Branching Time Temporal Logic and Tree Automata *

Orna Kupferman
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, N.J. 07974, U.S.A.
Email: ok@research.att.com

Orna Grumberg
Department of Computer Science
The Technion
Haifa 32000, Israel
Email: orna@cs.technion.ac.il

December 12, 1995

Abstract

In temporal-logic model checking, we verify the correctness of a program with respect to a desired behavior by checking whether a structure that models the program satisfies a temporal logic formula that specifies this behavior. The close connection between linear-time temporal logics and the theory of automata on infinite words has been an active area of research. In particular, automata-theoretic techniques have proven to be an effective approach to linear-time model checking. On the other hand, for branching-time temporal logics, current automata-theoretic techniques involve an exponential blow up, making them essentially useless for model-checking. In this paper we present an automata-theoretic framework for branching-time model checking. We introduce *simultaneous trees* and associate with every structure a simultaneous tree that enables an automaton to visit different nodes on the same path of the structure simultaneously. With every formula ψ we associate an automaton that accepts exactly these simultaneous trees that originate from structures that satisfy ψ. This enables to use the automaton for model checking which is reduced to the membership problem.

We demonstrate our framework with the branching-time temporal logic CTL and show that it yields a linear automata-based model-checking algorithm, matching the known bound. This is the first time that a model-checking algorithm for a branching-time temporal logic is placed in the automata-theoretic framework.

1 Introduction

Propositional *temporal logics*, which are propositional modal logics that enable description of occurrence of events in time, serve as a classical tool for verifying concurrent programs [Pnu81]. Two possible views regarding the nature of time induce two types of temporal logics. In *linear-time* temporal logics time is treated as if each moment in time has a unique possible future

*A preliminary version appears in [BG93].

while in *branching-time* temporal logics each moment in time may split into various possible futures.

The close connection between the theory of *automata on infinite words* and linear-time temporal logics has been an active area of research and yields benefits for both fields [VW94]. The basic idea is to associate with each linear-time temporal logic formula a finite automaton on infinite words. This enables the reduction of linear-time temporal logic problems (such as satisfiability and model checking) to known automata theory problems (such as nonemptiness and inclusion checking). These reductions are very helpful for implementing temporal-logic based verification methods, and are the key to techniques such as on the fly verification [VW86b, JJ89, CVWY92] that help coping with the "state explosion" problem.

When branching-time temporal logics are considered, automata on words are too weak. Rabin had first defined automata on infinite trees (tree automata, for short) in [Rab69]. Basically, tree automata are similar to automata on words in the sense that a sequence (either a word or a path of a tree) is accepted if and only if the automaton, when running on the sequence, fulfills certain acceptance conditions. But while an automaton on words handles each word independently, handling a path in a tree by a tree automaton depends on the whole tree.

Büchi tree automata, in which the run on each path in the tree is required to visit infinitely often some designated state, are suggested in [VW86a] as a suitable tool for general automata-based techniques for reasoning about branching-time temporal logics. There, Vardi and Wolper associate with each given formula ψ, an automaton of exponential size that accepts some tree if and only if ψ is satisfiable. Moreover, they show that the nonemptiness problem for these automata can be solved in polynomial time. This suggests an exponential automata-based procedure for the satisfiability problem for branching-time temporal logics.

Two problems have impeded the development of automata-based model-checkers for branching-time temporal logics:

1. Tree automata have a fixed branching degree. Yet, for model checking, they should be able to handle trees of various branching degrees.

2. Model checking of the branching-time temporal logic CTL is in deterministic linear time [CES86]. Therefore, an efficient automata-based model checker should consist of linear-size tree automata whose nonemptiness problem is linear. This, however, sounds hopeless: satisfiability of CTL, that has an exponential lower bound, could then be reduced to the nonemptiness problem of such automata. The same phenomenon (of model checking cheaper than satisfiability) appears also in other branching-time temporal logics.

While the first problem is technical and can be easily solved using a more flexible definition of tree automata (we discussed this solution in [BG93]), the second problem is really puzzling; the automata-theoretic approach does not seem to be applicable to branching-time model checking. We this paper we introduce *Simultaneous trees*, and show that they provide a solution for this problem. Note that when linear-time temporal logic is considered, these problems do not exist. There, automata on words handle all models and the lower bounds of satisfiability and model checking coincide. Simultaneous trees (easily constructed from the original structures) are trees in which each sub-tree is duplicated twice as the two leftmost successors of its root. Simultaneous trees enable the automaton to visit different nodes of the same path (in the original structure) simultaneously. As a result, the number of states required for the automaton is smaller.

With every CTL formula ψ and a set $\mathcal{D} \subset \mathbb{N}$ of branching degrees, we associate an automaton $U_{\mathcal{D},\psi}$. When restricted to simultaneous trees, the language of $U_{\mathcal{D},\psi}$ contains exactly these simultaneous trees of branching degrees in $\mathcal{D}$ that originate from structures satisfying ψ. The construction of $U_{\mathcal{D},\psi}$ is based on the *"inheritance approach"* for CTL. Namely, satisfaction of a certain formula ψ, in a state w, depends on the atomic propositions true in w and on satisfaction of related formulas in the successors of w, induced by both ψ and the atomic propositions true in w. We say that these formulas are *inherited* to the successors of w. To employ the inheritance approach, we assume a *positive normal form* for CTL. Consequently, we avoid the need to struggle with formulas of the form $\neg\varphi$, for which an expensive automata complementation is required (the translation to the normal form involves no blow up).

The inheritance approach is incorporated into the tree automata-based framework, by defining the automaton to accept a certain tree iff all the sub-trees rooted at successors of the tree's root are accepted. Each state s_φ in $U_{\mathcal{D},\psi}$ is associated with a *single* sub-formula φ of ψ, such that the automaton with an initial state s_φ accepts a simultaneous tree iff this simultaneous tree originates from a structure satisfying φ. In order to check that a simultaneous tree V originates from structure that satisfies φ, the automaton checks that the sub-trees of V originate from structures satisfying formulas inherited to them, based on φ and on the labeling of the root of V. Since the number of subformulas of a CTL formula ψ is linear in $|\psi|$, the size of $U_{\mathcal{D},\psi}$ is linear in both $|\psi|$ and $|\mathcal{D}|$. Note that in previous methods each state of the automaton is associated with a set of formulas, and the number of states is therefore exponential in the size of the checked formula. Associating with each state a single sub-formula is possible thanks to the usage of simultaneous trees. The latter is possible thanks to the fact that in model-checking, versus satisfiability, the structure is part of the input.

