Relating Word and Tree Automata[*]

Orna Kupferman, Hebrew University[†]
Shmuel Safra, Tel Aviv University[‡]
Moshe Y. Vardi, Rice University[§]

August 10, 2004

Abstract

In the automata-theoretic approach to verification, we translate specifications to automata. Complexity considerations motivate the distinction between different types of automata. Already in the 60's, it was known that deterministic Büchi word automata are less expressive than nondeterministic Büchi word automata. The proof is easy and can be stated in a few lines. In the late 60's, Rabin proved that Büchi tree automata are less expressive than Rabin tree automata. This proof is much harder. In this work we relate the expressiveness gap between deterministic and nondeterministic Büchi word automata and the expressiveness gap between Büchi and Rabin tree automata. We consider tree automata that recognize *derived languages*. For a word language L, the derived language of L, denoted $L\triangle$, is the set of all trees all of whose paths are in L. Since often we want to specify that all the computations of the program satisfy some property, the interest in derived languages is clear. Our main result shows that L is recognizable by a nondeterministic Büchi word automaton but not by a deterministic Büchi word automaton iff $L\triangle$ is recognizable by a Rabin tree automaton and not by a Büchi tree automaton. Our result provides a simple explanation to the expressiveness gap between Büchi and Rabin tree automata. Since the gap between deterministic and nondeterministic Büchi word automata is well understood, our result also provides a characterization of derived languages that can be recognized by Büchi tree automata. Finally, it also provides an exponential determinization of Büchi tree automata that recognize derived languages.

1 Introduction

While *program verification* was always a desirable, but never an easy task, the advent of concurrent programming has made it significantly more necessary and difficult. The first step

[*]A preliminary version of this paper appears in the Proceedings of the 11th Symposium on Logic in Computer Science, pages 322-333, IEEE Computer Society Press, 1996.

[†]Address: School of Computer Science and Engineering, Jerusalem 91904, Israel. Email: orna@cs.huji.ac.il

[‡]Address: School of Mathematics, Tel Aviv 69978, Israel. Email: safra@math.tau.ac.il

[§]Address: Department of Computer Science, Houston, TX 77005-1892, U.S.A. Email: vardi@cs.rice.edu

in program verification is to come with a *formal specification* of the program. One of the more widely used specification languages for concurrent finite-state programs is *temporal logic* [Pnu77, MP92]. Temporal logic comes in two varieties: *linear* and *branching*. In linear temporal logics, formulas are interpreted over linear sequences and describe a behavior of a single infinite computation of a program. In branching temporal logics, formulas are interpreted over infinite trees and describe the behavior of the possible computations of a nondeterministic program. In both versions, formulas are generated with respect to a set AP of the program's atomic propositions. Each formula describes a language (of either infinite words or infinite trees) over the alphabet 2^{AP}.

Automata on infinite objects also describe languages [Tho90]. As automata on finite objects, they either accept or reject an input object. Since a run on an infinite object does not have a final state, acceptance is determined with respect to the set of states visited infinitely often during the run. For example, in the *Büchi acceptance condition*, some of the states are designated as accepting states and a run is accepting iff it visits states from the accepting set infinitely often [Büc62]. As temporal logics, automata on infinite objects come in two varieties. Automata on infinite *words* (word automata, for short) and automata on infinite *trees* (tree automata). The automata-theoretic approach to temporal logic uses the theory of automata as a unifying paradigm for program specification, verification, and synthesis [ES84, VW86a, EJ91, VW94, Kur94, KVW00]. In this paradigm, both the program and the specification are translated to (or are given as) automata. Linear temporal logic formulas correspond to word automata and branching temporal logic formulas correspond to tree automata. Then, questions about programs and their specifications can be reduced to questions about automata. More specifically, questions such as satisfiability of specifications and correctness of programs with respect to their specifications can be reduced to questions such as nonemptiness and containment of automata. These reductions yield clean and optimal algorithms and are very helpful in implementing formal verification methods [Var96].

An important factor to be considered when we examine a specification language is its ability to describe behaviors accurately. We can control the *expressive power* of temporal logics by limiting their syntax. For example, while the branching temporal logic CTL* permits an arbitrary combination of linear-time operators in its path formulas, its subset CTL restricts path formulas to have only a single linear-time operator. This restriction makes CTL less expressive than CTL* [EH86].

We can also control the expressive power of automata. One way to do it is to restrict their transition relations to be *deterministic*. Every automaton on finite words can be determinized. This is not true for automata on infinite words. In [Lan69], Landweber proved that deterministic Büchi word automata are less expressive than nondeterministic Büchi word automata. That is, he showed that there exists a language of infinite words that is recognizable by a nondeterministic Büchi word automaton but not recognizable by any nondeterministic Büchi word automaton [1]. Today, the gap between nondeterministic and deterministic Büchi word automata is well understood. While nondeterministic Büchi automata can describe any ω-regular language, deterministic Büchi automata can describe an ω-regular language L iff there exists a regular language W such that L contains exactly all words that have infinitely many prefixes in W [Lan69].

Another way to control the expressive power of automata is by defining various acceptance conditions. For example, one may wonder whether there exists an acceptance condition for which deterministic automata are as expressive as nondeterministic ones. In 1966, McNaughton answered this question to the positive. In the suggested acceptance condition, now known as the *Rabin acceptance condition*, we have a set of pairs of subsets of the states. A run is accepting iff there exists a pair $\langle G, B\rangle$ for which the run visits states from G infinitely often but visits states from B only finitely often. McNaughton showed that deterministic Rabin word automata are as expressive as nondeterministic Rabin word automata and that they are both as expressive as nondeterministic Büchi word automata [McN66]. A different picture is drawn when we consider automata on infinite trees. In 1969, Rabin showed that, though their expressive power with respect to words coincide, nondeterministic Büchi tree automata are less expressive than nondeterministic Rabin tree automata [Rab69]. That is, there exists a language of infinite trees that is recognizable by a Rabin tree automaton but not recognizable by any Büchi tree automaton.

Let us use DBW,NBW,DRW,NRW,NBT, and NRT to denote, respectively, deterministic Büchi word, nondeterministic Büchi word, deterministic Rabin word, nondeterministic Rabin word, nondeterministic Büchi tree, and nondeterministic Rabin tree automata. We sometimes refer by these notations also to the set of languages recognizable by the corresponding automata. So, for example, NBW $\setminus$ DBW denotes the set of languages that are recognizable by NBW and are not recognizable by DBW. Let us also use DBW $<$ NBW to indicate that this set is not empty; i.e., that DBW are less expressive than NBW. Summarizing the expressiveness results we have mentioned so far, we have DBW $<$ NBW $=$ DRW $=$ NRW and NBT $<$ NRT.

There is a price to expressive power. The more expressive a language is, the higher is the *complexity* of solving questions about it. For example, the complexities of the model-checking and the satisfiability problems for the logic CTL* are significantly higher than these for its less expressive subset CTL [SC85, VS85]. Similarly, while the containment problem for DBW can be solved in NLOGSPACE [WVS83, Kur87], it is PSPACE-complete for NBW [Wol82]. Finally, while the complexity of the nonemptiness problem for NBT can be solved in quadratic time [VW86b], it is NP-complete for NRT [Eme85, VS85, EJ88]. The interested readers can find more examples in [Eme90, Tho90].

In the automata-theoretic approach to verification, we translate specifications to automata. Which type of automata? The answer, obviously, should be "the weakest type that is still

[1] It is easy to see that deterministic automata on infinite trees are less expressive than their nondeterministic counterpart. Indeed, only the latter can quantify over paths existentially [TW68].

strong enough to express the required behaviors accurately". In this paper we consider tree automata that describe *derived languages*. Let L be a language of words. The derived language of L, denoted $L\triangle$, consists of all trees all of whose paths are in L. Since often we want to specify that all the computations of the program satisfy some property, the interest in derived languages is clear.

Proving that DBW $<$ NBW, Landweber showed that the language $L_1 = (0+1)^*1^\omega$ (only finitely many 0's) is in NBW $\setminus$ DBW. The proof is simple and can be stated in a few lines. Much harder is the proof that NBT $<$ NRT. In [Rab69], Rabin had to use a complicated construction and a complicated inductive argument. Interestingly, the language that Rabin used in his proof is the derived language of L_1. That is, the set of all trees all of whose paths have only finitely many 0's. In terms of temporal logics, if follows from Landweber's result that the LTL formula $FG1$ can not be translated to a DBW, and it follows from Rabin's result that the CTL* formula $AFG1$ can not be translated to a NBT.

Our main result shows that Rabin's choice of L_1 was not at all arbitrary. We prove that for every word language L, we have that $L \in$ NBW $\setminus$ DBW iff $L\triangle \in$ NRT $\setminus$ NBT. Our proof suggests an additional proof and provides a simple explanation to the expressiveness gap between Büchi and Rabin tree automata. Since the gap between DBW and NBW is well understood, it also provides a characterization of derived languages that can be described by NBT. The difficult part in the proof is to show that if $L\triangle \in$ NBT, then $L \in$ DBW. Given a Büchi tree automaton $\mathcal{U}$ that recognizes $L\triangle$, we construct a deterministic Büchi word automaton $\mathcal{A}$ that recognizes L. For $\mathcal{U}$ with n states, the automaton $\mathcal{A}$ has 2^{n+1} states. We can expand $\mathcal{A}$ in a straightforward way to a deterministic tree automaton that recognizes $L\triangle$. This suggests an exponential determinization for Büchi tree automata that recognize derived languages. We note that optimal determinization of a Büchi word automaton with n states results in a deterministic Rabin word automaton with $2^{O(n \log n)}$ states and n pairs [Mic88].

