epsilon, \langle q, p_i, \bot\rangle\rangle$, where $\widetilde{R}$ is the regular expression for the complement of R, marks p_i as false and makes sure that $x \notin R_{q,p_i}$. The automaton $\widetilde{\mathcal{U}}_{q,p_i}$ that recognizes the language of $\widetilde{R}_{q,p_i}$ may be exponentially larger than $\mathcal{U}_{q,p_i}$. Thus, the system R' may be exponentially larger than R. However, reasoning about the correctness of R' requires the automata for the regular expressions to be deterministic, thus although R' may be exponentially larger than R, the model-checking problem of R' is only exponential in $\|L\|$ and not doubly exponential. The full construction of the prefix-recognizable system R' is given in Appendix E.

In order to define the LTL formula φ' we define the function f from LTL formulas to LTL formulas as follows. Intuitively, $f(\varphi)$ adjusts φ to the representation of a single state and its labeling by a chain of $|AP|+1$ states (the function assumes that $|AP| = n$).

- For the proposition $p_i \in AP$ we have $f(p_i) = \bigcirc^i p_i$
- $f(\neg a) = \neg f(a)$, $f(a \vee b) = f(a) \vee f(b)$, and $f(a \wedge b) = f(a) \wedge f(b)$.
- $f(a\mathcal{U}b) = (start \rightarrow f(a))\mathcal{U}(f(b) \wedge start)$
- $f(\bigcirc a) = \bigcirc^{n+1} f(a)$

The proof that $G_R \models \varphi$ iff $G_{R'} \models f(\varphi)$ resembles the one in Appendix D. If we use this construction in conjunction with Theorem 4.4, we get an algorithm whose complexity coincides with the one in [EKS01].

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 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 Conference on Concurrency Theory*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150, Warsaw, July 1997. Springer-Verlag.

[BLM01] P. Biesse, T. Leonard, and A. Mokkedem. Finding bugs in an alpha microprocessors using satisfiability solvers. In *Computer Aided Verification, Proc. 13th International Conference*, volume 2102 of *Lecture Notes in Computer Science*, pages 454–464. Springer-Verlag, 2001.

[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 Conference on Concurrency Theory*, volume 630 of *Lecture Notes in Computer Science*, pages 123–137. Springer-Verlag, 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. In G. Goos, J. Hartmanis, and J. van Leeuwen, editors, *Lecture Notes in Computer Science*, volume 1354. Springer-Verlag, 1997.

[Bur97b] O. Burkart. Model checking rationally restricted right closures of recognizable graphs. In F. Moller, editor, *Proc. 2nd International workshop on verification of infinite states systems*, 1997.

[Cau96] D. Caucal. On infinite transition graphs having a decidable monadic theory. In *Automata, Languages, and Programming, Proc. 23st ICALP*, volume 1099 of *Lecture Notes in Computer Science*, pages 194–205. Springer-Verlag, 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.

[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 *Computer Aided Verification, Proc. 13th International Conference*, volume 2102 of *Lecture Notes in Computer Science*, pages 436–453. Springer-Verlag, 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. *Journal of the Association for Computing Machinery*, 28(1):114–133, January 1981.

[Dam94] M. Dam. CTL* and ECTL* as fragments of the modal μ-calculus. *Theoretical Computer Science*, 126:77–96, 1994.

[EHRS00] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Computer Aided Verification, Proc. 12th International Conference*, 2000. To appear.

[EJ91] E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 368–377, San Juan, October 1991.

[EKS01] J. Esparza, A. Kucera, and S. Schwoon. Model-checking LTL with regular valuations for pushdown systems. In *Proc. 4th International Symposium on Theoretical Aspects of Computer Software*, volume 2215 of *Lecture Notes in Computer Science*, pages 316–339, Sendai, Japan, October 2001. Springer-Verlag.

[EL86] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Proc. 1st Symp. on Logic in Computer Science*, pages 267–278, Cambridge, June 1986.

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

[HHWT95] T.A. Henzinger, P.-H Ho, and H. Wong-Toi. A user guide to HYTECH. In *Tools and algorithms for the construction and analysis of systems*, volume 1019 of *Lecture Notes in Computer Science*, pages 41–71. Springer-Verlag, 1995.

[KP95] O. Kupferman and A. Pnueli. Once and for all. In *Proc. 10th IEEE Symp. on Logic in Computer Science*, pages 25–35, San Diego, June 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 *Computer Aided Verification, Proc. 12th International Conference*, volume 1855 of *Lecture Notes in Computer Science*, pages 36–52. Springer-Verlag, 2000.

[KV01a] O. Kupferman and M.Y. Vardi. On clopen specifications. In *Proc. 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 2250 of *Lecture Notes in Computer Science*, pages 24–38. Springer-Verlag, 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.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 97–107, New Orleans, January 1985.

[LPY97] K. G. Larsen, P. Petterson, and W. Yi. UPPAAL: Status & developments. In *Computer Aided Verification, Proc. 9th International Conference*, volume 1254 of *Lecture Notes in Computer Science*, pages 456–459. Springer-Verlag, 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. *Theoretical Computer Science*, 37:51–75, 1985.

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

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. on Foundation of Computer Science*, pages 46–57, 1977.

[PV01a] N. Piterman and M. Vardi. From bidirectionality to alternation. In *26th International Symposium on Mathematical Foundations of Computer Science*, volume 2136 of *Lecture Notes in Computer Science*, pages 598–609. Springer-Verlag, August 2001.

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

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th International Symp. on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, 1981.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symp. on Logic in Computer Science*, pages 332–344, Cambridge, 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. Pushdown processes: games and modal logic. In *Computer Aided Verification, Proc. 8th International Conference*, volume 1102 of *Lecture Notes in Computer Science*, pages 62–74. Springer-Verlag, 1996.

[Wal00] I. Walukiewicz. Model checking ctl properties of pushdown systems. In *Proc. 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1974 of *Lecture Notes in Computer Science*, pages 127–138, New Delhi, India, December 2000. Springer-Verlag.

[WVS83] P. Wolper, M.Y. Vardi, and A.P. Sistla. Reasoning about infinite computation paths. In *Proc. 24th IEEE Symp. on Foundations of Computer Science*, pages 185–194, Tucson, 1983.