Given a structure K and a branching-time temporal logic formula ψ, we show that the model-checking problem of K with respect to ψ is *linearly* reducible to the membership problem of the simultaneous tree induced by K in the language of $U_{\mathcal{D},\psi}$, where $\mathcal{D}$ is determined by the branching degrees in K. Solving the membership problem involves testing the nonemptiness of the language of a Büchi tree automaton. The latter problem, however, can only be solved in quadratic time [VW86b]. To obtain a linear procedure we show that $U_{\mathcal{D},\psi}$ has a restricted structure. In [MSS86], Muller et al. introduce *weak* alternating automata. We adapt the notion of weakness to Büchi tree automata, show that $U_{\mathcal{D},\psi}$ is weak, and use the fact that the nonemptiness problem for weak Büchi tree automata is solvable in linear time [BVW94].

Consequently, our model-checking procedure is linear in both $|K|$ and $|\psi|$ and matches the complexity of non-automata-based approaches [CES86]. In addition, the automata-based approach broadens the scope of techniques as *on-the-fly* verification [CVWY92] and *partial-order* methods [WG93], which involve automata-theoretic frameworks, from linear-time temporal logics also to branching-time temporal logics.

2 Preliminaries

2.1 Trees and Automata

An *infinite tree* is a set $T \subseteq \mathbb{N}^*$ such that if $x \cdot c \in T$ where $x \in \mathbb{N}^*$ and $c \in \mathbb{N}$, then also $x \in T$, and for all $c' < c$, we have that $x \cdot c' \in T$. The elements of T are called *nodes*, and the empty word ϵ is the *root* of T. For every $x \in T$, the nodes $x \cdot c$ where $c \in \mathbb{N}$ are the *successors* of x, and $d(x)$ denotes the *branching degree* of x (the number of different successors x has). Note that each node of T may have a different branching degree. For a set $\mathcal{D} \subset \mathbb{N}$ of branching degrees and $d \in \mathbb{N}$, let $\mathcal{D} + d$ denote the set $\{x + d : x \in \mathcal{D}\}$. A *path* π of an infinite tree T is a set $\pi \subseteq T$ satisfying:

1. $\epsilon \in \pi$.

2. For all $x \in \pi$, there exists a *unique* $c \in \mathbb{N}$, such that $x \cdot c \in \pi$.

3. If $x \cdot c \in \pi$ then $x \in \pi$.

Note that each path $\pi \subseteq T$ corresponds to a sequence $x_0, x_1, x_2 \ldots$, such that for every $i \geq 0$, x_i is the unique word of length i in π. Given $y \in \pi$, the *suffix* π^y of π, is $x_i, x_{i+1}, x_{i+2} \ldots$, where $y = x_i$.

Given a tree T and $x \in T$, the *sub-tree* T^x of T, rooted at node x, is the infinite tree in which for every $y \in \mathbb{N}^*$, we have that $y \in T^x$ iff $x \cdot y \in T$. Given an alphabet Σ, a Σ*-tree* is a pair $\langle T, V \rangle$ where T is an infinite tree and $V : T \rightarrow \Sigma$ maps each node of T to an element in Σ. Given a Σ-tree $\langle T, V \rangle$ and $x \in T$, the sub-Σ-tree $\langle T^x, V^x \rangle$ is the Σ-tree in which for every $y \in T^x$, we have $V^x(y) = V(x \cdot y)$. If $x \in \mathbb{N}$, we say that $\langle T^x, V^x \rangle$ is an *immediate sub-tree* of $\langle T, V \rangle$. If for each node $x \in T$, we have that $\langle T^x, V^x \rangle = \langle T^{x \cdot 0}, V^{x \cdot 0} \rangle = \langle T^{x \cdot 1}, V^{x \cdot 1} \rangle$, we say that $\langle T, V \rangle$ is a *simultaneous tree*. That is, in a simultaneous tree, sub-trees rooted at the two leftmost successors of each node are duplications of the sub-tree rooted at that node.

Finite automata on infinite trees run on Σ-trees. A Büchi automaton on infinite trees is $U = \langle \Sigma, \mathcal{D}, S, M, s_0, F \rangle$, where Σ is an alphabet, $\mathcal{D} \subset \mathbb{N}$ is a finite set of branching degrees, $M : S \times \Sigma \times \mathcal{D} \rightarrow 2^{S^*}$ is a transition function satisfying $M(s, \sigma, d) \in S^d$, for every $s \in S$, $\sigma \in \Sigma$, and $d \in D$, s_0 is an initial state, and $F \subseteq S$ is an acceptance condition. The size of an automaton U, denoted $|U|$, is defined as $|S| + |M| + |F|$, where $|M|$ is the sum of the lengths of tuples that appear in the transitions in M. Note that U can be encoded in size $O(|U|)$. We use U^s to denote the automaton $\langle \Sigma, \mathcal{D}, S, M, s, F \rangle$, that is, the automaton U with s as the initial state.

A run of U on an input Σ-tree $\langle T, V \rangle$ is an S-tree $\langle T, r \rangle$ such that $r(\epsilon) = s_0$ and for every $x \in T$, we have that $\langle r(x \cdot 0), r(x \cdot 1), \ldots, r(x \cdot d) \rangle \in M(r(x), V(x), d(x))$. If, for instance, $r(0 \cdot 1) = s$, $V(0 \cdot 1) = \sigma$, $d(0 \cdot 1) = 2$, and $M(s, \sigma, 2) = \{\langle s_1, s_2 \rangle, \langle s_4, s_5 \rangle\}$, then either $r(0 \cdot 1 \cdot 0) = s_1$ and $r(0 \cdot 1 \cdot 1) = s_2$, or $r(0 \cdot 1 \cdot 0) = s_4$ and $r(0 \cdot 1 \cdot 1) = s_5$. Given a run $\langle T, r \rangle$ and a path $\pi \subset T$, we define $Inf(r|\pi) = \{s \in S$: for infinitely many $x \in \pi$, we have $r(x) = s\}$. Acceptance of a Σ-tree $\langle T, V \rangle$ by a tree automaton is defined with respect to $Inf(r|\pi)$ of all the paths $\pi \subset T$. Formally, a run $\langle T, r \rangle$ of the automaton U accepts a Σ-tree $\langle T, V \rangle$ iff for all paths $\pi \subset T$, we have $Inf(r|\pi) \cap F \neq \emptyset$. Namely, we require the run to visit states from F infinitely often. An automaton U accepts $\langle T, V \rangle$ iff there exists a run $\langle T, r \rangle$ of U on $\langle T, V \rangle$ such that r accepts $\langle T, V \rangle$. We use $\mathcal{L}(U)$ to denote the language of the automaton U; i.e., the set of all trees accepted by U.