We study the problem of deciding whether the language of a given tree automaton is derivable and show that it is EXPTIME-complete. We also consider the corresponding problem in the temporal-logic paradigm, of deciding whether the set of trees that satisfy a given branching temporal logic formula is derivable. We show how our result can be used in order to obtain inexpressibility results in temporal logic and in order to check the derivability of formulas. Finally, we discuss the exponential blow-up of the construction and question its optimality.

2 Preliminaries

A *Büchi word automaton* is $\mathcal{A} = \langle \Sigma, Q, \delta, Q^0, F\rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of accepting states. Since $\mathcal{A}$ may have several initial states and since the transition function may specify many possible transitions for each state and letter, $\mathcal{A}$ may be *nondeterministic*. If $|Q^0| = 1$ and δ is such that for every $q \in Q$ and $\sigma \in \Sigma$, we have that $|\delta(q,\sigma)| \leq 1$, then $\mathcal{A}$ is a *deterministic* automaton.

Given an input word $\sigma = \sigma_0 \cdot \sigma_1 \cdots$ in Σ^ω, a *run* of $\mathcal{A}$ on σ can be viewed as a function $r : \mathbb{N} \to Q$ where $r(0) \in Q^0$ and for every $i \geq 0$, we have $r(i+1) \in \delta(r(i), \sigma_i)$; i.e., the run starts in one of the initial states and obeys the transition function. Note that a nondeterministic automaton can have many runs on σ. In contrast, a deterministic automaton has a single run on σ. For a run r, let $inf(r)$ denote the set of states that r visits infinitely often. That is,

$$inf(r) = \{q \in Q \ : \text{for infinitely many } i \geq 0, \text{we have } r(i) = q\}.$$

As Q is finite, it is guaranteed that $inf(r) \neq \emptyset$. The run r is *accepting* iff $inf(r) \cap F \neq \emptyset$. That is, iff there exists a state in F that r visits infinitely often. A run which is not accepting is *rejecting*. An automaton $\mathcal{A}$ accepts an input word σ iff there exists an accepting run of $\mathcal{A}$ on σ. The *language* of $\mathcal{A}$ is the set of all words in Σ^ω that $\mathcal{A}$ accepts.

The *infinite binary tree* is the set $T = \{0,1\}^*$. The elements of T are called *nodes*, and the empty word ϵ is the *root* of T. For every $x \in T$, the nodes $x \cdot 0$ and $x \cdot 1$ are, respectively, the left and right *successors* of x. Each node x is the root of the *subtree* T^x of T. Formally, $T^x = \{x \cdot y : y \in T\}$. The subtrees $T^{x \cdot 0}$ and $T^{x \cdot 1}$ are, respectively, the left and right subtrees of T^x. We sometimes simply say that $T^{x \cdot 0}$ is the left subtree of x. A *path* π of the tree T is a set $\pi \subset T$ such that $\epsilon \in \pi$ and for every $x \in \pi$, exactly one successor of x is in π (we denote strict containment by $\subset$). Note that each path $\pi \subset T$ corresponds to a unique word in $\{0,1\}^\omega$. For example, the leftmost path corresponds to 0^ω. For a path π and $j \geq 0$, let $\pi[j]$ denote the node of length j in π, and let π^j denote the suffix $\pi[j] \cdot \pi[j+1] \cdots$ of π. Given an alphabet Σ, a Σ-*labeled tree* is a function $V : T \to \Sigma$ that maps each node of T to a letter in Σ. We sometimes extend V to paths and use $V(\pi)$ to denote the infinite word $V(\pi[0]) \cdot V(\pi[1]) \cdot V(\pi[2]) \cdots$. We denote by $\Sigma_\mathcal{T}$ the set of all Σ-labeled trees.

Tree automata run on such Σ-labeled trees. A *Büchi tree automaton* is $\mathcal{U} = \langle \Sigma, Q, \delta, Q_0, F \rangle$, where Σ, Q, Q_0, and F, are as in Büchi word automata, and $\delta : Q \times \Sigma \to 2^{Q \times Q}$ is a (nondeterministic) transition function. Intuitively, in each of its transitions, $\mathcal{U}$ splits into two copies. One copy proceeds to the left subtree and one copy proceeds to the right subtree. A pair $\langle q_l, q_r \rangle \in \delta(q, \sigma)$ means that if $\mathcal{U}$ is now in state q and it reads the letter σ, then a possible transition is one in which the copy that proceeds to the left subtree moves to state q_l and the copy that proceeds to the right subtree moves to state q_r.

A *run* of $\mathcal{U}$ on an input Σ-labeled tree V is a Q-labeled tree r such that $r(\epsilon) \in Q^0$ and for every $x \in T$, we have that $\langle r(x \cdot 0), r(x \cdot 1) \rangle \in \delta(r(x), V(x))$. If, for instance, $r(0) = q_2$, $V(0) = a$, and $\delta(q_2, a) = \{\langle q_1, q_2 \rangle, \langle q_4, q_5 \rangle\}$, then either $r(0 \cdot 0) = q_1$ and $r(0 \cdot 1) = q_2$, or $r(0 \cdot 0) = q_4$ and $r(0 \cdot 1) = q_5$. Given a run r and a path $\pi \subset T$, we define

$$inf(r|\pi) = \{q \in Q \ : \text{for infinitely many } x \in \pi, \text{we have } r(x) = q\}.$$

A run r is accepting iff for all paths $\pi \subset T$, we have $inf(r|\pi) \cap F \neq \emptyset$. That is, iff for each path $\pi \subset T$ there exists a state in F that r visits infinitely often along π. An automaton $\mathcal{U}$ accepts

V iff there exists an accepting run of $\mathcal{U}$ on V. *Rabin word automata* are identical to Büchi word automata, only that the acceptance condition $F \subseteq 2^Q \times 2^Q$ is a set $\{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$ of pairs of subsets of Q, and a run r is accepting iff there is $1 \leq i \leq k$ such that $inf(r) \cap G_i \neq \emptyset$ and $inf(r) \cap B_i = \emptyset$. That is, there is a pair $\langle G_i, B_i\rangle$ such that r visits G_i infinitely often but visits B_i only finitely often. Similarly, *Rabin tree automata* are identical to Büchi tree automata, only that the accepting condition is $F \subseteq 2^Q \times 2^Q$ as above, and a run r is accepting iff for each path $\pi \subset T$, there is $1 \leq i \leq k$ such that $inf(r|\pi) \cap G_i \neq \emptyset$ and $inf(r|\pi) \cap B_i = \emptyset$. In the sequel, when we write tree automata, we refer to automata with any acceptance condition, thus, in particular, Büchi or Rabin automata.

Consider a tree automaton $\mathcal{U} = \langle \Sigma, Q, \delta, Q_0, F\rangle$. For $S \subseteq Q$, we denote by $\mathcal{U}^S$ the tree automaton $\langle \Sigma, Q, \delta, S, F\rangle$, i.e., $\mathcal{U}$ with S as the set of initial states, and denote by $\mathcal{U}[S]$ the set of trees accepted by $\mathcal{U}^S$. A state q of $\mathcal{U}$ is *null* iff $\mathcal{U}[\{q\}] = \emptyset$. We assume that $\mathcal{U}[Q_0] \neq \emptyset$ and eliminate all null states and all transitions that involve null states (i.e., transitions $\langle q_l, q_r\rangle$ for which either q_l or q_r is null).

For $S \subseteq Q$ and $a \in \Sigma$, we denote by $\delta_L(S, a)$ the set of states reachable from S by reading a, on the left branch, disregarding what happens on the right branch, i.e.,

$$\delta_L(S, a) = \{q_l : \text{exists } q_r \text{ such that } \langle q_l, q_r\rangle \in \bigcup_{q \in S} \delta(q, a)\}.$$

The set $\delta_R(S, a)$ is defined symmetrically for the right. For two states q and q', and $a \in \Sigma$, we say that q' is *a-reachable* from q iff $q' \in \delta_L(q, a) \cup \delta_R(q, a)$.

For a word language $L \subseteq \Sigma^\omega$, the *derived language* of L, denoted by $L\triangle$, is the set of all trees all of whose paths are labeled with words in L. Formally,

$$L\triangle = \{V \in \Sigma_\mathcal{T} : \text{all paths } \pi \subset T \text{ satisfy } V(\pi) \in L\}.$$

For a tree language X and a word language L, we say that L *derives* X iff $X = L\triangle$. We say that X is *derivable* iff there exists some word language L such that L derives X.

For a word language L and a letter a, let $L^a = \{\sigma : a \cdot \sigma \in L\}$. Let $\mathcal{U}$ be a tree automaton, S a subset of the states of $\mathcal{U}$, and let $\mathcal{U}[S] = L\triangle$. It is a good exercise to see that

$$\mathcal{U}[\delta_L(S, a)] \cup \mathcal{U}[\delta_R(S, a)] = L^a\triangle.$$

Indeed, $L^a\triangle$ contains exactly all trees that are either left or right subtrees of some tree in $L\triangle$, with root labeled a. Moreover, as $L\triangle$ is derivable, then each left subtree of some tree in $L\triangle$ is also a right subtree of some tree in $L\triangle$, and vice versa. Hence, we can strengthen the above and have

$$\mathcal{U}[\delta_L(S, a)] = \mathcal{U}[\delta_R(S, a)] = L^a\triangle.$$

What if instead taking S we would have taken some subset S' of S? Then, obviously (e.g., when $S' = \emptyset$), it might be that

$$\mathcal{U}[\delta_L(S', a)] \cup \mathcal{U}[\delta_R(S', a)] \subset L^a\triangle.$$

Also, here, though $\mathcal{U}[S]$ is derivable, it might be that $\mathcal{U}[\delta_L(S',a)] \neq \mathcal{U}[\delta_R(S',a)]$. For example, in a case where $\mathcal{U}[\delta_L(S',a)] = L^a\triangle$ but $\mathcal{U}[\delta_R(S',a)] \subset L^a\triangle$.

