Relating Word and Tree Automata

Orna Kupferman
Bell Laboratories*

Shmuel Safra
Tel Aviv University†

Moshe Y. Vardi
Rice University‡

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 programing has made it significantly more necessary and difficult. The first step 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

*Address: 600 Mountain Avenue, Murray Hill, NJ 07974, U.S.A. Email: ok@research.att.com

†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 URL: http://www.cs.rice.edu/~vardi

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, BVW94, Kur94]. 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

[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].

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

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 BW [Wol82]. Finally, while the complexity of the nonemptiness problem for BT can be solved in quadratic time [VW86b], it is NP-complete for RT [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 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. Branching temporal logic formulas that describe derived languages constitute a strict fragment of CTL*. In fact, this fragment, called *strongly linear* in [GK94], is a strict fragment of the universal fragment $\forall$CTL* of CTL*. A necessary and sufficient condition for CTL* formulas to be strongly-linear is given in [CD88]: a CTL* formula ψ is strongly linear iff omitting all its path quantifiers results in an LTL formula ξ such that ψ and $A\xi$ are equivalent.

Let us go back to automata. Proving that $DBW < BW$, Landweber showed that the language $L_1 = (0+1)^*1^\omega$ (only finitely many 0's) is in $BW \setminus DBW$. The proof is simple and can be stated in a few lines. Much harder is the proof that $BT < RT$. 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 BT.

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 $L \in BW \setminus DBW$ iff $L\triangle \in RT \setminus BT$. 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 BW is well understood, it also provides a characterization of derived languages that can be described by BT.

The difficult part in the proof is to show that if $L\triangle \in BT$, 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.

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 [2]. 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 Σ_τ 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

[2] We denote strict containment by $\subset$.

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. In the sequel, when we write tree automata, we refer to automata with any acceptance condition, thus, in particular, Büchi 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.,

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

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,

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

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 Büchi tree automaton, then L is recognized by a deterministic Büchi word automaton.*

Proof: Given a 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, $S \setminus F$ 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^k_σ, 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.

□

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 $\sigma \in L$. □

We can now relate the expressiveness gap between RT and BT and the one between BW and DBW.

Theorem 4.3 *For every word language language L,*

$$L \in BW \setminus DBW \Leftrightarrow L\triangle \in RT \setminus BT.$$

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. $L \in BW \Rightarrow L\triangle \in RT$.
2. $L\triangle \in BT \Rightarrow L \in DBW$.
3. $L\triangle \in RT \Rightarrow L \in BW$
4. $L \in DBW \Rightarrow L\triangle \in BT$.

Lemma 4.1 implies Claim 4. Also, as $BW = DRW$, the lemma implies Claim 1 too. Claim 3 follows from Lemma 4.2 and the fact that $BW = RW$. Finally, Claim 2 follows from Theorem 3.1. □

Given a CTL* formula ψ and a Büchi tree automaton $\mathcal{U}_\psi$ associated with ψ, we can use the characterization in [CD88] in order to determine whether ψ is strongly linear [GK94], in which case the language of $\mathcal{U}_\psi$ is derivable. When the language of $\mathcal{U}_\psi$ is derivable, it follows from Theorem 4.3 that the linear requirement that ψ imposes on all computations can be specified by a deterministic Büchi word automaton and that the automaton $\mathcal{U}_\psi$ may be determinized as well.

Our results may also be used to obtain simple proofs for inexpressibility results for temporal logics. It is known, for example, that formulas of CTL can be translated to BT [VW86b]. 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 CTL [EH86] and that the CTL formula $AFAGp$ is not strongly linear [CD88].

Acknowledgment We thank Anca Browne and Mihalis Yannakakis for carefully reading an early draft of this work.

References

[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.

[BVW94] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In D. L. Dill, editor, *Computer Aided Verification, Proc. 6th Int. Conference*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155, Stanford, June 1994. Springer-Verlag, Berlin.

[CD88] E.M. Clarke and I.A. Draghicescu. Expressibility results for linear-time and branching-time logics. In *Proc. Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, pages 428–437. Lecture Notes in Computer Science, 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 *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, White Plains, October 1988.

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

[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. *Handbook of theoretical computer science*, pages 997–1072, 1990.

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

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

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

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

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

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

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

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

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

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

[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 *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*. 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 *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, 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 Symposium on Foundations of Computer Science*, pages 185–194, Tucson, 1983.