A The reduction from 2NBP to 1AWW

A.1 Definition of Alternating Automata on infinite words

For a set S, let $B^+(S)$ denote the set of all positive formulas over the set S with $\mathbf{true}$ and $\mathbf{false}$ (i.e., for all $s \in S$, s is a formula and if f_1 and f_2 are formulas, so are $f_1 \wedge f_2$ and $f_1 \vee f_2$). We say that a subset $S' \subseteq S$ *satisfies* a formula $\theta \in B^+(S)$ (denoted $S' \models \theta$) if by assigning true to all members of S' and false to all members of $S \setminus S'$ the formula θ evaluates to true.

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

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

We also consider weak alternating automata. A weak alternating automaton (AWW) is an ABW where the set of states Q is partitioned into disjoint sets, Q_i, such that for each set Q_i, either $Q_i \subseteq F$, in which case Q_i is an *accepting* set, or $Q_i \cap F = \emptyset$, in which case Q_i is a *rejecting* set. In addition there exists a partial order $\leq$ on the collection of the Q_i's such that for every $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, a)$, for some $a \in \Sigma$, we have $Q_j \leq Q_i$. Thus, transitions from a state in Q_i lead to states in either the same Q_i or a lower one. It follows that every infinite path of a run of an AWW ultimately gets "trapped" within some set Q_i. The path then satisfies the acceptance condition F if and only if Q_i is an accepting set.

Again, the size of the automaton is determined by the number of its states and the size of its transition function. The size of the transition function is $|\eta| = \Sigma_{q \in Q} \Sigma_{a \in \Sigma} |\eta(q, a)|$ where, for a formula in $B^+(\{0, 1\} \times Q)$ we define $|(\Delta, q)| = |\mathbf{true}| = |\mathbf{false}| = 1$ and $|\theta_1 \vee \theta_2| = |\theta_1 \wedge \theta_2| = |\theta_1| + |\theta_2| + 1$.

The emptiness of an AWW over 1-letter alphabet was considered in [KVW00]. The algorithm is based on bottom up labeling of the states of the automaton while evaluating the formulas in the transitions of the states. We note that the length of a formula in $B^+(\{0, 1\} \times Q)$ is at most exponential in $|Q|$ and that such formulas can be evaluated in logarithmic space [Lyn77]. Hence, the algorithm can be implemented in space linear in $|Q|$. All it needs is to evaluate formulas in $B^+(\{0, 1\} \times Q)$ and to record for each state whether its language is empty or not.

Theorem A.1 [KVW00] *Given an AWW over 1-letter alphabet $A = \langle \{a\}, Q, q_0, \eta, F \rangle$ we can check whether $L(A)$ is empty in time $O(|\eta|)$ and space $O(|Q|)$.*

A.2 The proof

We reduce the membership problem of a regular tree $\langle \Upsilon^*, \tau \rangle$ in the language of a 2NBP to the emptiness problem of a 1AWW with one letter alphabet.

Given a two-way nondeterministic Büchi automaton [PV01a] show how to construct a 1ABW that accepts the same language. Their construction can be generalized so that given a 2NBP $\mathcal{S}$ we construct a 1ABW with 1-letter alphabet $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\langle \Upsilon^*, \tau \rangle \in \mathcal{L}(\mathcal{S})$. Then we use methods given in [PV01b, Pit00] to convert this 1ABW into a 1AWW.

Theorem A.2 *Given a 2NBP $\mathcal{S} = \langle \Sigma, P, p_0, \delta, F \rangle$ and a regular tree $\langle \Upsilon^*, \tau \rangle$ there exists a 1ABW on 1-letter alphabet $\mathcal{A} = \langle \{a\}, Q, q_0, \eta, F' \rangle$ such that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\langle \Upsilon^*, \tau \rangle \in \mathcal{L}(\mathcal{S})$ and $\mathcal{A}$ has $O(|P|^2 \cdot \|\tau\|)$ states and the size of the transition function is $O(|\delta| \cdot |P|^2 \cdot \|\tau\|)$.*

We use the fact that $\mathcal{A}$ is a one-way automaton and remember the state of the transducer that gives the label to the current node in the tree $\langle \Upsilon^*, \tau \rangle$ as part of the finite control of $\mathcal{A}$. For better intuition of the construction we refer the reader to [PV01a, PV01b].

Proof: Let $\langle \mathcal{D}_\tau, L_\tau \rangle$ be the transducer that generates the labels of τ where $\mathcal{D}_\tau = \langle \Upsilon, D_\tau, \rho_\tau, d^0_\tau, F_\tau \rangle$ is the deterministic automaton and $L_\tau : D_\tau \to \Sigma$ is the labeling function. For a word $w \in \Upsilon^*$ we denote by $\rho_\tau(w)$ the unique state that $\mathcal{D}_\tau$ gets to after reading w. We construct the 1ABW $\mathcal{A} = \langle \{a\}, Q, q_0, \eta, F' \rangle$ as follows.

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

- $F' = (F \times D_\tau \times \{\bot\}) \cup (P \times D_\tau \times \{\top\})$.

In order to define the transition function we have the following defintions. Two functions $f_\alpha : P \times P \to \{\bot, \top\}$ where $\alpha \in \{\bot, \top\}$, and for every state $p \in P$ and alphabet letter $\sigma \in \Sigma$ the set C_p^σ is the set of states from which p is reachable by a sequence of ϵ-transitions reading letter σ and one final $\uparrow$-transition reading σ. Formally

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

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

$$C_p^\sigma = \left\{ p' \;\middle|\; \begin{array}{l} \exists p_0, p_1, \ldots, p_n \in P^+ \text{ such that} \\ p_0 = p',\ p_n = p, \\ \forall 0 < i < n,\ \langle \epsilon, p_i \rangle \in \delta(p_{i-1}, \sigma), \text{ and} \\ \langle \uparrow, p_n \rangle \in \delta(p_{n-1}, \sigma) \end{array} \right\}$$

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

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

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

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

Claim A.3 $\mathcal{L}(\mathcal{A}) \neq \emptyset$ *iff* $\langle \Upsilon^*, \tau \rangle \in \mathcal{L}(\mathcal{S})$

Proof: The proof is very similar to the proof in [PV01a]. It is included here for the sake of completeness.