Let $\mathcal{U}[S] = L\triangle$. For a set $S' \subseteq S$, a letter a, and a direction $d \in \{\text{left, right}\}$, we say that S' *d-covers* $\langle S, a\rangle$, iff $\mathcal{U}[\delta_d(S',a)] = L^a\triangle$. That is, S' d-covers $\langle S, a\rangle$ iff the set of states reachable from S' by reading a on the d-branch suffices to accept all trees accepted by the set of states reachable from S by reading a, on either the left or the right branch.

Lemma 2.1 *Let $\mathcal{U}$ be a tree automaton, S a subset of the states of $\mathcal{U}$, and let $\mathcal{U}[S] = L\triangle$. Then, for every $S' \subseteq S$, and a letter a, either S' left-covers $\langle S,a\rangle$ or $S \setminus S'$ right-covers $\langle S,a\rangle$.*

Proof: If S' does not left-cover $\langle S,a\rangle$, there exists a tree $V \in L^a\triangle \setminus \mathcal{U}[\delta_L(S',a)]$. Consider all trees that have a as their root, V as the left subtree, and some tree in $L^a\triangle$ as the right subtree. All these trees are in $L\triangle$, yet none of them is in $\mathcal{U}[S']$. Hence, as $L\triangle = \mathcal{U}[S]$, they are all in $\mathcal{U}[S \setminus S']$. Therefore, since their right subtree is *an arbitrary* tree in $L^a\triangle$, it must be that $S \setminus S'$ right-covers $\langle S,a\rangle$. □

3 Determinization

Theorem 3.1 *If $L \subseteq \Sigma^\omega$ is such that $L\triangle$ is recognized by a nondeterministic Büchi tree automaton, then L is recognized by a deterministic Büchi word automaton.*

Proof: Given a nondeterministic Büchi tree automaton $\mathcal{U} = \langle \Sigma, Q, \delta, Q_0, F\rangle$ that recognizes $L\triangle$, we construct a deterministic Büchi word automaton $\mathcal{A} = \langle \Sigma, 2^Q \times \{0,1\}, \nu, \langle Q_0, 1\rangle, 2^Q \times \{1\}\rangle$ that recognizes L.

Intuitively, the states of $\mathcal{A}$ consist of subsets of the states of $\mathcal{U}$ plus a *green light* that can be either off (0) or on (1). The initial state of $\mathcal{A}$ is the set of initial states of $\mathcal{U}$ with the green light on. Below we describe the transition function ν.

We consider only states $\langle S, g\rangle$ of $\mathcal{A}$ for which $\mathcal{U}[S]$ is derivable. The initial state clearly satisfies this property and, by the definition of ν below, states that do not satisfy it are not reachable in $\mathcal{A}$ from the initial state.

For a state $q = \langle S, g\rangle$ with $S \neq \emptyset$ and $g \in \{0,1\}$, we define ν, for all $a \in \Sigma$, as follows.

- If $S \cap F$ left-covers $\langle S,a\rangle$, then $\nu(q,a) = \langle \delta_L(S \cap F, a), 1\rangle$.
- Otherwise, by Lemma 2.1, $S \backslash F$ right-covers $\langle S,a\rangle$, in which case $\nu(q,a) = \langle \delta_R(S \setminus F, a), 0\rangle$.

For a state $q = \langle \emptyset, g\rangle$ with $g \in \{0,1\}$, we define $\nu(q,a) = \emptyset$ for all $a \in \Sigma$.

That is, $\mathcal{A}$ always tries to proceed with states from F. As long as it succeeds, the green light is on. Only when states in F might not suffice, $\mathcal{A}$ proceeds with states not in F and turns the green light off. It is easy to see that $\mathcal{A}$ is deterministic. We show that it recognizes L.

Before we get to the proof we need the following definitions. In each step of $\mathcal{A}$, its run on a word $\sigma \in \Sigma^\omega$ (and let $\sigma = \sigma_0 \cdot \sigma_1 \cdots$) either *gets stuck* (in the case it is in a state $\langle \emptyset, g\rangle$), or takes a *left move* (in the case it proceeds according to a left-covering set), or takes a *right move* (in the case where it proceeds according to a right-covering set). This fixes, for any word σ on which the run does not get stuck, an infinite path $\pi_\sigma \subset T$. Precisely, for every $j > 0$, we have that $\pi_\sigma[j] = \pi_\sigma[j-1] \cdot 0$ if $\mathcal{A}$ takes a left move in its j's step, and $\pi_\sigma[j] = \pi_\sigma[j-1] \cdot 1$ if $\mathcal{A}$ takes a right move. Consider a node $x \in \pi_\sigma$. The node x has two subtrees. One subtree contains the suffix of π_σ. We say that this subtree *continues with* π_σ. The other subtree is disjoint with π_σ. We say that this subtree *quits* π_σ.

Given a word $\sigma \in \Sigma^\omega$, we first show that if $\mathcal{A}$ accepts σ, then $\sigma \in L$.

Let $r = \langle S_0, g_0 \rangle, \langle S_1, g_1 \rangle, \langle S_2, g_2 \rangle, \ldots$ be the accepting run of $\mathcal{A}$ on σ. Since r is accepting, it does not get stuck and there are infinitely many j's with $g_j = 1$. Consider the following (not necessarily binary) Q-labeled tree. The tree has a root labeled ϵ. Nodes of length 1 are labeled with states in S_0. For $i \geq 0$, the nodes of length $i+1$ have the following successors. If $\mathcal{A}$ proceeds from S_i with a left move, then nodes labeled with a state in $S_i \setminus F$ have no successors and a node labeled with a state $q \in S_i \cap F$ has as successors nodes labeled with states that are σ_i-reachable from q. In a dual way, if $\mathcal{A}$ proceeds from S_i with a right move, then nodes labeled with a state in $S_i \cap F$ have no successors and a node labeled with a state in $S_i \setminus F$ has as successors nodes labeled with states that are σ_i-reachable from it. The way we define $\mathcal{A}$ implies that the nodes of length $i+1$ are labeled with all states in S_i.

By König's lemma, we can therefore pick a sequence $r' = q_0, q_1, \ldots$ such that for all $j \geq 0$, we have that $q_j \in S_j$, q_{j+1} is σ_j-reachable from q_j, and there are infinitely many j's with $q_j \in F$. We show that there exists a tree V, accepted by $\mathcal{U}$, in which $V(\pi_\sigma) = \sigma$. As $\mathcal{U}$ recognizes $L\triangle$, this implies that $\sigma \in L$. We define V according to r', proceeding over π_σ.

For each node $\pi_\sigma[j]$ of π_σ, if the run of $\mathcal{A}$ on σ is in S_j and takes a left (right) move, let q be such that $\langle q_{j+1}, q \rangle \in \delta(q_j, \sigma_j)$ ($\langle q, q_{j+1} \rangle \in \delta(q_j, \sigma_j)$). There exists some tree in $\mathcal{U}[\{q\}]$. Our tree V has this tree as the right (left) subtree of $\pi_\sigma[j]$ (i.e. as the subtree that quits π_σ), it has $V(\pi_\sigma[j]) = \sigma_j$, and definition proceeds to $\pi_\sigma[j+1]$. It is easy to see that $\mathcal{U}$ accepts V with a run that agrees with r' over π_σ.

We now show that if $\mathcal{A}$ does not accept σ, then $\sigma \notin L$.

Let $r = \langle S_0, g_0 \rangle, \langle S_1, g_1 \rangle, \langle S_2, g_2 \rangle, \ldots$ be the rejecting run of $\mathcal{A}$ on σ. We first consider the case where there exists $j \geq 0$ for which $S_j = \emptyset$. Intuitively, the existence of such j implies that all runs of $\mathcal{U}$ on a tree with a path labeled σ eventually get stuck. For a word $\tau \in \Sigma^\omega$ and $j \geq 0$, let V_τ^j be the tree derived from $\{\tau^j\}$. We prove that for all $\tau \in L$ and for all $j \geq 0$ for which τ agrees with σ on their first j letters, we have that $V_\tau^j \in \mathcal{U}[S_j]$. The proof proceeds by induction on j as follows. Since $\mathcal{U}[S_0] = L\triangle$, then clearly, for all $\tau \in L$, we have $V_\tau^0 \in \mathcal{U}[S_0]$. Assume that the claim holds for words in L that agree with σ on their first j letters. Let $\tau \in L$ be such that τ agrees with σ on their first $j+1$ letters. By the definition of $\mathcal{A}$, we have that $\mathcal{U}[S_{j+1}]$ contains either all trees that are left subtrees in some tree in $\mathcal{U}[S_j]$ with root labeled σ_j, or all trees that are right subtrees in such a tree. Recall that $\sigma_j = \tau_j$. Hence, since V_τ^{j+1} is the left and right subtree in V_τ^j, that has a root labeled τ_j and that, by the induction hypothesis, is in

$\mathcal{U}[S_j]$, it must be that $V_\tau^{j+1} \in \mathcal{U}[S_{j+1}]$ and we are done. Assume now, by way of contradiction, that $\sigma \in L$. Then, by the above, there exists $j \geq 0$ for which both $S_j = \emptyset$ and $V_\sigma^j \in \mathcal{U}[S_j]$. This, however, is not possible.