In [MSS86], Muller et al. introduce *weak alternating automata*. The definition applies also to Büchi tree automata. In a weak Büchi automaton there exists a partition of S into disjoint sets, S_i, such that for each set S_i, either $S_i \subseteq F$, in which case S_i is an *accepting set*, or $S_i \cap F = \emptyset$, in which case S_i is a *rejecting set*. In addition, there exists a partial order $\leq$ on the collection of the S_i's such that for every $s \in S_i$ and $s' \in S_j$, if s' occurs in $M(s, \sigma, d)$, for some $\sigma \in \Sigma$ and $d \in \mathcal{D}$, then $S_j \leq S_i$. Thus, transitions from a state in S_i lead to states in either the same S_i or in a lower one. It follows that every path of a run of a weak Büchi automaton, ultimately gets "trapped" within some S_i. The path then satisfies the acceptance condition iff S_i is an accepting set.

In [VW86b], Vardi and Wolper show that the nonemptiness problem for Büchi tree automata is decidable in quadratic running time. Below we prove that if the automata are weak, the problem is decidable in linear running time. A similar proof is presented in [BVW94] for showing that the 1-letter nonemptiness problem for weak alternating automata is solvable in linear time.

Theorem 2.1 *The nonemptiness problem for weak Büchi tree automata is decidable in linear time.*

Proof: We present an algorithm with linear running time for checking the nonemptiness of the language of a weak Büchi automaton $U = \langle \Sigma, \mathcal{D}, S, M, s_0, F \rangle$. We assume, without loss of generality [Rab69], that $\Sigma = \{a\}$ is a singleton alphabet.

The algorithm labels the states of U with either 'T' or 'F'. Typically, states $s \in S$ for which the language of U^s is nonempty are labeled with 'T' and states s for which the language of U^s is empty are labeled with 'F'. The language of A is thus nonempty iff the initial state s_0 is labeled with 'T'. The algorithm works in phases and proceeds up the partial order. Let $S_1 \leq \ldots \leq S_n$

be an extension of the partial order to a total order. In each phase i, the algorithm handles states from the least set S_i which still has not been labeled.

States that belong to a set S_i that is minimal in the partial order, are labeled according to the classification of S_i. Thus, they are labeled with 'T' if S_i is an accepting set, and with 'F' if it is rejecting. For $s \in S$, let $M(s, a)$ denote $\bigcup_{d \in \mathcal{D}} M(s, a, d)$. Once a state $s \in S$ is labeled with 'T' or 'F', transition functions in which s occurs are simplified accordingly as follows. A tuple $\langle s_1, s_2, \ldots, s_d \rangle$ in which one of the s_i's is labeled with 'F' is omitted. If all the tuples in $M(s, a)$ are omitted, s is labeled with 'F'. In a dual way, if $M(s, a)$ has a tuple $\langle s_1, s_2, \ldots, s_d \rangle$ in which all the s_i's are labeled with 'T', then s is labeled with 'T' too. Simplification then proceeds further from s.

Since the algorithm proceeds up the total order, when it reaches a state $s \in S_i$ that is still not labeled, it is guaranteed that all the states in all S_j for which $S_j < S_i$, have already been labeled. Hence, all the states that occur in $M(s, a)$ have the same status as q. That is, they belong to S_i and they are still not labeled. The algorithm then labels s and all the states in $M(s, a)$ according to the classification of S_i. They are labeled 'T' if S_i is accepting and are labeled 'F' otherwise.

Correct operation of the algorithm can be understood as follows. It is guaranteed that once the automaton visits a state that belongs to a minimal set, it visits only states from this set thereafter. Similarly, when the automaton reaches a state q whose labeling can not be decided according to labeling of states in lower sets, this state belongs to a cycle of states of the same status. Moreover, every possible transition of the automaton from s leaves it in such a cycle. This justifies the labeling of the states according to the classification of the set they belong to.

Using an AND/OR graph, the algorithm can be implemented in linear running time. Typically, the graph, induced by the transition relation, keeps the labeling performed during the algorithm execution. Simplification of each transition $M(s, a)$ for all $s \in S$, then costs $O(|M(s, a)|)$. $\square$

2.2 The Temporal Logic CTL

The temporal logic CTL (Computation Tree Logic) provides branching time operators. A linear-time operators, X ("next time") or U ("until"), must be immediately preceded by a path quantifier, either A ("for all paths") or E ("for some path"). A *positive normal form* of CTL is a form of CTL formulas in which negations are applied only to atomic propositions. It can be obtained by pushing negations inward as far as possible using De Morgan's laws and dualities. Translating a formula to a positive normal form at most doubles the length of the formula. We use the temporal operator $\tilde{U}$ as a duality for the U operator, and assume that CTL formulas are given in a positive normal form. Thus, given a set AP of atomic propositions, a CTL formula over AP is one of the following:

- *True*, *False* (represented in the sequel as t and f, respectively), p, or $\neg p$, for all $p \in AP$.
- $\varphi_1 \vee \varphi_2$ or $\varphi_1 \wedge \varphi_2$, where φ_1 and φ_2 are CTL formulas.
- $AX\varphi_1$, $EX\varphi_1$, $A\varphi_1 U\varphi_2$, $E\varphi_1 U\varphi_2$, $A\varphi_1\tilde{U}\varphi_2$, or $E\varphi_1 U\varphi_2$, where φ_1 and φ_2 are CTL formulas.

We use the following abbreviations in writing formulas:

- $F\varphi = tU\varphi$ ("eventually").
- $G\varphi = f\tilde{U}\varphi$ ("always").

For a CTL formula φ, we say that φ is *a proposition*, iff φ is either p or $\neg p$ for some $p \in AP$, and we say that φ is an *U-formula*, if there exist φ_1 and φ_2 such that $\varphi = A\varphi_1 U\varphi_2$ or $\varphi = E\varphi_1 U\varphi_2$. The formula φ_2 is then called the *eventuality* of φ. Similarly, an *$\tilde{U}$-formula* is a formula of the form $A\varphi_1\tilde{U}\varphi_2$ or $E\varphi_1\tilde{U}\varphi_2$.