Let $r = \langle p_0, w_0 \rangle \cdot \langle p_1, w_1 \rangle \cdot \langle p_2, w_2 \rangle \cdots$ be an accepting run of $\mathcal{S}$ on $\langle \Upsilon^*, \tau \rangle$. We add the annotation of the locations in the run $\langle p_0, w_0, 0 \rangle \cdot \langle p_1, w_1, 1 \rangle \cdot \langle p_2, w_2, 2 \rangle \cdots$. We construct the run $\langle T', r' \rangle$ of $\mathcal{A}$. For every node $x \in T'$, if x is labeled by a singleton state we add a tag to x some triplet from the run r. If x is labeled by a pair state we add two tags to x, two triplets from the run r. The labeling and the tagging conform to the following.

- Given a node x lebeled by state $\langle p, d, \alpha \rangle$ and tagged by the triplet $\langle p', w, i \rangle$ from r, we build r' so that $p = p'$ and $d = \rho_\tau(w)$. Furthermore all triplets in r whose third element is greater than j have their second element greater or equal to w (Υ^* is ordered according to the lexical order on the reverse of the words).

- Given a node x labeled by state $\langle q, p, d, \alpha \rangle$ and tagged by the triplets $\langle q', w, i \rangle$ and $\langle p', w', j \rangle$ from r, we build r' so that $q = q'$, $p = p'$, $w = w'$, $d = \rho_\tau(w)$, and $i < j$. Furthermore all triplets in r whose third element k is between i and j, have their second element greater or equal to w. Also, if $j > i + 1$ then $w_{j-1} = \upsilon \cdot w_j$ for some $\upsilon \in \Upsilon$.

Construct the run tree $\langle T', r' \rangle$ of $\mathcal{A}$ as follows. Label the root of T' by $\langle p_0, d^0_\tau, \perp \rangle$. Given a node $x \in T'$ labeled by $\langle p, d, \alpha \rangle$ tagged by $\langle p_i, w_i, i \rangle$. Let $\langle p_j, w_j, j \rangle$ be the minimal $j > i$ such that $w_j = w_i$. If $j = i+1$ then add one son to x, label it $\langle p_j, d, \perp \rangle$ and tag it $\langle p_j, w_j, j \rangle$. If $j > i+1$, then $w_{j-1} = \upsilon \cdot w_i$ for some $\upsilon \in \Upsilon$ and we add two sons to x, label them $\langle p_i, p_j, d, \beta \rangle$ and $\langle p_j, d, \beta \rangle$. We tag $\langle p_i, p_j, d, \beta \rangle$ by $\langle p_i, w_i, i \rangle$ and $\langle p_j, w_j, j \rangle$, and tag $\langle p_j, d, \beta \rangle$ by $\langle p_j, w_j, j \rangle$, β is $\top$ if there is a visit to F between locations i and j in r. If there is no other visit to w_i then $w_{i+1} = \upsilon \cdot w$ for some $\upsilon \in \Upsilon$. We add one son to x and label it $\langle p_{i+1}, \rho_\tau(d, \upsilon), \perp \rangle$. Obviously the labeling and the tagging conform to the asssumption.

Given a node x labeled by $\langle p, q, d, \alpha \rangle$ tagged by $\langle p, w, i \rangle$ and $\langle q, w, j \rangle$. Let $\langle p_k, w, k \rangle$ be the first visit to w between i and j. If $k = i+1$ then add one son to x and label it $\langle p_k, q, d, f_\alpha(p_k, q) \rangle$. If $k > i+1$ then add two sons to x and label them $\langle p, p_k, d, f_{\beta_1}(p, p_k) \rangle$ and $\langle p_k, q, d, f_{\beta_2}(p_k, q) \rangle$ where β_1, β_2 are determined according to the visits to F between i and j. We tag $\langle p, p_k, d, f_{\beta_1}(p, p_k) \rangle$ by $\langle p, w, i \rangle$ and $\langle p_k, w, k \rangle$ and tag $\langle p_k, t, d, f_{\beta_2}(p_k, q) \rangle$ by $\langle p_k, w, k \rangle$ and $\langle q, w', j \rangle$.

If there is no visit to w between i and j it must be the case that all triplets in r between i and j have the same suffix $\upsilon \cdot w$ for some $\upsilon \in \Upsilon$ (otherwise w is visited). We add one son to x labeled $\langle p_{i+1}, q_{j-1}, \rho_\tau(d, \upsilon), f_\alpha(p', q') \rangle$ and tagged by $\langle p_{i+1}, \upsilon \cdot w, i+1 \rangle$ and $\langle p_{j-1}, \upsilon \cdot w, j-1 \rangle$. We are ensured that $p_{j-1} \in C_q^{L_\tau(\rho_\tau(d,\upsilon))}$ as $(\uparrow, p_j) \in \delta(p_{j-1}, \tau(\upsilon \cdot w))$.

Given an accepting run $\langle T', r' \rangle$ of $\mathcal{A}$ we use the recursive algorithm in Figure 1 to construct a run of $\mathcal{S}$ on $\langle \Upsilon^*, \tau \rangle$.

A node $x \cdot a$ in T' is *advancing* if the transition from x to $x \cdot a$ results from an atom $(r'(x \cdot a), 1)$ that appears in $\eta(r'(x))$. An advancing node that is the immediate successor of a singleton state satisfies the disjunct $\bigvee_{\upsilon \in \Upsilon} \bigvee_{(\upsilon, p') \in \delta(p, L(d))} (\langle p', \rho_\tau(d, \upsilon), \perp \rangle, 1)$ in η. We tag this node by the letter υ that was used to satisfy the transition. Similarly, an advancing node that is the immediate successor of a pair state satisfies the disjunct $\bigvee_{\upsilon \in \Upsilon} \bigvee_{\langle \upsilon, p' \rangle \in \delta(p_1, L_\tau(d))} \bigvee_{p'' \in C_{p_2}^{L_\tau(d)}} (\langle p', p'', \rho_\tau(d, \upsilon), f_\alpha(p', p'') \rangle, 1)$ in η. We tag this node by the letter υ that was used to satisfy the transition. We use these tags in order to build the run of $\mathcal{S}$. When handling advancing nodes we update the location on the tree Υ^* according to the tag. For an advancing node x we denote by $tag(x)$ the letter in Υ that tags it. A node is *non advancing* if the transition from x to $x \cdot a$ results from an atom $(r'(x \cdot a), 0)$ that appears in $\eta(r'(x))$.