We now consider the more intriguing case, where $S_j \neq \emptyset$ for all $j \geq 0$. We show that there exists a tree V, rejected by $\mathcal{U}$, such that $V(\pi_\sigma) = \sigma$ and all other paths are labeled with words in L. It follows that $\sigma \notin L$. We define V according to r, proceeding over π_σ. For all $j \geq 0$, we have $V(\pi_\sigma[j]) = \sigma_j$. The subtree that quits π_σ in level j is defined as follows:

- If $S_j \cap F$ left-covers $\langle S_j, \sigma_j \rangle$, we chose as the right subtree some tree in $\mathcal{U}[\delta_L(S_j \cap F, \sigma_j)]$.
- Otherwise (in which case $S_j \setminus F$ right-covers $\langle S_j, \sigma_j \rangle$), we chose as the left subtree some tree in $\mathcal{U}[\delta_R(S_j \setminus F, \sigma_j)] \setminus \mathcal{U}[\delta_L(S_j \cap F, \sigma_j)]$; i.e., a tree that causes V not to be accepted by runs r with $r(\pi_\sigma[j]) \in S_j \cap F$.

For all $j \geq 0$, we denote by V_j the subtree of $\pi_\sigma[j]$ that quits π_σ. That is, V_j is the right subtree of $\pi_\sigma[j]$ whenever $\mathcal{A}$ takes a left move and it is the left subtree of $\pi_\sigma[j]$ whenever $\mathcal{A}$ takes a right move. Since r never reach a state with $S_j = \emptyset$, it is guaranteed that for all $j \geq 0$, if $\mathcal{A}$ takes a left move, then $\mathcal{U}[\delta_L(S_j \cap F, \sigma_j)] \neq \emptyset$. In addition, since $\mathcal{A}$ takes a right move only when $S_j \cap F$ does not left-cover $\langle S_j, \sigma_j \rangle$, it is guaranteed that if $\mathcal{A}$ takes a right move, then $\mathcal{U}[\delta_R(S_j \setminus F, \sigma_j)] \setminus \mathcal{U}[\delta_L(S_j \cap F, \sigma_j)] \neq \emptyset$. Thus, in both cases, a suitable V_j exists.

By the construction, the labels along the path π_σ form the word σ. It is not hard to see that all the other paths of V are labeled with words in L. To see this, note that each such other path has some finite prefix $\sigma_0 \cdot \sigma_1 \cdots \sigma_j$ that agrees with σ and has a suffix that continues as a path in V_j. Also, by the definition of V, all the subtrees V_j that quit π_σ satisfy $V_j \in \mathcal{U}[S_{j+1}]$.

Hence, it is sufficient to prove that for all $i \geq 0$, all trees Y in $\mathcal{U}[S_i]$, and all paths $\tau \subset T$, we have that $\sigma_0 \cdot \sigma_1 \cdots \sigma_{i-1} \cdot Y(\tau) \in L$. The proof proceeds by induction on i. Since $\mathcal{U}[S_0] = L\triangle$, then clearly, all the paths in trees in $\mathcal{U}[S_0]$ are in L. Assume now that for all trees Y in $\mathcal{U}[S_i]$ and all paths $\tau \subset Y$, we have that $\sigma_0 \cdot \sigma_1 \cdots \sigma_{i-1} \cdot Y(\tau) \in L$. Let Y' be a tree in $\mathcal{U}[S_{i+1}]$. There exists a tree in $\mathcal{U}[S_i]$ such that this tree has a root labeled σ_i and has Y' as its left or right subtree. Therefore, by the induction hypothesis, all the paths $\tau \subset T$ have $\sigma_0 \cdot \sigma_1 \cdots \sigma_{i-1} \cdot (\sigma_i \cdot Y'(\tau)) \in L$, and we are done.

It remains to see that V is rejected.

Let b be a run of $\mathcal{U}$ on V and let $q_0, q_1, q_2, \ldots$ be the sequence of states that b visits along π_σ. We say that a state q_j *agrees with* ν if the following holds.

- $S_j \cap F$ left-covers $\langle S_j, \sigma_j \rangle$ and $q_j \in S_j \cap F$, or
- $S_j \cap F$ does not left-cover $\langle S_j, \sigma_j \rangle$ and $q_j \in S_j \setminus F$.

We say that a run b agrees with ν iff almost all the states along π_σ agree with ν. That is, if there exists $k \geq 0$ for which all states q_j with $j \geq k$ agree with ν.

In order to show that no run of $\mathcal{U}$ accepts V, we prove the following two claims:

Claim 1. For every run b on a tree V with $V[\pi_\sigma] = \sigma$, if b agrees with ν then b is a rejecting run.

Claim 2. If a run b accepts V, then there exist a tree V' and an accepting run b' of $\mathcal{U}$ on V', such that $V'[\pi_\sigma] = \sigma$ and b' agrees with ν.

According to the above claims, there exists no accepting run of $\mathcal{U}$ on V. Indeed, assuming that such a run exists, leads to a contradiction.

We start with Claim 1. Let b be some run on a tree V with $V[\pi_\sigma] = \sigma$, and let $q_0, q_1, \ldots$ be the sequence of states that b visits along π_σ. If b agrees with ν, then there exists $k \geq 0$ such that for every $j \geq k$, it is possible that q_j is in F only when $S_j \cap F$ left-covers $\langle S_j, \sigma_j \rangle$. That is, only in steps whose corresponding steps in r cause the green light to turn on. Since r is a rejecting run, there are only finitely many such states. Thus, a run b that agrees with ν can visit only finitely many states in F along π_σ. Hence, it is a rejecting run.

We now prove Claim 2. We first show that if b accepts V, then for every $j \geq 0$, the subtree $V^{\pi_\sigma[j]}$ is in $\mathcal{U}[S_j]$. The proof proceeds by induction on j. Since $S_0 = Q_0$, the case $j = 0$ is straightforward. Assume now that $V^{\pi_\sigma[j]} \in \mathcal{U}[S_j]$. Consider the case where $\mathcal{A}$ takes a left move. Then, $S_{j+1} = \delta_L(S_j \cap F, \sigma_j)$. Since $S_j \cap F$ left covers $\langle S_j, \sigma_j \rangle$, then all the left subtrees of trees in $\mathcal{U}[S_j]$ with root labeled σ_j are in $\mathcal{U}[S_{j+1}]$, and we are done. The case where $\mathcal{A}$ takes a right move is similar.

Consider a state q_j that appears in the run b along π_σ. If $j > 0$ and q_{j-1} agrees with ν, then, by the definition of ν, the state q_j must be in S_j. Also, q_0 is always in S_0. Therefore, if $j = 0$ or q_{j-1} agrees with ν, and q_j does not agree with ν, then one of the following holds:

- $S_j \cap F$ left-covers $\langle S_j, \sigma_j \rangle$ and $q_j \in S_j \setminus F$, or
- $S_j \cap F$ does not left-cover $\langle S_j, \sigma_j \rangle$ and $q_j \in S_j \cap F$.

Since whenever $S_j \cap F$ does not left-cover $\langle S_j, \sigma_j \rangle$ we have as V_j a tree that leaves all the states in $S_j \cap F$ "helpless" ($V_j \notin \mathcal{U}[\delta_L(S_j \cap F, \sigma_j)]$), the latter disagreement can not happen in an accepting run. Hence, if we come across a state q_j such that $j = 0$ or q_{j-1} agrees with ν, and q_j does not agree with ν, then it must be that $S_j \cap F$ left-covers $\langle S_j, \sigma_j \rangle$ and $q_j \in S_j \setminus F$. Moreover, since r is a rejecting run (and hence visits only finitely many states in which the green light is on), there are only finitely many j's for which $S_j \cap F$ left-covers $\langle S_j, \sigma_j \rangle$. Thus, there exists $k \geq 0$ such that for all $j \geq k$, we have that $S_j \cap F$ does not left-cover $\langle S_j, \sigma_j \rangle$. By the above, if $k = 0$ or if q_{k-1} agrees with ν, then so do all q_j for $j \geq k$.

Given V and b, we define V' and b' as follows. Let k be as above. If $k = 0$, then b agrees with ν, we define $b' = b$, $V' = V$, and we are done. Otherwise, consider the set S_k. It is guaranteed that $S_k \setminus F$ right-covers $\langle S_k, \sigma_k \rangle$. Let q'_k be a state in $S_k \setminus F$ for which there exist q and q' such that $\langle q', q \rangle \in \delta(q'_k, \sigma_k)$ and the right subtree of $\pi_\sigma[j]$ (the one that continues with π_σ) is in $\mathcal{U}[\{q\}]$. Since $S_k \setminus F$ right-covers $\langle S_k, \sigma_k \rangle$ and since the right subtree of $\pi_\sigma[j]$ is in $\mathcal{U}[S_{k+1}]$, it is guaranteed that such q'_k exists. The tree V' has some tree in $\mathcal{U}[\{q'\}]$ as the left subtree of

$\pi_\sigma[j]$ (instead V_k that was there in V). The run b' has $b'(\pi_\sigma[k]) = q'_k$, and it continues on the left and right subtrees with some accepting run. It is guaranteed that along the suffix π_σ^k, all the states agree with ν.