Below we define the *closure of a formula* for every CTL formula. The closure of φ, $cl(\varphi)$, is the set of CTL formulas inductively defined as follows:

- If φ is a proposition then $cl(\varphi) = \{\varphi, t, f\}$.
- If φ is one of $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, $A\varphi_1 U\varphi_2$, $E\varphi_1 U\varphi_2$, $A\varphi_1\tilde{U}\varphi_2$, or $E\varphi_1\tilde{U}\varphi_2$, then $cl(\varphi) = \{\varphi\} \cup cl(\varphi_1) \cup cl(\varphi_2)$.
- If φ is $AX\varphi_1$ or $EX\varphi_1$, then $cl(\varphi) = \{\varphi\} \cup cl(\varphi_1)$.

It is easy to see that for every φ, we have $|cl(\varphi)| \leq |\varphi| + 2$.

We define the semantics of CTL with respect to a *Kripke structure*, $K = \langle W, R, w^0, L\rangle$, where W is a set of states, $R \subseteq W \times W$ is a transition relation that must be total (i.e., for every $w \in W$ there exists $w' \in W$ such that $\langle w, w'\rangle \in R$), w^0 is an initial state, and $L : W \to 2^{AP}$ maps each state to a set of atomic propositions true in this state. Note that the requirement for a total transition relation reflects the fact that we regard Kripke structures as models of nonterminating programs. The *size* of a Kripke structure K, denoted $|K|$, is defined as $|R| + (|W| \cdot |AP|)$. Note that K can be encoded in space $O(|K|)$. A *path* in K is a sequence of states, $\pi = w_0, w_1, \ldots$ such that for every $i \geq 0$, we have $\langle w_i, w_{i+1}\rangle \in R$. We denote by $bd(w)$ the branching degree of the state w. We use $w \models \varphi$ to indicate that a formula φ holds at a state w (assuming an agreed structure K). For a Kripke structures K, we have that $K \models \varphi$ iff $w^0 \models \varphi$. Given a state w and a CTL formula φ, the relation $\models$ is inductively defined as follows:

- $w \models t$ and $w \not\models f$.
- For an atomic proposition $p \in AP$, $w \models p$ iff $p \in L(w)$ and $w \models \neg p$ iff $p \notin L(w)$.

- $w \models \varphi_1 \vee \varphi_2$ iff $w \models \varphi_1$ or $w \models \varphi_2$.
- $w \models \varphi_1 \wedge \varphi_2$ iff $w \models \varphi_1$ and $w \models \varphi_2$.
- $w \models AX\varphi_1$ iff for all states w' for which $\langle w, w' \rangle \in R$, we have $w' \models \varphi_1$.
- $w \models EX\varphi_1$ iff there exists a state w', such that $\langle w, w' \rangle \in R$ and $w' \models \varphi_1$.
- $w \models A\varphi_1 U \varphi_2$ iff for every path, $w_0, w_1, \ldots$, for which $w_0 = w$, there exists $k_2 \geq 0$, such that $w_{k_2} \models \varphi_2$ and for all $0 \leq k_1 < k_2$, $w_{k_1} \models \varphi_1$.
- $w \models E\varphi_1 U \varphi_2$ iff there exists a path, $w_0, w_1, \ldots$, with $w_0 = w$, and there exists $k_2 \geq 0$, such that $w_{k_2} \models \varphi_2$ and for all $0 \leq k_1 < k_2$, $w_{k_1} \models \varphi_1$.
- $w \models A\varphi_1 \tilde{U} \varphi_2$ iff for every path, $w_0, w_1, \ldots$, for which $w_0 = w$, and for every $k_2 \geq 0$, such that $w_{k_2} \not\models \varphi_2$, there exists $0 \leq k_1 < k_2$, such that $w_{k_1} \models \varphi_1$.
- $w \models E\varphi_1 \tilde{U} \varphi_2$ iff there exists a path, $w_0, w_1, \ldots$, with $w_0 = w$, in which for every $k_2 \geq 0$, such that $w_{k_2} \not\models \varphi_2$, there exists $0 \leq k_1 < k_2$, such that $w_{k_1} \models \varphi_1$.

Note that $w \models A\varphi_1 \tilde{U} \varphi_2$ iff $w \not\models E(\neg\varphi_1)U(\neg\varphi_2)$ and similarly, $w \models E\varphi_1 \tilde{U} \varphi_2$ iff $w \not\models A(\neg\varphi_1)U(\neg\varphi_2)$. That is, a path π satisfies $\varphi_1 \tilde{U} \varphi_2$ if φ_2 holds everywhere along π (thus, the U does not reach its eventuality), or if the first occurrence of $\neg\varphi_2$ is strictly preceded by an occurrence of φ_1 (thus, $\neg\varphi_1$ is falsified before reaching the eventuality). Another way to understand the $\tilde{U}$ operator is to interpret $\varphi_1 \tilde{U} \varphi_2$ by "as long as φ_1 is false, φ_2 must be true".

3 Simultaneous Computation Trees

As previously discussed, in order to reduce their size, the automata we are going to associate with CTL formulas assume simultaneous input trees. In this section we associate with each Kripke structure K, a simultaneous tree $\langle T_K, V_K \rangle$ and an automaton U_K such that the language of U_K is $\{\langle T_K, V_K \rangle\}$.

Recall that simultaneous trees are trees in which each sub-tree is duplicated twice as the two leftmost successors of its root. Typically, given a Kripke structure $K = \langle W, R, w^0, L \rangle$, the simultaneous tree $\langle T_K, V_K \rangle$ is obtained by unwinding K into a tree in which each node points, in addition to the simultaneous trees of its successors in K, also to two new copies of itself. Formally, $\langle T_K, V_K \rangle$ is the W-tree defined as follows: let $succ_R(w) = \langle w_0, \ldots, w_{bd(w)-1} \rangle$ be an ordered list of w's R-successors (we assume that the nodes of W are ordered). Then, T_K and V_K are the smallest sets that satisfy the following:

1. $\epsilon \in T_K$ and $V_K(\epsilon) = w^0$.

2. For $y \in T_K$ with $succ_R(V_K(y)) = \langle w_0, \ldots, w_m \rangle$ and for all $0 \leq i \leq m+2$, we have $y \cdot i \in T_K$, $V_K(y \cdot 0) = V_K(y \cdot 1) = V_K(y)$, and for all $i \geq 2$, we have that $V_K(y \cdot i) = w_{i-2}$.

We sometimes refer to $\langle T_K, V_K \rangle$ as a simultaneous computation tree over 2^{AP}, taking instead $V_K(x)$, the label $L(V_K(x))$. In Figure 1, we present a Kripke structure and its induced simultaneous computation tree.