The function **build_run** uses the variable w to hold the location in the tree $\langle \Upsilon^*, \tau \rangle$. Working on a singleton $\langle p, d, \alpha \rangle$ the variable add_l is used to determine whether p was already added to the run. Working on a pair $\langle p, q, d, \alpha \rangle$ the variable add_l is used to determine whether p was already added to the run and the variable add_r is used to determine whether q was already added to the run.

The intuition behind the algorithm is quite simple. We start with a node x labeled by a singleton $\langle p, d, \alpha \rangle$. If the node is advancing we update w by $tag(x)$. Now we add p to r (if needed). The case where x has one son matches a transition of the form $\langle \Delta, p' \rangle \in \delta(p, L_\tau(d))$. In this case we move to handle the son of x and clearly p' has to be added to the run r. In case $\Delta = \epsilon$ the son of x is non advancing and p' reads the same location w. Otherwise, w is updated by Δ and p' reads $\Delta \cdot w$. The case where x has two sons matches a guess that there is another visit to w. Thus, the computation splits into two sons $\langle p, q, d, \beta \rangle$ and $\langle q, d, \beta \rangle$. Both sons are non advancing. The state p was already added to r and q is added to r only in the first son.

With a node x labeled by a pair $\langle p, q, d, \alpha \rangle$, the situation is similar. The case where x has one non advancing son matches a transition of the form $\langle \epsilon, s' \rangle \in \delta(p, A)$. Then we move to the son. The state p' is added to r but q is not. The case where x has two non advancing sons matches a split to $\langle p, p', d, \alpha_1 \rangle$ and $\langle p', q, d, \alpha_2 \rangle$. Only p' is added to r as p and q are added by the current call to build_run or by an earlier call to build_run. The case where x has one advancing son matches the move to the state $\langle p', q', \rho_\tau(d, \upsilon), \alpha \rangle$ and checking that $q' \in C_q^{L_\tau(\rho_\tau(d,\upsilon))}$. Both p' and q' are added to r.

It is quite simple to see that the resulting run is a valid and accepting run of $\mathcal{S}$ on $\langle \Upsilon^*, \tau \rangle$.

```
build_run (x, r'(x) = ⟨p, d, α⟩, w, add_l, add_r)
  if (advancing(x))
    w := tag(x) · w;
  if (add_l)
    r := r · ⟨p, w⟩;
  if (x has one son x · a)
    build_run (x · a, r'(x · a), w, 1, 0)
  if (x has two sons x · a and x · b)
    build_run (x · a, r'(x · a), w, 0, 1)
    build_run (x · b, r'(x · b), w, 0, 0)
```

```
build_run (x, r'(x) = ⟨p, q, d, α⟩, w, add_l, add_r)
  if (advancing(x))
    w := tag(x) · w;
  if (add_l)
    r := r · ⟨p, w⟩;
  if (x has one son x · a)
    build_run (x · a, r'(x · a), w, 1, 0)
  if (x has two sons x · a and x · b both non advancing)
    build_run (x · a, r'(x · a), w, 0, 1)
    build_run (x · b, r'(x · b), w, 0, 0)
  if (x has two sons x · a and x · b both advancing)
    build_run (x · a, r'(x · a), w, 1, 1)
  if (add_r)
    r := r · ⟨q, w⟩;
```

Figure 1: Converting a run of A into a run of S

□

□

The emptiness of a 1ABW can be determined in linear space [EL86]. For a 1ABW with one letter alphabet, we can convert the 1ABW into a 1AWW [KV01b] and then use Theorem A.1. However, the construction in [KV01b] results in a quadratic blow up. Piterman and Vardi show that given a 1ABW as above, they can use the special structure of the automaton to construct an equivalent 1AWW with only a minor increase in the size of the transition function and the number of states [PV01b, Pit00].

Theorem A.4 *Given a 2NBP $\mathcal{S} = \langle \Sigma, P, p_0, \delta, F \rangle$ and a regular tree $\langle \Upsilon^*, \tau \rangle$ there exists a 1AWW on 1-letter alphabet $\mathcal{A} = \langle \{a\}, Q, q_0, \eta, F' \rangle$ such that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\langle \Upsilon, \tau \rangle \in \mathcal{L}(\mathcal{S})$ and $\mathcal{A}$ has $O(|P|^2 \cdot \|\tau\|)$ states and the size of the transition function is $O(|\delta| \cdot |P|^2 \cdot \|\tau\|)$.*

B Lower Bund

It was shown by [BEM97] that the problem of model checking an LTL formula with respect to a pushdown graph is EXPTIME-hard in the size of the formula. The problem is polynomial in the size of the pushdown system inducing the graph. Our algorithm for model checking an LTL formula with respect to a prefix-recognizable graph is exponential both in the size of the formula and in $|Q_\beta|$.

As prefix-recognizable systems are a generalization of pushdown systems the exponential resulting from the formula cannot be improved. We show that also the exponent resulting from Q_β cannot be removed. We use the EXPTIME-hard problem of whether a linear space alternating Turing machine accepts the empty tape [CKS81]. We reduce this question to the problem of model checking a fixed LTL formula with respect to the graph induced by a prefix-recognizable system with a constant number of states and transitions. Furthermore Q_α and Q_γ depend only on the alphabet of the Turing machine. The component Q_β does 'all the hard work'. Combining this with the above algorithm we get the following.

Theorem B.1 *The problem of model checking the graph induced by the prefix-recognizable system* $R = \langle 2^{AP}, V, Q, L, T, q_0, x_0 \rangle$ *is EXPTIME-complete in* $|Q_\beta|$.

Proof: Consider an alternating linear-space Turing machine $M = \langle \Gamma, S_u, S_e, \mapsto, s_0, F_{acc}, F_{rej} \rangle$, where the four sets of states ,S_u, S_e, F_{acc}, and F_{rej} are disjoint, and contain the universal, the existential, the accepting, and the rejecting states, respectively. We denote their union (the set of all states) by S. Our model of alternation prescribes that $\mapsto \subseteq S \times \Gamma \times S \times \Gamma \times \{L, R\}$ has a binary branching degree. When a universal or an existential state of M branches into two states, we distinguish between the left and the right branches. Accordingly, we use $(s, a) \mapsto^l (s_l, b_l, \Delta_l)$ and $(s, a) \mapsto^r (s_r, b_r, \Delta_r)$ to indicate that when M is in state $s \in S_u \cup S_e$ reading input symbol a, it branches to the left with (s_l, b_l, Δ_l) and to the right with (s_r, b_r, Δ_r). (Note that the directions left and right here have nothing to do with the movement direction of the head; these are determined by Δ_l and Δ_r.)