We are still not done. The run b' is not a legal run: replacing q_k with q'_k, we did not make sure that q'_k is σ_{k-1}-reachable from q_{k-1}. We now climb up π_σ and repair b' further. By definition, $q'_k \in S_k$. Therefore, there exists $q'_{k-1} \in S_{k-1}$ such that q'_k is σ_{k-1}-reachable from q'_{k-1}. Let q be such that $\langle q, q'_k \rangle \in \delta(q'_{k-1}, \sigma_k)$, in case we reach S_k with a left move, or $\langle q'_k, q \rangle \in \delta(q'_{k-1}, \sigma_k)$, in case we reach S_k with a right move. We define V'_{k-1} as some tree in $\mathcal{U}[\{q\}]$. The run b' has $b'(\pi_\sigma[k-1]) = q'_{k-1}$ and it continues on V'_{k-1} with some accepting run. Since $q'_{k-1} \in S_{k-1}$ we can go on climbing π_σ until we reach the root of V. It is easy to see that the repair results in a legal run b' that agrees with ν. Since each path of b' eventually reaches a subtree of an accepting run, b' is accepting.

$\square$

4 Relating Word and Tree Automata

Given a deterministic word automaton $\mathcal{A} = \langle \Sigma, Q, \delta, Q^0, F \rangle$, let $\mathcal{A}_t = \langle \Sigma, Q, \delta_t, Q^0, F \rangle$ be the tree automaton where for every $q \in Q$ and $a \in \Sigma$ with $\delta(q, a) = q'$, we have $\delta_t(q, a) = \langle q', q' \rangle$. Since each prefix of a word in Σ^ω corresponds to a single prefix of a run of $\mathcal{A}$, the following lemma is straightforward.

Lemma 4.1 *For every deterministic word automaton $\mathcal{A}$ and word language L, if $\mathcal{A}$ recognizes L, then $\mathcal{A}_t$ recognizes $L\triangle$.*

We note that the fact $\mathcal{A}$ is deterministic is crucial. A similar construction for a nondeterministic $\mathcal{A}$ results in $\mathcal{A}_t$ whose language may be strictly contained in $L\triangle$. The dual construction, as we shall now see, does work also for nondeterministic automata. Given a tree automaton $\mathcal{U} = \langle \Sigma, Q, \delta, Q^0, F \rangle$, we define the word automaton $\mathcal{U}_w = \langle \Sigma, Q, \delta_w, Q^0, F \rangle$, where for every $q \in Q$ and $a \in \Sigma$, we have $\delta_w(q, a) = \{q' \ : \ q' \text{ is } a\text{-reachable from } q \text{ in } \delta\}$.

Lemma 4.2 *For every tree automaton $\mathcal{U}$ and word language L, if $\mathcal{U}$ recognizes $L\triangle$, then $\mathcal{U}_w$ recognizes L.*

Proof: We first prove that if $\sigma \in L$ then $\mathcal{U}_w$ accepts σ. Let V_σ be the tree derived from $\{\sigma\}$. Since $V_\sigma \in L\triangle$, there exists an accepting run r of $\mathcal{U}$ on it. It is easy to see that each path of r suggests a legal and accepting run of $\mathcal{U}_w$ on σ. Assume now that $\mathcal{U}_w$ accepts σ. It is easy to see that then, we can construct a tree V such that V has a path labeled σ and V is accepted by $\mathcal{U}$. Hence, it must be that $\sigma \in L$. $\square$

We can now relate the expressiveness gap between NRT and NBT and the one between NBW and DBW.

Theorem 4.3 *For every word language language L,*

$$L \in NBW \setminus DBW \Leftrightarrow L\triangle \in NRT \setminus NBT.$$

Proof: We prove the following four claims. The $\Rightarrow$ direction follows from the first two claims and the $\Leftarrow$ direction follows from the last two.

1. $L \in$ NBW $\Rightarrow L\triangle \in$ NRT.
2. $L\triangle \in$ NBT $\Rightarrow L \in$ DBW.
3. $L\triangle \in$ NRT $\Rightarrow L \in$ NBW
4. $L \in$ DBW $\Rightarrow L\triangle \in$ NBT.

Lemma 4.1 implies Claim 4. Also, as NBW=DRW, the lemma implies Claim 1 too. Claim 3 follows from Lemma 4.2 and the fact that NBW=NRW. Finally, Claim 2 follows from Theorem 3.1. $\square$

5 Derivability and Expressiveness

Our results in Section 4 consider derivable languages and tree automata that recognize derivable language. In this section we study the problem of deciding whether the language of a given tree automaton is derivable. We also consider branching temporal logics and formulas that define derivable languages.

Theorem 5.1 *For a Büchi tree automaton $\mathcal{U}$, checking whether $\mathcal{L}(\mathcal{U})$ is derivable is EXPTIME-complete.*

Proof: We start with the upper bound. Let $\mathcal{U}$ be an NBT with n states and let $\mathcal{A}$ be the DBW constructed from $\mathcal{U}$ in Theorem 3.1. The size of $\mathcal{A}$ is $2^{O(n)}$. We claim that $\mathcal{L}(\mathcal{U}) = \mathcal{L}(\mathcal{A})\triangle$ iff $\mathcal{L}(\mathcal{U})$ is derivable. First, if $\mathcal{L}(\mathcal{U}) = \mathcal{L}(\mathcal{A})\triangle$, then $\mathcal{L}(\mathcal{A})$ derives $\mathcal{L}(\mathcal{U})$, thus $\mathcal{L}(\mathcal{U})$ is derivable. Also, by Theorem 3.1, if $\mathcal{L}(\mathcal{U})$ is derivable, then $\mathcal{L}(\mathcal{U}) = \mathcal{L}(\mathcal{A})\triangle$. So, checking the derivability of $\mathcal{U}$ can be reduced to checking the equivalence of $\mathcal{L}(\mathcal{U})$ and $\mathcal{L}(\mathcal{A})\triangle$. This involves two checks:

1. $\mathcal{L}(\mathcal{U}) \subseteq \mathcal{L}(\mathcal{A})\triangle$. Let $\mathcal{C}$ be an NBW that complements $\mathcal{A}$. That is, $\mathcal{L}(\mathcal{C}) = \overline{\mathcal{L}(\mathcal{A})}$. By [Kur87], the size of $\mathcal{C}$ is $2^{O(n)}$. We can expand $\mathcal{C}$ to an NBT $\mathcal{B}$ that accepts a tree $\langle T, V\rangle$ iff there exists a path $\pi \subseteq T$ such that $V(\pi) \in \mathcal{L}(\mathcal{C})$. (in each transition, $\mathcal{B}$ guesses both its next state and the direction in the tree to which it proceed). The size of $\mathcal{B}$ is also $2^{O(n)}$, and $\mathcal{L}(\mathcal{B}) = \overline{\mathcal{L}(\mathcal{A})\triangle}$. It follows that $\mathcal{L}(\mathcal{U}) \subseteq \mathcal{L}(\mathcal{A})\triangle$ iff the intersection $\mathcal{L}(\mathcal{U}) \cap \mathcal{L}(\mathcal{B})$ is empty. Let $\mathcal{U} \times \mathcal{B}$ be the product of $\mathcal{U}$ and $\mathcal{B}$, thus $\mathcal{L}(\mathcal{U} \times \mathcal{B}) = \mathcal{L}(\mathcal{U}) \cap \mathcal{L}(\mathcal{B})$. By [VW86b], it is possible to construct $\mathcal{U} \times \mathcal{B}$ as an NBT of size $|\mathcal{U}| \cdot |\mathcal{B}|$. Since the nonemptiness problem for NBT can be solved in quadratic time [VW86b], the check can be performed in time exponential in n.

2. $\mathcal{L}(\mathcal{A})\triangle \subseteq \mathcal{L}(\mathcal{U})$. Consider the DBW $\mathcal{A}$. We can expand $\mathcal{A}$ to an NBT $\mathcal{B}$ that accepts $\mathcal{L}(\mathcal{A})\triangle$ (in each transition, $\mathcal{B}$ proceeds to its next state in all directions, thus $\mathcal{B}$ is really deterministic). The size of $\mathcal{B}$ is $2^{O(n)}$. Let $\mathcal{C}$ be a Streett tree automaton that complements $\mathcal{U}$. That is, $\mathcal{L}(\mathcal{C}) = \overline{\mathcal{L}(\mathcal{U})}$. By [MS95], the automaton $\mathcal{C}$ has $2^{O(n \log n)}$ states and $O(n)$ pairs[2]. It follows that $\mathcal{L}(\mathcal{A})\triangle \subseteq \mathcal{L}(\mathcal{U})$ iff the intersection $\mathcal{L}(\mathcal{B}) \cap \mathcal{L}(\mathcal{C})$ is empty. Let $\mathcal{D}$ be the product $\mathcal{B}$ and $\mathcal{C}$. That is $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{B}) \cap \mathcal{L}(\mathcal{C})$. The automaton $\mathcal{D}$ is a Streett automaton with $2^{O(n \log n)}$ states and $O(n)$ pairs (each state of $\mathcal{D}$ is a pair $\langle q_{\mathcal{B}}, q_{\mathcal{C}} \rangle$, thus $\mathcal{D}$ follows $\mathcal{B}$ in its first element and follows $\mathcal{C}$ in its second element. The acceptance condition of $\mathcal{D}$ adds to that of $\mathcal{C}$ a new pair imposing infinitely many visits in the accepting set of $\mathcal{B}$). Since the nonemptiness problem for nondeterministic Streett tree automata can be solved in time polynomial in the number of states and exponential in the number of pairs [EJ88, PR89, KV98b], the check can be performed in time exponential in n.