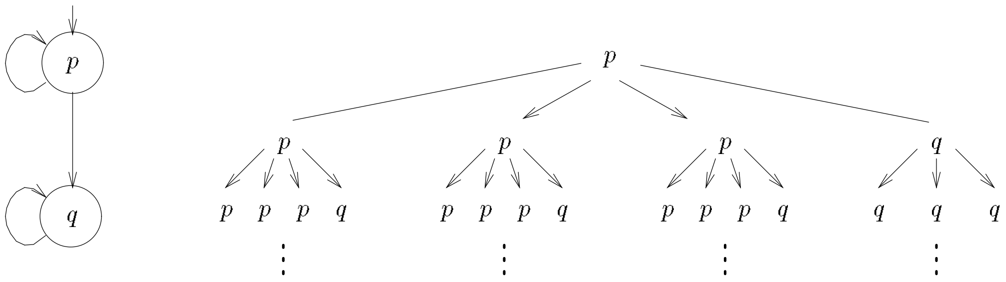

Figure 1: A Kripke structure and its induced simultaneous computation tree.

Given a Kripke structure K, we construct an automaton U_K such that $\mathcal{L}(U_K) = \{\langle T_K, V_K \rangle\}$. The automaton $U_K = \langle W, \mathcal{D}, W \cup \{r\}, M, w^0, W \rangle$, where $\mathcal{D}$ and M are defined as follows:

- $\mathcal{D} = 2 + \bigcup_{w \in W} \{bd(w)\}$.
- $M : (W \cup \{r\}) \times W \times \mathcal{D} \rightarrow 2^{(W \cup \{r\})^*}$ is defined as follows. Let $succ_R(w) = \langle w_0, \ldots, w_{bd(w)-1} \rangle$.

$$M(s, \sigma, d) = \begin{cases} \langle w, w, w_0, \ldots, w_{bd(w)-1} \rangle & \text{If } s = w \text{ and } d = bd(w) + 2. \\ \langle r, r, \ldots, r \rangle & \text{Otherwise.} \end{cases}$$

It is easy to see that the language of U_K contains indeed a single tree, $\langle T_K, V_K \rangle$, and that the size of U_K is linear in the size of K.

4 Translating CTL Formulas to Automata

In this section, we present a translation from CTL formulas to automata. Given a CTL formula ψ and a set $\mathcal{D}$ of branching degrees, we construct an automaton $U_{\mathcal{D},\psi}$, such that $U_{\mathcal{D},\psi}$ accepts a simultaneous computation tree $\langle T_K, V_K \rangle$ with branching degrees in $\mathcal{D}$ iff K satisfies ψ. Thus, with respect to simultaneous computation trees, the language of the automaton $U_{\mathcal{D},\psi}$ is exactly the set of trees with branching degrees in $\mathcal{D}$ that originate from Kripke structures that satisfy ψ.

The key to our construction is the inheritance approach for CTL. This approach is based on the idea that satisfaction of a certain CTL formula in a state w, depends on satisfaction of related and simpler CTL formulas in w and in the successor states of w. Each state in $U_{\mathcal{D},\psi}$ is associated with a *single* formula $\varphi \in cl(\psi)$. Associating a state s_φ with a formula φ, means that the automaton, with s_φ as the initial state, accepts exactly all simultaneous computation trees that originate from structures satisfying φ. In order to determine whether a given tree $\langle T, V\rangle$ originates from a structure satisfying φ, we check whether the d immediate sub-trees of $\langle T, V\rangle$ fulfill one of the "inheritance packages" induced by $M(\varphi, \sigma, d)$. Precisely, the transition function $M(\varphi, \sigma, d)$ associates with each immediate sub-tree of $\langle T, V\rangle$, a formula that should hold on the structure that this sub-tree originates from. Since $\langle T, V\rangle$ is a simultaneous tree, $\langle T^0, V^0\rangle$ and $\langle T^1, V^1\rangle$ are copies of $\langle T, V\rangle$. Therefore, a requirement inherited to them is actually a requirement on $\langle T, V\rangle$ itself. For example, $M(\varphi_1 \wedge \varphi_2, \sigma, d)$ inherits φ_1 to $\langle T^0, V^0\rangle$, inherits φ_2 to $\langle T^1, V^1\rangle$, and inherits t (i.e. no requirement) to all other immediate sub-trees.

Given a CTL formula ψ and a set $\mathcal{D}$ of branching degrees, the automaton associated with ψ and $\mathcal{D}$ is $U_{\mathcal{D},\psi} = \langle 2^{AP}, \mathcal{D}, cl(\psi), M, \psi, F\rangle$ where,

- $F = \{t\} \cup \{\varphi : \varphi \in cl(\psi)$ and φ is an $\tilde{U}$-formula$\}$.
- $M : cl(\psi) \times 2^{AP} \times \mathcal{D} \to 2^{cl(\psi)^*}$ is defined according to the structure of φ as follows.
  - If $\varphi = t$, $\varphi = p$ with $p \in \sigma$, or $\varphi = \neg p$ with $p \notin \sigma$, then $M(\varphi, \sigma, d) = \{\langle \overbrace{t, t, \ldots, t}^{d}\rangle\}$.
  - If $\varphi = f$, $\varphi = p$ with $p \notin \sigma$, or $\varphi = \neg p$ with $p \in \sigma$, then $M(\varphi, \sigma, d) = \{\langle f, f, \ldots, f\rangle\}$.
  - $M(\varphi_1 \wedge \varphi_2, \sigma, d) = \{\langle \varphi_1, \varphi_2, t, t, \ldots, t\rangle\}$.
  - $M(\varphi_1 \vee \varphi_2, \sigma, d) = \{\langle \varphi_1, t, t, \ldots, t\rangle, \langle \varphi_2, t, t, \ldots, t\rangle\}$.
  - $M(AX\varphi_1, \sigma, d) = \{\langle t, t, \varphi_1, \varphi_1, \ldots, \varphi_1\rangle\}$.
  - $M(EX\varphi_1, \sigma, d) = \{\langle t, t, \varphi_1, t, t, \ldots, t\rangle, \langle t, t, t, \varphi_1, t, \ldots, t\rangle, \ldots, \langle t, t, \ldots, t, \varphi_1\rangle\}$.
  - $M(A\varphi_1 U\varphi_2, \sigma, d) = \{\langle \varphi_2, t, t, \ldots, t\rangle, \langle \varphi_1, t, A\varphi_1 U\varphi_2, A\varphi_1 U\varphi_2, \ldots, A\varphi_1 U\varphi_2\rangle\}$.
  - $M(E\varphi_1 U\varphi_2, \sigma, d) = \{\langle \varphi_2, t, t, \ldots, t\rangle,$ $\langle \varphi_1, t, E\varphi_1 U\varphi_2, t, t, \ldots, t\rangle, \langle \varphi_1, t, t, E\varphi_1 U\varphi_2, t, \ldots, t\rangle, \ldots, \langle \varphi_1, t, t, \ldots, t, E\varphi_1 U\varphi_2\rangle\}$.
  - $M(A\varphi_1 \tilde{U}\varphi_2, \sigma, d) = \{\langle \varphi_2, \varphi_1, t, t, \ldots, t\rangle, \langle \varphi_2, t, A\varphi_1 \tilde{U}\varphi_2, A\varphi_1 \tilde{U}\varphi_2, \ldots, A\varphi_1 \tilde{U}\varphi_2\rangle\}$.
  - $M(E\varphi_1 \tilde{U}\varphi_2, \sigma, d) = \{\langle \varphi_2, \varphi_1, t, t, \ldots, t\rangle,$ $\langle \varphi_2, t, E\varphi_1 \tilde{U}\varphi_2, t, t, \ldots, t\rangle, \langle \varphi_2, t, t, E\varphi_1 \tilde{U}\varphi_2, t, \ldots, t\rangle, \ldots, \langle \varphi_2, t, t, \ldots, t, E\varphi_1 \tilde{U}\varphi_2\rangle\}$.