Let $f : \mathbb{N} \to \mathbb{N}$ be the linear function such that M uses $f(n)$ cells in its working tape in order to process an input of length n. We encode a configuration of M by a string $s\gamma_1\gamma_2 \ldots (\downarrow, \gamma_i) \ldots \gamma_{f(n)}$. That is, a configuration starts with the state of M, all its other letters are in Γ, except for one letter in $\{\downarrow\} \times \Gamma$. The meaning of such a configuration is that the jth cell in the configuration, for $1 \leq j \leq f(n)$, is labeled γ_j, the reading head points at cell i, and M is in state s. For example, the initial configuration of M is $s_0(\downarrow, b)b \cdots b$ (with $f(n) - 1$ occurrences of b's) where b stands for an empty cell. A configuration c' is a successor of configuration c if c' is a left or right successor of c. We can encode now a computation of M by a tree whose branches describe sequences of configurations of M. The computation is legal if a configuration and its successors satisfy the transition relation.

Note that though M has an existential (thus nondeterministic) mode, there is a single computation tree that describes all the possible choices of M. Each run of M corresponds to a pruning of the computation tree in which all the universal configurations have both successors and all the existential configurations have at least one successor. The run is accepting if all the branches in the pruned tree reach an accepting configuration.

In order to make sure that M does not accept the empty tape, we have to check that every legal pruning of the computation tree of M contains one rejecting branch.

Given an alternating linear-space Turing machine M as above, we construct a prefix-recognizable system R and an LTL formula φ such that $G_R \models \varphi$ iff M does not accept the empty tape. The system R has a constant number of states and rewrite rules. For every rewrite rule $\langle q, \alpha_i, \beta_i, \gamma_i, q' \rangle$ we have that the languages of the regular expressions α_i and γ_i are subsets of $\Gamma \cup (\{\downarrow\} \times \Gamma) \cup S \cup \{\epsilon\}$. The language of the regular expression β_i, can be encoded by a nondeterministic automaton whose size is linear in n. The LTL formula φ does not depend on the structure of M.

The graph induced by R has one infinite trace. This trace searches for rejecting configurations in all the pruning trees. The trace first explores the left son of every configuration. If it reaches an accepting configuration, the trace backtracks until it reaches a universal configuration for which only the left son was explored. It then goes forward again and explores under the right son of the universal configuration. If the trace returns to the root without finding such a configuration then the currently explored pruning tree is accepting. Once a rejecting configuration is reached, the trace backtracks until it reaches an existential configuration for which only the left son was explored. It then explores under the right son of the existential configuration. In this mode, if the trace backtracks all the way to the root, it means that all pruning trees were checked and that there is no accepting pruning tree for M.

Let $V = (S \times \{l, r\}) \cup \Gamma \cup (\Gamma \times \{\downarrow\})$ and let $(s, l) \cdot \sigma_1 \ldots \sigma_{f(n)} \cdot (s', d)\sigma_1^l \ldots \sigma_{f(n)}^l$ be a configuration of M and its left successor. We also set σ_0 to (s, l) and σ_0^l to (s', d). Given s, σ_i and the unique σ_j for which $\sigma_j = (\downarrow, \gamma)$ for some $\gamma \in \Gamma$, we know, by the transition relation of M, what σ_i^l should be. In addition a symbol

from $S \times \{l, r\}$ should repeat exactly every $f(n)+1$ letters. Let $next_l(\langle s, \sigma_i, \sigma_j \rangle)$ denote our expectation for σ_i^l. That is, $next_l(\langle s, (s, l), (\downarrow, \gamma_j) \rangle) = (s', d)$ where $(s, \gamma_j) \rightarrow^l (s', \gamma'_j, \alpha)$, and

$$next_l(\langle s, \gamma_i, (\downarrow, \gamma_j) \rangle) = \begin{cases} \gamma_i & j \notin \{i-1, i, i+1\} \\ \gamma_i & j = i+1 \text{ and } (s, \gamma_{i+1}) \rightarrow^l (s', \gamma'_{i+1}, R) \\ (\downarrow, \gamma_i) & j = i+1 \text{ and } (s, \gamma_{i+1}) \rightarrow^l (s', \gamma'_{i+1}, L) \\ \gamma_i & j = i-1 \text{ and } (s, \gamma_{i-1}) \rightarrow^l (s', \gamma'_{i-1}, L) \\ (\downarrow, \gamma_i) & j = i-1 \text{ and } (s, \gamma_{i-1}) \rightarrow^l (s', \gamma'_{i-1}, R) \\ \gamma'_i & i = j \text{ and } (s, \gamma_i) \rightarrow^l (s', \gamma'_i, \alpha) \end{cases}$$

The expectation $next_r(\langle s, \sigma_i, \sigma_j \rangle)$ for the letter σ_i^r, which is the ith letter in the right successor of the configuration is defined analogously (the state component is augmented with r instead of l). Consistency with $next_l$ and $next_r$ now gives us a necessary condition for a sequence in Σ^* to encode a branch in the computation tree of M.

The prefix-recognizable system starts from the initial configuration of M. It has two main modes, a *forward* mode and a *backward* mode. In forward mode, the system guesses a new configuration. The configuration is guessed one letter at a time, and this letter should match the functions $next_l$ or $next_r$. If the computation reaches an accepting configuration, this means that the currently explored pruning tree might still be accepting. The system moves to backward mode and remembers that it should explore other universal branches until it finds a rejecting state. In backward universal mode, the system starts backtracking and removes configurations. Once it reaches an universal configuration that is marked by l, it replaces the mark by r and moves to forward mode and explores the right son. If the root is reached (in backward universal mode), the computation enters a rejecting sink. If in forward mode, the system reaches a rejecting configuration, then the currently explored pruning tree is rejecting. The system moves to backward mode and remembers that it has to explore existential branches that were not explored. Hence, in backward existential mode, the system starts backtracking and removes configurations. Once it reaches an existential configuration that is marked by l, the mark is changed to r and the system returns to forward mode. If the root is reached (in backward existential mode) all pruning trees have been explored and found to be rejecting. Then the system enters an accepting sink. All that the LTL formula has to check is that there exists an infinite computation of the system and that it reaches the accepting sink. Note that the prefix-recognizable system accepts, when the alternating Turing machine rejects and vice versa.