For the lower bound, we do a reduction from alternating linear-space Turing machines. Given a machine T, we construct an NBT $\mathcal{U}$, of size linear in T, such that $\mathcal{L}(\mathcal{U})$ is derivable iff the machine T does not accept the empty tape. For that, we define $\mathcal{U}$ to accept an input tree iff the tree does not represent an accepting computation tree of T on the empty tape (we assume that once T reaches a final configuration it looks there forever, which is why we can handle infinite trees). In Appendix A, we describe the construction of $\mathcal{U}$ in detail. The construction is similar to a construction described in [Sei90] for automata on finite words. Let Σ be the alphabet of $\mathcal{U}$. We prove two claims:

1. The machine T rejects the empty tape iff $\mathcal{U}$ is universal. Indeed, T rejects the empty tape iff there exists no tree that represents an accepting computation tree of T on the empty tape, which holds iff $\mathcal{U}$ accepts all trees.

2. The NBT $\mathcal{U}$ is universal iff $\mathcal{L}(\mathcal{U})$ is derivable. Clearly, if $\mathcal{U}$ is universal, then $\mathcal{L}(\mathcal{U})$ is derivable. For the other direction, assume that $\mathcal{L}(\mathcal{U})$ is derivable. Then, $\mathcal{L}(\mathcal{U}) = L\triangle$ for some $L \subseteq \Sigma^\omega$. We claim that $L = \Sigma^\omega$, in which case $\mathcal{U}$ is universal. Consider a word $w \in \Sigma^\omega$. Clearly, w can be extended to a tree that does not represent an accepting computation tree of T on the empty tape. Hence, w can be extended to a tree accepted by $\mathcal{U}$. Since $\mathcal{L}(\mathcal{U}) = L\triangle$, it must be that $w \in L$.

From the two claims it follows that T rejects the empty tape iff $\mathcal{L}(\mathcal{U})$ is derivable. □

Recall that automata are used in order to specify the behaviors of reactive systems. If the language of a tree automaton is derivable, the specification it induces is really a linear

[2] The Streett acceptance condition is dual to the Rabin acceptance condition. That is, $F \subseteq 2^Q \times 2^Q$, and a run r is accepting if for each path π of the tree and each pair $\langle G_i, B_i \rangle$ in F, either r visits G_i only finitely often along π, or r visits B_i infinitely often along π. The automaton $\mathcal{C}$ is obtained by first dualizing $\mathcal{U}$ to an alternating co-Büchi automaton (this involves no blow up), and then removing the alternation, which involves an exponential blow up in the number of states and a linear blow up in the number of pairs.

specification, and it refers to the system's set of computations, ignoring its tree structure. Often, specifications are given as formulas in temporal logics. We say that a formula ψ of the branching temporal logic CTL* is derivable iff the set of trees that satisfy ψ is derivable. Derivable formulas constitute a strict fragment of the universal fragment $\forall$CTL* of CTL* [GK94]. By [GK94], a CTL* formula is derivable (called *strongly linear* in [GK94]) iff it is equivalent to some formula of the linear temporal logic LTL. By [CD88], a CTL* formula is equivalent to some LTL formula iff it is equivalent to the LTL formula obtained by eliminating its path quantifiers. Given a CTL* formula ψ, let ψ_{lin} be the LTL formula obtained from ψ by eliminating its path quantifiers. By the above, ψ is derivable iff the CTL* formula $\psi \leftrightarrow A\psi_{lin}$ is valid. Since validity of CTL* is 2EXPTIME-complete [ES84, VS85, EJ88], this suggests a 2EXPTIME upper bound for the problem of deciding whether a given CTL* formula is derivable.

Symbolic model-checking, which enables the verification of large systems, proceeds by calculating fixed-point expressions over the system's set of states. The μ-calculus is a branching-time temporal logic with fixed-point operators. As such, it is a convenient logic for symbolic model-checking tools. In particular, the alternation-free fragment of μ-calculus (AFMC [EL86]) has a restricted syntax, making the symbolic evaluation of its formulas computationally easy. On the other hand, specifiers find the μ-calculus inconvenient, and they often prefer to use linear-time formalisms. Such formalisms, however, cannot in general be translated to AFMC, and their symbolic evaluation involves nesting of fixed-points. Our results can be used to obtain simple proofs for inexpressibility results for temporal logics. It is known, for example, that formulas of AFMC can be translated to NBT [VW86b, KVW00]. As the LTL formula FGp can not be translated to a DBW, it follows from Theorem 4.3 that the CTL* formula $AFGp$ can not be expressed in AFMC. Previous proofs of this inexpressibility result follows from the work in [Rab70, MSS86, KVW00] and are complicated. More generally, our results here are used in [KV98a] in order to show that a linear-time property can be specified in AFMC iff it can be recognized by a deterministic Büchi automaton.

6 The Translation Blow-up

In Section 3, we constructed, given a nondeterministic Büchi tree automaton $\mathcal{U}$ that recognizes $L\triangle$, a deterministic Büchi word automaton $\mathcal{A}$ that recognizes L. For $\mathcal{U}$ with n states, the automaton $\mathcal{A}$ has 2^{n+1} states. Our motivation was the study of the expressive power of automata and we ignored the complexity of the construction. In this section we question the optimality of the construction and conjecture that a better, perhaps even linear, construction is possible. For that, we show that in the case of finite trees with arbitrary branching degrees, a linear construction is possible. Note that an existence of a linear construction would imply that, in the context of derivable languages, nondeterminism adds no power to tree automata, neither in terms of expressiveness nor in terms of succinctness.

We first adjust the definition of derivable languages to the case of finite trees with an arbitrary branching degree. For $k \geq 1$, let $[k] = \{1, \ldots, k\}$. A finite tree with branching degrees at most k is a finite prefix closed set $T \subseteq [k]^*$. A Σ-labeled tree is $\langle T, V \rangle$, where T is a tree as above and $V : T \to \Sigma$ maps each node of T to a letter in Σ. Given a regular language $R \subseteq \Sigma^*$ and an integer $k \geq 1$, the language $R^k\triangle$ consists of all the Σ-labeled trees with branching degrees at most k all of whose paths are labeled by words in R. Finally, $R\triangle = \bigcup_{k \geq 1} R^k\triangle$.

Let $R \subseteq \Sigma^*$ be a regular language, $k \geq 1$, and let $\mathcal{U}$ be a nondeterministic tree automaton that recognizes $R^k\triangle$. Since $\mathcal{U}$ recognizes a derivable language, we can assume that its transition relation is closed under symmetry; that is, if $\langle q_1, q_2 \rangle \in \delta(q, \sigma)$, then $\{\langle q_1, q_1 \rangle, \langle q_2, q_2 \rangle, \langle q_2, q_1 \rangle\} \subseteq \delta(q, \sigma)$ (since we assume that $\mathcal{U}$ has no empty states, we can close it to symmetry if this is not the case). Accordingly, it is convenient to define nondeterministic tree automata recognizing derivable languages as having a transition function $\delta : Q \times \Sigma \to 2^{2^Q}$, where a set $S \in \delta(q, \sigma)$ stands for all the tuples with elements in S. Thus, when $\mathcal{U}$ runs on a tree $\langle T, V \rangle$ and it visits a node x with $V(x) = \sigma$ at state q, then $\delta(q, \sigma) = \{S_1, \ldots, S_m\}$ means that $\mathcal{U}$ should chose a set S_i and send to all the successors of x copies in states in S_i. Formally, a *run* of $\mathcal{U}$ on $\langle T, V \rangle$ is a Q-labeled tree $\langle T, r \rangle$, where $r(\epsilon) \in Q_0$, and for all $x \in T$, there is $S \in \delta(r(x), V(x))$ such that for all $d \in [k]$ with $x \cdot d \in T$, we have $r(x \cdot d) \in S$.

Myhill and Nerode study optimization of deterministic automata on finite words. Let R be a regular language over an alphabet Σ. We say that two words x and y in Σ^* are R-equivalent ($x \sim_R y$) iff for every word $z \in \Sigma^*$, we have $x.z \in R$ iff $y.z \in R$. Let $\langle R \rangle$ be the set of equivalence classes of the relation $\sim_R$. Myhill and Nerode show that the smallest deterministic automaton that recognizes R has $|\langle R \rangle|$ states. This result is constructive. Namely, given a nondeterministic automaton for R, we know how to construct a deterministic automaton with $|\langle R \rangle|$ states for R.

Theorem 6.1 *Let $R \subseteq \Sigma^*$ be a regular language, and let $k = |\langle R \rangle|$. If $R^k\triangle$ is recognized by a nondeterministic tree automaton with n states, then R can be recognized by a deterministic word automaton with $|\Sigma| \cdot n$ states.*

Proof: Let $\mathcal{U}$ be a nondeterministic tree automaton that recognizes $R^k\triangle$ with n states. We prove that $k \leq |\Sigma| \cdot n$, implying that R can be recognized by a deterministic word automaton with at most $|\Sigma| \cdot n$ states. Recall that $k = |\langle R \rangle|$. So, there are k heads (words in Σ^*) $h_1, \ldots, h_k$ such that for each pair $1 \leq i < j \leq k$, there either exists a tail (a word in Σ^*) $t_{i,j}$ such that $h_i \cdot t_{i,j} \in R$ and $h_j \cdot t_{i,j} \notin R$, or exists a tail $t_{j,i}$ such that $h_i \cdot t_{j,i} \notin R$ and $h_j \cdot t_{j,i} \in R$. For every $1 \leq i \leq k$, let T_i be the set of tails $t_{i,j}$ defined above (Note that T_i can have any size from 0 to $k-1$). Consider the tree T described in Figure 1. From the root, T branches to k paths, with the i'th path labeled by h_i. At the end of h_i, the tree T branches to a subtree that contains paths labeled by exactly all tails in T_i. Clearly, all the words labeling paths of T are in R. Thus, $T \in R^k\triangle$ and $\mathcal{U}$ has an accepting run r on T. For all $1 \leq i \leq k$, let $r(i)$ be the state $\mathcal{U}$ in r when it reads the last letter of h_i.