  Note that when φ is not a proposition, $M(\varphi, \sigma, d)$ is independent of σ.

Keeping in mind that the two leftmost elements in each tuple induce requirements on the current state, the idea behind the automata is straightforward. A state satisfies $E\varphi_1 \tilde{U}\varphi_2$ for instance, if it either satisfies both φ_1 and φ_2, in which case no requirements are imposed on its successors (which corresponds to the transition $\langle \varphi_2, \varphi_1, t, t, \ldots, t\rangle$), or it satisfies φ_2, in which case there exists one successor that satisfies $E\varphi_1 \tilde{U}\varphi_2$ (which corresponds to one of the other transitions). In the second case, since $E\varphi_1\tilde{U}\varphi_2$ is in F, inheriting $E\varphi_1\tilde{U}\varphi_2$ forever through states that are all satisfying φ_2 is legitimate.

To show that $U_{\mathcal{D},\psi}$ is a weak Büchi tree automaton, we define a partition of its state set into disjoint sets and a partial order over these sets. Each formula $\varphi \in cl(\psi)$ constitutes a (singleton) set $\{\varphi\}$ in the partition. The partial order is then defined by $\{\varphi_1\} \leq \{\varphi_2\}$ iff $\varphi_1 \in cl(\varphi_2)$. Since each transition of the automaton from a state φ leads to states associated with formulas in $cl(\varphi)$, the weakness conditions hold. In particular, each set is either contained in F or disjoint from F.

In examples 4.1 and 4.2 below, we describe the automata of two CTL formulas.

Example 4.1 *Consider the formula $\psi' = A((\neg AXp)Uq)$. Its positive normal form is $\psi = A((EX\neg p)Uq)$. For every set $\mathcal{D} \subset I\!N$, the automaton associated with $\mathcal{D}$ and ψ is $U_{\mathcal{D},\psi} = \langle 2^{\{p,q\}}, \mathcal{D}, \{\psi, EX\neg p, \neg p, q, t, f\}, M, \psi, \{t\}\rangle$ where M is described in the following table.*

S	$M(s, \sigma, d)$ (for all $\sigma \subseteq \{p, q\}$ and $d \in \mathcal{D}$)
ψ	$\{\langle q, t, t, \ldots, t\rangle, \langle EX\neg p, t, \psi, \psi, \ldots, \psi\rangle\}$
$EX\neg p$	$\{\langle t, t, \neg p, t, t, \ldots, t\rangle, \langle t, t, t, \neg p, t, \ldots, t\rangle, \ldots, \langle t, t, \ldots, t, \neg p\rangle\}$
$\neg p$	$\{\langle t, t, \ldots, t\rangle\}$ if $p \notin \sigma$
	$\{\langle f, f, \ldots, f\rangle\}$ if $p \in \sigma$
q	$\{\langle t, t, \ldots, t\rangle\}$ if $q \in \sigma$
	$\{\langle f, f, \ldots, f\rangle\}$ if $q \notin \sigma$
t	$\{\langle t, t, \ldots, t\rangle\}$
f	$\{\langle f, f, \ldots, f\rangle\}$

In the state ψ, choosing $\langle q, t, t, \ldots, t\rangle$ means that the automaton guesses that q, the eventuality of ψ is satisfied in the present. On the other hand, choosing $\langle EX\neg p, t, \psi, \psi, \ldots, \psi\rangle$ means that the automaton postpones the requirement of fulfilling the eventuality to the future, and therefore requires $EX\neg p$ to be satisfied in the present and ψ to be satisfied by all the successors. Infinite postponing, however, is impossible since $\psi \notin F$. In the state $EX\neg p$, the automaton imposes no requirements on the present (the two leftmost elements are t), but it nondeterministically chooses a successor to which a requirement for satisfying $\neg p$ is inherited.

Example 4.2 *Consider the formula $\psi' = AFAGp$. Its positive normal form is $\psi = A(tUAf\tilde{U}p)$. For every set $\mathcal{D}$, the automaton associated with $\mathcal{D}$ and ψ is $U_{\mathcal{D},\psi} = \langle 2^{\{p,q\}}, \{\psi, \mathcal{D}, Af\tilde{U}p, p, t, f\}, M, \psi, \{t, Af\tilde{U}p\}\rangle$ where M is described in the following table.*

S	$M(s, \sigma, d)$ *(for all* $\sigma \subseteq \{p, q\}$ *and* $d \in \mathcal{D}$*)*
ψ	$\{\langle Af\tilde{U}p, t, t, \ldots, t\rangle, \langle t, t, \psi, \psi, \ldots, \psi\rangle\}$
$Af\tilde{U}p$	$\{\langle p, f, t, t, \ldots, t\rangle, \langle p, t, Af\tilde{U}p, Af\tilde{U}p, \ldots, Af\tilde{U}p\rangle\}$
p	$\{\langle t, t, \ldots, t\rangle\}$ *if* $p \in \sigma$
	$\{\langle f, f, \ldots, f\rangle\}$ *if* $p \notin \sigma$
t	$\{\langle t, t, \ldots, t\rangle\}$
f	$\{\langle f, f, \ldots, f\rangle\}$