More formally we have the LTL formula is $\diamond reject$ and the rewrite system is $R = \langle 2^{AP}, V, Q, L, T, q_0, x_0 \rangle$ where

- $AP = \{reject\}$
- $V = (S \times \{L, R\}) \cup \Gamma \cup (\{\downarrow\} \times \Gamma)$
- $Q = \{forward, backward_\exists, backward_\forall, sink_a, sink_r\}$
- $L(q, \alpha) = \begin{cases} \emptyset & q \neq sink_a \\ \{reject\} & q = sink_a \end{cases}$
- $q_0 = forward$
- $x_0 = \perp \cdot (s_0, l) \cdot (\downarrow, b) \cdot b \ldots b$

In order to define the transition relation we use the following languages.

- $L^1_{egal} = \left\{ (s,d) \cdot \Gamma^i \cdot \gamma \cdot \Gamma^j \cdot (\downarrow, \gamma') \cdot \Gamma^{f(n)-i-j-2} \cdot (s',d') \cdot \Gamma^i \cdot \gamma'' \;\middle|\; \begin{array}{l} \gamma'' = next_d(\langle s, \gamma, (\downarrow, \gamma') \rangle) \\ \text{and } d, d' \in \{l, r\} \end{array} \right\}$

  $L^2_{egal} = \left\{ (s,d) \cdot \Gamma^i \cdot (\downarrow, \gamma') \cdot \Gamma^j \cdot \gamma \cdot \Gamma^{f(n)-i-j-2} \cdot (s',d') \cdot \Gamma^{i+j+1} \cdot \gamma'' \;\middle|\; \begin{array}{l} \gamma'' = next_d(\langle s, \gamma, (\downarrow, \gamma') \rangle) \\ \text{and } d, d' \in \{l, r\} \end{array} \right\}$

  $L^3_{egal} = \left\{ (s,d) \cdot \Gamma^i \cdot (\downarrow, \gamma') \cdot \Gamma^{f(n)-i-1} \cdot (s',d') \cdot \Gamma^i \cdot \gamma'' \;\middle|\; \begin{array}{l} \gamma'' = next_d(\langle s, \gamma, (\downarrow, \gamma') \rangle) \\ \text{and } d, d' \in \{l, r\} \end{array} \right\}$

  $L^4_{egal} = \left\{ (s,d) \cdot \Gamma^i \cdot (\downarrow, \gamma') \cdot \Gamma^{f(n)-i-1} \cdot (s',d') \;\middle|\; \begin{array}{l} s' = next_d(\langle s, (s,d), (\downarrow, \gamma') \rangle) \\ \text{and } d, d' \in \{l, r\} \end{array} \right\}$

  $L_{egal} = \{\perp\} \cdot V^* \cdot (L^1_{egal} \cup L^2_{egal} \cup L^3_{egal} \cup L^4_{egal})$

  Thus, this language contains all words whose last letter is the $next_l$ or $next_r$ correct successor of the previous configuration.

- $A_{ccept} = \{\perp\} \cdot V^* \cdot \{F_{acc} \times \{l, r\}\}$

  Thus, this language contains all words whose final letter is an accepting state.

- $R_{eject} = \{\perp\} \cdot V^* \cdot \{F_{rej} \times \{l, r\}\}$

  Thus, this language contains all words whose final letter is a rejecting state.

- $R^{S_u \times \{l\}}_{emove} = \Gamma \cup (\{\downarrow\} \times \Gamma) \cup ((S \setminus S_u) \times \{l\}) \cup (S \times \{r\})$

  Thus, this language contains all the letters that are not universal states marked by l.

- $R^{S_e \times \{l\}}_{emove} = \Gamma \cup (\{\downarrow\} \times \Gamma) \cup ((S \setminus S_e) \times \{l\}) \cup (S \times \{r\})$

  Thus, this language contains all the letters that are not existential states marked by l.

Clearly the languages L_{egal}, A_{ccept}, and R_{eject} can be accepted by nondeterministic automata whose size is linear in $f(n)$.

The transition relation includes the following rewrite rules:

1. $\langle forward, \{\epsilon\}, L_{egal}, V \setminus (S \times \{r\}), forward \rangle$ - guess a new letter and put it on the store. States are guessed only with direction l. The fact that L_{egal} is used ensures that the currently guessed configuration (and in particular the previously guessed letter) is the successor of the previous configuration on the store.

2. $\langle forward, \{\epsilon\}, A_{ccept}, \{\epsilon\}, backward_\forall \rangle$ - reached an accepting configuration. Do not change the store and move to backward universal mode.

3. $\langle forward, \{\epsilon\}, R_{eject}, \{\epsilon\}, backward_\exists \rangle$ - reached a rejecting configuration. Do not change the store and move to backward existential mode.

4. $\langle backward_\forall, R^{S_u \times \{l\}}_{emove}, V^*, \{\epsilon\}, backward_\forall \rangle$ - remove one letter that is not in $S_u \times \{l\}$ from the store.

5. $\langle backward_\forall, S_u \times \{l\}, V^*, S_u \times \{r\}, forward \rangle$ - replace the marking l by the marking r and move to forward mode. The state s does not change[4].

[4] Actually, we guess all states in S_u. As we change state into $forward$, the next transition verifies that indeed the state is the same state.

6. $\langle backward_\forall, \epsilon, \{\bot\}, \epsilon, sink_r\rangle$ - when the root is reached in backward universal mode enter the rejecting sink

7. $\langle backward_\exists, R_{remove}^{S_e\times\{l\}}, V^*, \{\epsilon\}, backward_\exists\rangle$ - remove one letter that is not in $S_e \times \{l\}$ from the store.

8. $\langle backward_\exists, S_e \times \{l\}, V^*, S_e \times \{r\}, forward\rangle$ - replace the marking l by the marking r and move to forward mode. The state s does not change.