Assume, by way of contradiction, that $k > |\Sigma| \cdot n$. Then, there must be two heads h_j and h_l,

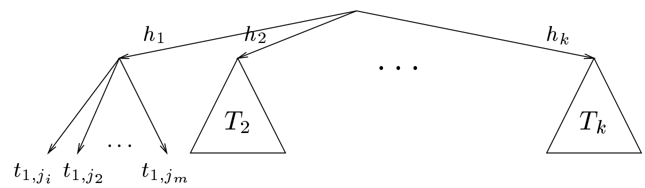

Figure 1: The tree T.

such that h_j and h_l agree on their last letter and $r(j) = r(l)$. Assume, without less of generality, that T_j contains the word $t_{j,l}$. Consider the tree T' obtained from T by switching T_j and T_l. The tree T' contains the word $h_l \cdot t_{j,l}$, which is not in R. On the other hand, the run r' of $\mathcal{U}$ on T' that is obtained by switching the subtrees T_j and T_l of r is a legal and accepting run of $\mathcal{U}$ on T'. If follows that $\mathcal{U}$ accepts a tree not in $R^k\triangle$ and we reach a contradiction. $\square$

The proof of Theorem 6.1 make use of the fact that every regular language R has a minimal deterministic automaton, whose states are the equivalence classes of the relation $\sim_R$. This helpful fact does not hold for deterministic Büchi automata, and Theorem 6.1 cannot be applied. A Büchi automaton $\mathcal{U}$ is *weak* [MSS86] if each strongly connected component in the transition graph of $\mathcal{U}$ is either contained in F or is disjoint from F. Note that a run of an automaton eventually gets trapped in some strongly connected component. Thus, in a weak automaton, a run is accepting if this component is contained in F and is rejecting if the component is disjoint from F. Deterministic weak automata are used in reasoning about real numbers [BJW01] and enjoy several combinatorial properties that deterministic Büchi automata do not have (c.f., [MS97, KMM04]). In particular, it is shown in [Löd01] that the analysis of Myhill and Nerode apply to deterministic weak automata. Thus, like finite automata, every language R that is recognized by a deterministic weak automaton has a minimal deterministic automaton whose states are the equivalence classes of the relation $\sim_R$. It is then shown in [Mor03] that Theorem 6.1 applies also to weak automata.

Determinizing a nondeterministic automaton on finite words may involve an exponential blow up. Thus, there is a regular language R such that the translation from an NFW for R to an NFW for R is exponential. By Lemma 4.2, an NFT for $R^k\triangle$ induces an NFW for R of the same size. Hence, it follows from Theorem 6.1 that there is a language R such that the translation from an NFW for R to an NBT for $R^k\triangle$ is exponential. Let \$ be a letter not in Σ, and let $R \cdot \$^\omega$ be the ω-regular language obtained by appending $\$^\omega$ to words in R. An NFW for R induces an NBW for $R \cdot \$^\omega$ with the same number of states. Also, an NBT for $(R \cdot \$^\omega)^k\triangle$ induces an NFT for $R^k\triangle$ with the same number of states. Hence, there is an ω-regular language L such that the translation from an NBW for L to an NBT for $L^k\triangle$ is exponential.

So, our conjecture that the construction can be improved is supported by two evidences. The first is that such an improved construction is possible for the case of finite trees and even trees accepted by a weak automaton, and the second is that the translation from an NBW for L to an NBT for $L^k\triangle$ may be exponential, just like the translation from NBW to DBW. Thus, the computational advantage of nondeterminism in the case of derivable languages is not clear.

7 Related and Future Work

In a recent related work [NW98], Niwinski and Walukiewicz relate hierarchies of deterministic word automata with hierarchies of nondeterministic tree automata. They consider *parity* and *co-parity* automata. In parity automata, the acceptance condition is a set $\{G_1, G_2, \ldots, G_k\}$ with $G_1 \subset G_2 \subset \ldots \subset G_k = Q$, and a run r satisfies the acceptance condition iff the minimal $1 \leq i \leq k$ for which $inf(r) \cap G_i \neq \emptyset$ is even. In co-parity automata, the minimal i is required to be odd. The number k of sets in the acceptance condition is called the *index* of the automaton. Languages of words and trees can be characterized by the minimal index of a parity or the co-parity automaton that recognizes them. It is proven in [NW98], that a word language L can be recognized by a deterministic parity word automaton of index k iff the tree language $L\triangle$ can be recognized by a nondeterministic parity tree automaton of index k, and similarly for co-parity automata. In particular, it follows that $L \in DBW$ iff $L\triangle \in NBT$.

The proof in [NW98] considers the structure of the deterministic automata that recognize L. The authors show that whenever L cannot be recognized by a deterministic parity word automaton of a certain index k, the transition graph of a deterministic automaton that does recognize L contains some special subgraph (called *flower* in [NW98]). This subgraph prevents nondeterministic tree automata of index k from recognizing $L\triangle$.

Thus, unlike our proof, the proof in [NW98] does not involve a direct construction of a deterministic word automaton for L. An interesting open problem is to combine the two works and extend our construction in Theorem 3.1 so that, given a nondeterministic parity tree automaton of index k for $L\triangle$, it would construct deterministic parity word automaton of index k for L.

References

[BJW01] B. Boigelot, S. Jodogne, and P. Wolper. On the use of weak automata for deciding linear arithmetic with integer and real variables. In *Proc. International Joint Conference on Automated Reasoning (IJCAR)*, volume 2083 of *Lecture Notes in Computer Science*, pages 611–625, Siena, June 2001. Springer-Verlag.

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method. and Philos. Sci. 1960*, pages 1–12, Stanford, 1962. Stanford University Press.

[CD88] E.M. Clarke and I.A. Draghicescu. Expressibility results for linear-time and branching-time logics. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Proc. Workshop on*

Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency, volume 354 of *Lecture Notes in Computer Science*, pages 428–437. Springer-Verlag, 1988.

[EH86] E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *Journal of the ACM*, 33(1):151–178, 1986.

[EJ88] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 328–337, White Plains, October 1988.

[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.

[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.

[Eme85] E.A. Emerson. Automata, tableaux, and temporal logics. In *Proc. Workshop on Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 79–87. Springer-Verlag, 1985.

[Eme90] E.A. Emerson. Temporal and modal logic. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 997–1072. Elsevier, MIT Press, 1990.

[ES84] E.A. Emerson and A. P. Sistla. Deciding branching time logic. In *Proc. 16th ACM Symp. on Theory of Computing*, Washington, April 1984.

[GK94] O. Grumberg and R.P. Kurshan. How linear can branching-time be. In *Proc. 1st International Conference on Temporal Logic*, volume 827 of *Lecture Notes in Artificial Intelligence*, pages 180–194, Bonn, July 1994. Springer-Verlag.

[KMM04] O. Kupferman, G. Morgenstern, and A. Murano. Typeness for ω-regular automata. In *2nd International Symposium on Automated Technology for Verification and Analysis*, Lecture Notes in Computer Science. Springer-Verlag, 2004.

[Kur87] R.P. Kurshan. Complementing deterministic Büchi automata in polynomial time. *Journal of Computer and System Science*, 35:59–71, 1987.

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

[KV98a] O. Kupferman and M.Y. Vardi. Freedom, weakness, and determinism: from linear-time to branching-time. In *Proc. 13th IEEE Symp. on Logic in Computer Science*, pages 81–92, June 1998.

[KV98b] O. Kupferman and M.Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th ACM Symp. on Theory of Computing*, pages 224–233, Dallas, 1998.

[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.

[Lan69] L.H. Landweber. Decision problems for ω–automata. *Mathematical Systems Theory*, 3:376–384, 1969.

[Löd01] C. Löding. Efficient minimization of deterministic weak omega-automata. *Information Processing Letters*, 79(3):105–109, 2001.

[McN66] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

[Mic88] M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.

[Mor03] G. Morgenstern. Expressiveness results at the bottom of the ω-regular hierarchy. M.Sc. Thesis, The Hebrew University, 2003.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, Berlin, January 1992.

[MS95] D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141:69–107, 1995.

[MS97] O. Maler and L. Staiger. On syntactic congruences for ω-languages. *Theoretical Computer Science*, 183(1):93–112, 1997.

[MSS86] D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th International Colloquium on Automata, Languages and Programming*, volume 226 of *Lecture Notes in Computer Science*. Springer-Verlag, 1986.

[NW98] D. Niwinski and I. Walukiewicz. Relating hierarchies of word and tree automata. In *Symposium on Theoretical Aspects in Computer Science*, volume 1373 of *Lecture Notes in Computer Science*. Springer Verlag, 1998.

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

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, Austin, January 1989.

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

[Rab70] M.O. Rabin. Weakly definable relations and special automata. In *Proc. Symp. Math. Logic and Foundations of Set Theory*, pages 1–23. North Holland, 1970.

[SC85] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic. *Journal ACM*, 32:733–749, 1985.

[Sei90] H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424–437, 1990.

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 165–191, 1990.

[TW68] J.W. Thatcher and J.B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical System Theory*, 2:57–81, 1968.

[Var96] M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, Berlin, 1996.

[VS85] M.Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc 17th ACM Symp. on Theory of Computing*, pages 240–251, 1985.

[VW86a] 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.

[VW86b] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Science*, 32(2):182–221, April 1986.

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

[Wol82] P. Wolper. *Synthesis of Communicating Processes from Temporal Logic Specifications*. PhD thesis, Stanford University, 1982.