As in the previous example, in the state ψ*, the automaton may either guess that* $Af\tilde{U}p$*, the eventuality of* ψ*, is satisfied in the present, or choose* $\langle t, t, \psi, \psi, \ldots, \psi\rangle$ *which means the requirement for fulfilling the eventuality of* ψ *is postponed to the future. Again, since* $\psi \notin F$*, infinite postponing is impossible. In the state* $Af\tilde{U}p$*, the automaton should accept only trees that originate from structures in which* p *is always true in all paths. Since once reaching the state* f*, the automaton stays there forever, choosing* $\langle p, f, t, t, \ldots, t\rangle$ *never leads to an accepting run. Therefore, in an accepting run, the automaton must choose the second option,* $\langle p, t, Af\tilde{U}p, \ldots, Af\tilde{U}p\rangle$*, guaranteeing for satisfaction of* p *in the present and satisfaction of* $Af\tilde{U}p$ *in all successors. Here, the automaton has no choice but to keep visiting* $Af\tilde{U}p$ *forever; yet, since* $Af\tilde{U}p \in F$*, this is permitted.*

We now prove the correctness of our construction. That is, we prove that $U_{\mathcal{D},\psi}$ accepts a simultaneous computation tree iff the tree originates from a structure satisfying ψ, with the suitable branching degrees.

Theorem 4.3 *For every CTL formula* ψ *and for every Kripke structure* K *with branching degrees in* $\mathcal{D} - 2$*, we have that* $\langle T_K, V_K\rangle \in \mathcal{L}(U_{\mathcal{D},\psi})$ *iff* K *satisfies* ψ*.*

Proof: We first prove that $U_{\mathcal{D},\psi}$ is sound. I.e., that if $\langle T_K, V_K\rangle \in \mathcal{L}(U_{\mathcal{D},\psi})$ then K satisfies ψ. Given an accepting run $\langle T_K, r\rangle$ of $U_{\mathcal{D},\psi}$ on a simultaneous computation tree $\langle T_K, V_K\rangle$, we prove that for every $x \in T_r$ such that $r(x) = \varphi$, we have $V_K(x) \models \varphi$. Thus, in particular, $V_K(\epsilon) \models \psi$. The proof proceeds by induction on the structure of φ. The case where φ is an atomic proposition is immediate. The cases where φ is $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $AX\varphi_1$, or $AX\varphi_1$ follow easily, by the induction hypothesis, from the definition of M. Less immediate are the cases where φ is an U-formula or a $\tilde{U}$-formula. Consider first the case where φ is of the form $A\varphi_1 U\varphi_2$ or $E\varphi_1 U\varphi_2$. As $\langle T_K, r\rangle$ is an accepting run, it visits the state φ only finitely often. Since $U_{\mathcal{D},\psi}$ keeps inheriting φ as long as φ_2 is not satisfied, then it is guaranteed, by the definition of M and the induction hypothesis, that along all paths or some path, as required in φ, indeed φ_2 eventually holds and φ_1 holds everywhere until then. Consider now the case where φ is of the form $A\varphi_1\tilde{U}\varphi_2$ or $E\varphi_1\tilde{U}\varphi_2$. Here, it is guaranteed, by the definition of M and the induction hypothesis, that φ_2 holds either always or until both φ_2 and φ_1 hold.

We now prove that $U_{\mathcal{D},\psi}$ is complete. I.e., that if $K \models \psi$ and the branching degrees of K are in $\mathcal{D} - 2$, then $\langle T_K, V_K \rangle \in \mathcal{L}(U_{\mathcal{D},\psi})$. Given a simultaneous computation tree $\langle T_K, V_K \rangle$ such that all the branching degrees of T_K are in $\mathcal{D}$ and $K \models \psi$, we show that there exists an accepting run of $U_{\mathcal{D},\psi}$ over $\langle T_K, V_K \rangle$. Consider a run $\langle T_K, r \rangle$ and node $x \in T_K$ for which $r(x) = \varphi$ and $V_K(x) \models \varphi$ (note that for every run, ϵ is such a node). By the definition of M, r can proceed such that all the successors $x \cdot c$ of x in T_K have $r(x \cdot c) = \varphi_c$ with $V_K(x \cdot c) \models \varphi_c$. Let r be a run that proceeds as above and, in addition, whenever φ is of the form $A\varphi_1 U \varphi_2$ or $E\varphi_1 U \varphi_2$ and $V_K(x) \models \varphi_2$, proceeds according to $M(\varphi_2, V_K(x), d(x))$. It is easy to see that all the paths in such r either reach the state t or reach a state associated with a $\tilde{U}$-formula and stay there thereafter. Thus, r is accepting. $\square$

5 CTL Model Checking Using Automata

In this section we introduce a linear-time automata-based algorithm for the model checking problem for CTL. The model-checking problem is: given a Kripke structure K and a temporal logic formula ψ, determine whether $K \models \psi$. When automata on words are used to describe linear-time temporal logic formulas [VW86b], each Kripke structure may correspond to infinitely many computations. Accordingly, model checking is reduced to an inclusion check between the computations allowed by the Kripke structure and the language of the automaton describing the formula. Using tree automata to describe branching-time temporal logic formulas, each Kripke structure corresponds to a single non-deterministic computation. On that account, model checking is reduced to checking the membership of this computation in the language of the automaton describing the formula.

Given a CTL formula ψ, the language of $U_{\mathcal{D},\psi}$, when restricted to simultaneous computation trees, contains exactly all these trees which originate from Kripke structures with branching degrees in $\mathcal{D} - 2$ that satisfy ψ. Given a Kripke structure K, the language of the automaton U_K contains exactly one tree, $\langle T_K, V_K \rangle$, the simultaneous tree induced by K. The language of $U_{\mathcal{D},\psi}$, intersected with that of U_K (regarding it as a simultaneous computation tree), may either contain a single tree, V_K, in which case $K \models \psi$, or be empty, in which case $K \not\models \psi$.

Given the Büchi tree automaton $U_K = \langle W, \mathcal{D}, W \cup \{r\}, M_K, w^0, W \rangle$ and a weak büchi tree automaton $U_{\mathcal{D},\psi} = \langle 2^{AP}, \mathcal{D}, cl(\psi), M_\psi, \psi, F_\psi \rangle$, we define their product as a weak Büchi tree automaton $U_{K,\psi} = U_K \times U_{\mathcal{D},\psi}$ on computation trees. Since once visiting a rejecting state, U_K keeps visiting only rejecting states thereafter, the construction is simpler than the known product of Büchi tree automata [VW86a]. Formally, $U_{K,\psi} = \langle 2^{AP}, \mathcal{D}, S, M, s_0, F \rangle$ is defined as follows:

- $S = (W \cup \{r\}) \times cl(\psi)$.
- $s_0 = \langle w^0, \psi \rangle$.