9. $\langle backward_\exists, \epsilon, \{\bot\}, \epsilon, sink_a\rangle$ - when the root is reached in backward existential mode enter the accepting sink

10. $\langle sink_a, \epsilon, \{\bot\}, \epsilon, sink_a\rangle$ - remain in accepting sink

11. $\langle sink_r, \epsilon, \{\bot\}, \epsilon, sink_r\rangle$ - remain in rejecting sink

□

C Reduction from model-checking of prefix-recognizable systems to membership of 2NBP

We prove Theorem 4.2

Proof: As before we use the NBW $\mathcal{M}_{\neg\varphi} = \langle 2^{AP}, W, \eta_{\neg\varphi}, w_0, F\rangle$.

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

- $\Sigma = V \times \Pi_{i=1}^n D_{\beta_i}$.
- $P = \{\langle w, q, s, t_i\rangle \mid w \in W,\ q \in Q,\ t_i = \langle q', \alpha_i, \beta_i, \gamma_i, q\rangle \in T,$ and $s \in Q_{\alpha_i} \cup Q_{\gamma_i}\}$

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

- $p_0 = \langle w_0, q_0, x_0, t, \rangle$ where t is an arbitrary rewrite rule.

  Thus, $\mathcal{S}$ navigates down the tree to the location x_0. There, it chooses a new rewrite rule and updates the state of $\mathcal{M}$ and the Q component accordingly.

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

$$\delta(\langle w, q, s, t\rangle, \sigma) = \begin{cases} \left\{ (\langle w, q, s', t_i\rangle, \uparrow) \;\middle|\; \begin{array}{l} t_i = \langle q', \alpha_i, \beta_i, \gamma_i, q\rangle \\ s \in \eta_{\alpha_i}(s', \sigma[0]) \end{array} \right\} \bigcup \\ \left\{ (\langle w, q, s_0, t_i\rangle, \epsilon) \;\middle|\; \begin{array}{l} t_i = \langle q', \alpha_i, \beta_i, \gamma_i, q\rangle, \\ s = q^0_{alpha_i},\ s_0 = q^0_{\gamma_i}, \\ \text{and } \sigma[i] \in F_{\beta_i} \end{array} \right\} & s \in Q_\alpha \\ \\ \left\{ (\langle w, q, s', t_i\rangle, B) \;\middle|\; \begin{array}{l} t_i = \langle q', \alpha_i, \beta_i, \gamma_i, q\rangle \\ s' \in \eta_{\gamma_i}(s, B) \text{ and } B \in V \end{array} \right\} \bigcup \\ \left\{ (\langle w', q'', s', t_{i'}\rangle, \epsilon) \;\middle|\; \begin{array}{l} t_i = \langle q', \alpha_i, \beta_i, \gamma_i, q\rangle, \\ t_{i'} = \langle q, \alpha_{i'}, \beta_{i'}, \gamma_{i'}, q''\rangle, \\ w' \in \eta_{\neg\varphi}(w, L(q, \sigma[0])), \\ s \in F_{\gamma_i} \text{ and } s' \in F_{\alpha_{i'}} \end{array} \right\} & s \in Q_\gamma \end{cases}$$

  Thus, when $s \in Q_\alpha$ the 2NBP $\mathcal{S}$ either chooses a predecessor s' of s and goes up the tree or in case s is the initial state of $\mathcal{U}_{\alpha_i}$ and $\sigma[i] \in F_{\beta_i}$ then $\mathcal{S}$ chooses the initial state $q^0_{\gamma_i}$ of $\mathcal{U}_{\gamma_i}$.

  When $s \in Q_\gamma$ the 2NBP $\mathcal{S}$ either guesses a direction B and chooses a successor s' of s reading B or in case $s \in F_{\gamma_i}$ is an accepting state of $\mathcal{U}_{\gamma_i}$, the automaton $\mathcal{S}$ updates the state of $\mathcal{M}$, chooses a new rewrite rule $t_{i'} = \langle q, \alpha_{i'}, \beta_{i'}, \gamma_{i'}, q''\rangle$ and moves to an accepting state in $F_{\alpha_{i'}}$ of $\mathcal{U}_{\alpha_{i'}}$.

- $F' = \{\langle w, q, s, t_i\rangle \mid w \in F,\ q \in Q,\ t_i = \langle q', \alpha_i, \beta_i, \gamma_i, q\rangle, \text{ and } s \in F_{\gamma_i}\}$

  Only action states may be accepting. As accepting states have no outgoing edges, in an accepting run, no navigation stage can last indefinitely.

As before we can show that a trace that violates φ and the rewrite rules used to create this trace can be used to produce a run of $\mathcal{S}$ on $\langle V^*, \tau_\beta\rangle$.

Similarly, an accepting run of $\mathcal{S}$ on $\langle V^*, \tau_\beta\rangle$ is used to find a trace in G_R that violates φ. □

D Proof that the reduction works

Claim D.1 $G_R \models \varphi$ *iff* $G_{R'} \models \varphi'$

We first need some definitions and notations. We define a partial function g from traces in $G_{R'}$ to traces in G_R. Given a trace π' in $G_{R'}$, if $\pi' \models \varphi'_1$ then $g(\pi')$ is undefined. Otherwise, denote $\pi' = (p'_0, w_0), (p'_1, w_1), \ldots$ and

$$g(\pi') = \begin{cases} (p, w_0), g(\pi'_{\geq 1}) & p'_0 = \langle p, t, ch\text{-}rule\rangle \\ g(\pi'_{\geq 1}) & p'_0 = \langle p, t, \alpha\rangle \text{ and } \alpha \neq ch\text{-}rule \end{cases}$$

Thus, g picks from π' only the configurations marked by $ch\text{-}rule$, it then takes the state from Q that marks those configurations and the store. Furthermore given two traces π' and $g(\pi')$ we define a matching between locations in π' in which the configuration is marked by $ch\text{-}rule$ and the locations in $g(\pi')$. Given a location i in $g(\pi')$ we denote by $ch(i)$ the location in π' of the i-th occurrence of $ch\text{-}rule$ along π'.