[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 Derivability Lower Bound

Consider an alternating linear-space Turing machine $T = \langle \Gamma, Q_u, Q_e, \mapsto, q_0, F_{acc}, F_{rej} \rangle$, where the four sets of states, Q_u, Q_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 Q. Our model of alternation prescribes that $\mapsto \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L, R\}$ has a binary branching degree, is universal in its even-numbered steps, and is existential in its odd-numbered ones. Thus, $\mapsto$ is really a subset of $(Q_e \times \Gamma \times Q_u \times \Gamma \times \{L, R\}) \cup (Q_u \times \Gamma \times Q_e \times \Gamma \times \{L, R\})$. In particular, $q_0 \in Q_e$. When a universal or an existential state of T branches into two states, we distinguish between the left and the right branches. Accordingly, we use $(q, a) \mapsto^l (q_l, b_l, \Delta_l)$ and $(q, a) \mapsto^r (q_r, b_r, \Delta_r)$ to indicate that when T is in state $q \in Q_u \cup Q_e$ reading input symbol a, it branches to the left with (q_l, b_l, Δ_l) and to the right with (q_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.) Finally, we assume that once T reaches a final state, it loops there forever.

Let $s : \mathbb{N} \to \mathbb{N}$ be the linear function such that T uses $s(n)$ cells in its working tape in order to process an input of length n. We encode a configuration of T by a string $\#\gamma_1\gamma_2 \ldots (q, \gamma_i) \ldots \gamma_{s(n)}$. That is, a configuration starts with $\#$, and all its other letters are in Γ, except for one letter in $Q \times \Gamma$. The meaning of such a configuration is that the j'th cell in the configuration, for $1 \le j \le s(n)$, is labeled γ_j, the reading head points at cell i, and T is in state q. For example, the initial configuration of T is $\#(q_0, b)b \ldots b$ (with $s(n) - 1$ 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 a right successor of c. We can now encode a computation of T by a tree whose branches describe sequences of configurations of T. The computation is legal if a configuration and its successors satisfy the transition relation.

Note that though T has an existential (thus nondeterministic) mode, there is a single computation tree that describes all the possible choices of T. Each run of T 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.

Given an alternating linear-space Turing machine T as above, we construct an NBT $\mathcal{U}$ of size linear in T, such that $\mathcal{U}$ accepts an input tree iff the tree is not the (single) computation tree of T on the empty tape, or if it is the computation tree, but it cannot be pruned legally to a tree in which all branches are accepting. The construction is similar to the one described in [Sei90] for automata that run on finite trees and is given here in detail, adapted to Büchi automata.

We now state the condition above formally. For that, we first settle a technical difficulty, namely, the fact that while the computation tree of T branches only at the end of configurations, the input tree to $\mathcal{U}$ branches everywhere. For example, the computation tree of T starts with an encoding of the initial configuration and the first branching occurs only at level $s(n)+1$, where the tree branches to two nodes labeled $\#$, which starts the encoding of the left and right successors of the initial configuration. On the other hand, the input tree to $\mathcal{U}$, which represents this computation tree, branches everywhere, and the encoding of the initial configuration is repeated $2^{s(n)+1}$ times, along all the branches that reach the $2^{s(n)+1}$ nodes at level $s(n)+1$. Then, all these copies reach their first "real" branching, where they all branch to the left and right successors of the initial configuration. The fact that there is a single computation tree that describes all the possible choices of T makes the handling the fact that the input tree branches everywhere easy, as branching within a configuration leads to equivalent successors. So, we distinguish between two types of branching in the input tree. Branching at the end of a configuration is referred to as real branching, and branching within a configuration is referred to as virtual branching. Only in real branching, there may be a difference between the left and right successors. Checking whether the input tree represents the full computation tree of T refers to all branches of the input tree. On the other hand, checking whether the computation tree can be pruned as required considers only pruning in real branching.

Let $\Sigma = \{\#\} \cup \Gamma \cup (Q \times \Gamma)$ and let $\#\sigma_1 \ldots \sigma_{s(n)}\#\sigma_1^l \ldots \sigma_{s(n)}^l$ be a configuration of T and its left successor. We also set σ_0, σ_0^l, and $\sigma_{s(n)+1}$ to $\#$. For each triple $\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle$ with $1 \leq i \leq s(n)$, we know, by the transition relation of T, what σ_i^l should be. In addition, the letter $\#$ should repeat exactly every $s(n)+1$ letters. Let $next_l(\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle)$ denote our expectation for σ_i^l. That is,

- $next_l(\langle \gamma_{i-1}, \gamma_i, \gamma_{i+1} \rangle) = next_l(\langle \#, \gamma_i, \gamma_{i+1} \rangle) = next_l(\langle \gamma_{i-1}, \gamma_i, \# \rangle) = \gamma_i$.
- $next_l(\langle (q, \gamma_{i-1}), \gamma_i, \gamma_{i+1} \rangle) = next_l(\langle (q, \gamma_{i-1}), \gamma_i, \# \rangle) =$

$$\begin{cases} \gamma_i & \text{If } (q,\gamma_{i-1}) \rightarrow^l (q',\gamma'_{i-1},L) \\ (q',\gamma_i) & \text{If } (q,\gamma_{i-1}) \rightarrow^l (q',\gamma'_{i-1},R) \end{cases}$$

- $next_l(\langle\gamma_{i-1},(q,\gamma_i),\gamma_{i+1}\rangle) = next_l(\langle\#,(q,\gamma_i),\gamma_{i+1}\rangle) =$
  $next_l(\langle\gamma_{i-1},(q,\gamma_i),\#\rangle) = \gamma'_i$ where $(q,\gamma_i) \rightarrow^l (q',\gamma'_i,\Delta)$ [3].

- $next_l(\langle\gamma_{i-1},\gamma_i,(q,\gamma_{i+1})\rangle) = next_l(\langle\#,\gamma_i,(q,\gamma_{i+1})\rangle) =$

$$\begin{cases} \gamma_i & \text{If } (q,\gamma_{i+1}) \rightarrow (q',\gamma'_{i+1},R) \\ (q',\gamma_i) & \text{If } (q,\gamma_{i+1}) \rightarrow (q',\gamma'_i,L) \end{cases}$$

- $next_l(\langle\sigma_{s(n)},\#,\sigma^l_1\rangle) = \#$.

The expectation $next_r(\langle\sigma_{i-1},\sigma_i,\sigma_{i+1}\rangle)$ for the letter σ^r_i, which is the i'th letter in the right successor of the configuration is defined analogously. Consistency with $next_l$ and $next_r$ now gives us a necessary condition for a sequence in Σ^ω to encode a branch in the computation tree of T. In addition, the computation should start with the initial configuration and should eventually reach a final (accepting or rejecting) configuration. So far, we discussed when a sequence in Σ^ω can be a branch in the computation tree of T. In addition, it should be possible to prune the computation tree legally (that is, leave both successors of universal configurations and at least one successor of each existential configuration) and stay only with branches that reach an accepting configuration (or equivalently, branches that do not reach a rejecting configuration).

Recall that the automaton $\mathcal{U}$ accepts an input tree iff one of the following holds.

1. The tree is not the computation tree of T on the empty tape, or

2. The tree is the computation tree of T on the empty tape, but it cannot be pruned legally to a tree in which all branches are accepting.

In oder to check the first condition, $\mathcal{U}$ searches for inconsistency with $next_l$ or $next_r$, inconsistency with the initial configuration, or searches for a branch in which a final configuration is not reached. In order to check inconsistency with $next_l$ or $next_r$, the automaton $\mathcal{U}$ can use its nondeterminism and guesses a place where there is a violation of one of them. Thus, $\mathcal{U}$ guesses a direction, say left, guesses a triple $\langle\sigma_{i-1},\sigma_i,\sigma_{i+1}\rangle \in \Sigma^3$, guesses a node in the tree, checks whether the three letters to be read starting this node (in some direction) are σ_{i-1},σ_i, and σ_{i+1}, and checks whether $next_l(\langle\sigma_{i-1},\sigma_i,\sigma_{i+1}\rangle)$ is not the letter to come $s(n)+1$ letters later, in some left direction (that is by proceeding over virtual branches and one left branch). Once $\mathcal{U}$ sees such a violation, it goes to an accepting sink. In order to check that the first configuration is not the initial configuration, $\mathcal{U}$ simply compares the first $s(n)+1$ letters (in some direction it

[3] We assume that the reading head of T does not "fall" from the right or the left boundaries of the tape. Thus, the case where $(i=1)$ and $(q,\gamma_i) \rightarrow^l (q',\gamma'_i,L)$ and the dual case where $(i=s(n))$ and $(q,\gamma_i) \rightarrow^l (q',\gamma'_i,R)$ are not possible, and analogously for the right successors.

guesses) with $\#(q_0, b)b \ldots b$. Finally, in order to check that a final configuration is not reached, $\mathcal{U}$ starts in an accepting state, guesses a branch and moves to a rejecting sink only when it sees a final configuration.

In order to check the second condition, $\mathcal{U}$ proceeds as follows. A final rejecting configuration (one with $q \in F_{rej}$) is bad. An existential configuration is bad if both its successors are bad. A universal configuration is bad if one of its successors is bad. The computation tree of T cannot be legally pruned to a tree in which all branches are accepting iff the initial configuration is bad. Accordingly, whenever $\mathcal{U}$ reaches real branching (that is, whenever $\mathcal{U}$ reads $\#$), it either guesses a successor to continue with (in case the branching is from a universal configuration), or continues with both successors (in case the branching is from an existential configuration). When $\mathcal{U}$ reaches a final rejecting configuration, it moves to an accepting sink, and when it reaches a final accepting configuration, to moves to an accepting sink.