- $F = W \times F_\psi$.
- $M : S \times 2^{AP} \times \mathcal{D} \to 2^{S^*}$, where for every $\langle s_K, s_\psi \rangle \in S$, $\sigma \in \Sigma$, and $d \in \mathcal{D}$, we have

$$\langle \langle s_K^1, s_\psi^1 \rangle, \langle s_K^2, s_\psi^2 \rangle, \ldots, \langle s_K^d, s_\psi^d \rangle \rangle \in M(\langle s_K, s_\psi \rangle, \sigma, d)$$

iff $\langle s_K^1, s_K^2, \ldots, s_K^d \rangle \in M_K(s_K, \sigma, d)$ and $\langle s_\psi^1, s_\psi^2, \ldots, s_\psi^d \rangle \in M_\psi(s_\psi, \sigma, d)$.

Note that a precondition for doing the product is that the sets of branching degrees of $U_{\mathcal{D},\psi}$ and U_K coincide. This, however, is not a problem, as we can always determine $\mathcal{D}$ by K and construct the suitable $U_{\mathcal{D},\psi}$.

Note also that $U_{K,\psi}$ is a weak Büchi tree automaton. Here, every formula $\varphi \in cl(\psi)$, that constitutes a singleton set in $U_{\mathcal{D},\psi}$, constitutes the set $W \times \{\varphi\}$. In addition, we have the set $\{r\} \times cl(\psi)$. The partial order is then defined by $W \times \{\varphi_1\} \leq W \times \{\varphi_2\}$ iff $\varphi_1 \in cl(\varphi_2)$ with the set $\{r\} \times cl(\psi)$ minimal in the partial order.

Theorem 5.1 *Given a CTL formula ψ and a Kripke structure K with branching degrees in $\mathcal{D} - 2$, the automaton $U_{K,\psi} = U_K \times U_{\mathcal{D},\psi}$ satisfies:*

1. *$\mathcal{L}(U_{K,\psi}) \neq \emptyset$ iff $K \models \psi$.*
2. *The nonemptiness of $\mathcal{L}(U_{K,\psi})$ can be tested in time linear in $|K| \cdot |\psi|$.*

Proof: $\mathcal{L}(U_{K,\psi}) \neq \emptyset$ iff there exists $\langle T, V \rangle \in \mathcal{L}(U_{K,\psi})$. By the definition of $U_{K,\psi}$ as an intersection automaton, this holds iff $\langle T, V \rangle \in \mathcal{L}(U_K)$ and $\langle T, V \rangle \in \mathcal{L}(U_{\mathcal{D},\psi})$. This, by Theorem 4.3, holds iff K satisfies ψ.

We now refer to the size of $U_{K,\psi}$. The state space of $U_{K,\psi}$ is clearly linear in $|K| \cdot |\psi|$. For each state $\langle w, \varphi \rangle$ of $U_{K,\psi}$, there are single $\sigma \in 2^{AP}$ and $d \in \mathcal{D}$ such that $M(\langle w, \varphi \rangle, \sigma, d)$ does not contain states from $\{r\} \times cl(\psi)$. Indeed, only $\sigma = L(w)$ and $d = bd(w) + 2$ do not bring the run to a rejecting state. Therefore, the size of M is also linear in $|K| \cdot |\psi|$ and so is the size of $U_{K,\psi}$.

Since $U_{K,\psi}$ is a weak Büchi automaton, its nonemptiness can be tested, by Theorem 2.1, in time linear in its size. Since the size of $U_{K,\psi}$ is linear in $|K| \cdot |\psi|$, its nonemptiness can be tested in time linear in $|K| \cdot |\psi|$. $\square$

In [EJS93], a product of a μ-calculus formula and a Kripke structure is constructed as a base for a model-checking procedure. Theorem 5.1 can be viewed as an automata-theoretic generalization of their method.

We conclude with the automata-based model-checking algorithm. Given a CTL formula ψ and a Kripke structure K with branching degrees in $\mathcal{D}$, proceed as follows:

(1) Construct the weak Büchi tree automaton $U_{\mathcal{D}+2,\psi}$.

(2) Construct the Büchi tree automaton U_K.

(3) Construct $U_{K,\psi} = U_K \times U_{\mathcal{D},\psi}$.

(4) Output "Yes" if $\mathcal{L}(U_{K,\psi}) \neq \emptyset$, and "No", otherwise.

6 Conclusions

In this paper we showed that tree automata, which have been considered unsuitable for branching-time model checking, provide a neat and optimal framework for branching-time temporal logics and can handle, efficiently, model checking.

The crucial point in our method is that when branching-time temporal logics are considered, a wise presentation of the model can save the exponential blow up which is involved in conventional translations of formulas to automata. Moreover, when model checking is considered, and the model is part of the problem input, a wise presentation involves no complexity penalty and suits properly the automata-theoretic framework.

In conclusion, introducing an efficient automata-theoretic framework for branching time model checking, the paper both *closes* the theoretical long-standing gap between linear-time and branching-time temporal logics, and *opens* the way to practical automata-based model-checking algorithms that have already proved themselves in the context of linear time.

Acknowledgment

We thank Moshe Vardi for many helpful suggestions and discussions.

References

[BG93] O. Bernholtz and O. Grumberg. Branching time temporal logic and AmorPHOus tree automata. In *Proc. 4th Conferance on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 262–277, Hildesheim, August 1993. Springer-Verlag.

[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, California, June 1994. Springer-Verlag, Berlin.

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

[CVWY92] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.

[EJS93] E.A. Emerson, C. Jutla, and A.P. Sistla. On model-checking for fragments of μ-calculus. In *Computer Aided Verification, Proc. 5th Int. Workshop*, volume 697, pages 385–396, Elounda, Crete, June 1993. Lecture Notes in Computer Science, Springer-Verlag.

[JJ89] C. Jard and T. Jeron. On-line model-checking for finite linear temporal logic specifications. In *Automatic Verification Methods for Finite State Systems, Proc. Int. Workshop, Grenoble*, volume 407, pages 189–196, Grenoble, June 1989. Lecture Notes in Computer Science, Springer-Verlag.

[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 Int. Colloquium on Automata, Languages and Programming*. Springer-Verlag, 1986.

[Pnu81] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

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

[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):182–21, April 1986.

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

[WG93] P. Wolper and P. Godefroid. Partial-order methods for temporal verification. In *Proc. 4th Conferance on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 233–246, Hildesheim, August 1993. Springer-Verlag.