Lemma D.2 *1. For every trace π' of $G_{R'}$, $g(\pi')$ is either not defined or a valid trace of G_R.*

2. The function g is a bijection between $domain(g)$ and the traces of G_R.

3. For every trace π' of $G_{R'}$ such that $g(\pi')$ is defined, we have $(\pi', ch(i)) \models f(\varphi)$ iff $(g(\pi'), i) \models \varphi$

Proof: 1. Suppose $g(\pi')$ is defined, we have to show that it is a trace of G_R. The first pair in π' is $(\langle q_0, t, ch\text{-}rule\rangle, x_0)$. Hence $g(\pi')$ starts from (q_0, x_0). Assume by induction that the prefix of $g(\pi')$ up to location i is the prefix of some computation in G_R. We show that also the prefix up to location $i+1$ is a prefix of a computation. Let $(\langle q, t, ch\text{-}rule\rangle, x)$ be the i-th $ch\text{-}rule$ appearing in π', then the i-th location in $g(\pi')$ is (q, x). The computation of R' chooses some rewrite rule $t_i = \langle q, \alpha_i, \beta_i, \gamma_i, q'\rangle \in T$ and moves to state $\langle q', t, s\rangle$ where $s \in F_{\alpha_i}$. It must be the case that a state $\langle q', t, ch\text{-}dir\rangle$ appears in the computation of R' after location $ch(i)$. Otherwise, the computation is finite and does not interest us. The system R' can move to a state marked by $ch\text{-}dir$ only from $q^0_{\alpha_i}$, the initial state of $\mathcal{U}_{\alpha_i}$. Hence, we conclude that $x = y \cdot z$ where $y \in \alpha_i$. As not_wrong is asserted everywhere along π' we know that $z \in \beta_i$. Now R' adds a word y' in γ_i to z and reaches state $(\langle q', t', ch\text{-}rule\rangle, y' \cdot z)$. Thus, the transition t is possible also in R and can lead from $(q, y \cdot z)$ to $(q', y' \cdot z)$.

2. It is quite clear that g is an injection. As above, given a trace π in G_R we can construct the trace π' in $G_{R'}$ such that $g(\pi') = \pi$.

3. We prove that $(\pi, i) \models \varphi$ iff $(\pi', ch(i)) \models \varphi$ by induction on the structure of φ.

   - For a boolean combination of formulas the proof is immediate.
   - For a proposition $p \in AP$, it follows from the proof above that if in location i in $g(\pi')$ appears state (q, x) then in location $ch(i)$ in π' appears state $(\langle q, t, ch\text{-}rule\rangle, x)$. By definition $p \in L(q, x)$ iff $p \in L'(\langle q, t, ch\text{-}rule\rangle, x)$.
   - For a formula $\varphi = \psi_1 \mathcal{U} \psi_2$. Suppose $(g(\pi'), i) \models \varphi$. Then there exists some $j \geq i$ such that $(g(\pi'), j) \models \psi_2$ and for all $i \leq k < j$ we have $(g(\pi'), k) \models \psi_1$. By the induction assumption we have that $(\pi', ch(j)) \models f(\psi_2)$ (and clearly, $(\pi', ch(j)) \models ch\text{-}rule$), and for all $i \leq j < k$ we have $(\pi', ch(k)) \models \psi_1$. Furthermore, as every location marked by $ch\text{-}rule$ is associated by the function ch to some location in $g(\pi')$ all other locations are marked by $\neg ch\text{-}rule$. Hence, $(\pi', ch(i)) \models (ch\text{-}rule \rightarrow f(\psi_1))\mathcal{U}(f(\psi_2) \wedge ch\text{-}rule)$.
     The other direction is similar.
   - For a formula $\varphi = \bigcirc\psi$ the argument resembles the one above for $\mathcal{U}$.

□

We note that for every trace π' and $g(\pi')$ we have that $ch(0) = 0$. Claim D.1 follows immediately.

E The construction of the prefix-recognizable system

Given a pushdown system $R = \langle 2^{AP}, V, Q, T, L, q_0, x_0\rangle$ with a regular labeling function, we construct a prefix-recognizable system $R' = \langle 2^{AP'}, V, Q', T', L', q'_0, x_0\rangle$ with simple labeling as follows.

- $AP' = AP \cup \{start\}$. The proposition $start$ marks the beginning of the sequence of length $n+1$ states of $G_{R'}$ that relates to one state of G_R.

- $Q' = (Q \times AP \times \{\bot, \top\}) \cup (Q \times \{start\})$. A state (q, x) in G_R relates to the sequence that starts with $(\langle q, start\rangle, x)$, continues to $(\langle q, p_1, \lambda_1\rangle, x)$ where λ_1 is as before, and then proceeds to $(\langle q, p_2, \lambda_2\rangle, x)$ and forward until $(\langle q, p_n, \lambda_n\rangle, x)$.

- $L'((q, start), x) = \{start\}$.
  $L'((q, p, \top), x) = \{p\}$.
  $L'((q, p, \bot), x) = \emptyset$.

- $T'_0 = \{(\langle q, start\rangle, \epsilon, R_{q,p_1}, \epsilon, \langle q, p_1, \top\rangle), (\langle q, start\rangle, \epsilon, \widetilde{R}_{q,p_1}, \epsilon, \langle q, p_1, \bot\rangle) \mid q \in Q\}$
  For $1 \leq i < n$ we have
  $T'_i = \Big\{ \ (\langle q, p_i, \alpha\rangle, \epsilon, R_{q,p_{i+1}}, \epsilon, \langle q, p_{i+1}, \top\rangle), (\langle q, p_i, \alpha\rangle, \epsilon, \widetilde{R}_{q,p_{i+1}}, \epsilon, \langle q, p_{i+1}, \bot\rangle) \ \ \mid q \in Q \text{ and } \alpha \in \{\bot, \top\}\Big\}$
  $T'_n = \{(\langle q, p_n, \alpha\rangle, A, V^*, \{x\}, \langle q', start\rangle) \mid q \in Q,\ \alpha \in \{\bot, \top\},\ \textit{and}\ \langle q, A, x, q'\rangle \in T\}$
  Finally, we have $T' = \bigcup_{i=1}^{n} T'_i$. Thus, from a configuration marked by p_i we move to a state marked by p_{i+1} without changing the store. We mark p_{i+1} as true or false according to the store. From a configuration marked by p_n we apply a new rewrite rule according to the first letter in the store.

- $q'_0 = (q_0, start)$.