Branching Time Temporal Logic and *AmorPHOus* Tree Automata

Orna Bernholtz
Department of Computer Science
The Technion
Haifa 32000, Israel
email: ornab@cs.technion.ac.il

Orna Grumberg
AT&T Bell Laboratories
Murray Hill
NJ 07974, USA
email: orna@research.att.com

Abstract

An automata-theoretic framework for branching-time temporal logics is presented. We introduce a new type of finite automata on infinite trees, the *amorphous automata*, and use them as a formalism to represent efficiently CTL formulas. In addition, we introduce *simultaneous trees*, and associate with every model for CTL, a simultaneous tree that enables a tree automaton to visit different nodes on the same path of the tree simultaneously. With every formula ψ, we associate an amorphous automaton U_ψ, that accepts *exactly* those simultaneous trees (of any branching degree) that originate from models that satisfy ψ. This enables to use the automaton both for model checking which is reduced to the membership problem, and for satisfiability decision, which is reduced to testing the nonemptiness of an extension of U_ψ that does not assume simultaneous input trees.

The amorphous automata for CTL use the Büchi acceptance condition. The size of U_ψ is linear in $|\psi|$ and the extension required for satisfiability is exponential. Based on that, we get a polynomial model checking procedure and an exponential decision procedure for CTL, both match the known lower bounds. 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 ([Pn77]). 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 while in branching-time temporal logics each moment in time may split into various possible futures.

The close connection between sequential automata theory and linear-time temporal logics has been an active area of research and yields benefits for both fields ([Va91]). 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).

When branching-time temporal logics are considered, sequential automata are too weak. Rabin had first defined finite automata on infinite trees in [Ra69]. Basically, tree automata are similar to sequential automata 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 a sequential automaton 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 over 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 logics. There, Vardi and Wolper associate with each given formula ψ, an exponential size automaton 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.

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

1. Tree automata have fixed branching degrees. Yet, for model checking, they should be able to handle trees of various, not previously specified, and unbounded branching degrees.
2. Model checking of the branching-time temporal logic CTL is in deterministic polynomial time. Therefore, an efficient automata-based model checker should consist of polynomial size tree automata whose nonemptiness problem is polynomial. 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.

Note that when linear-time temporal logic is considered, those problems do not exist. There, sequential automata handle all models and the lower bounds of satisfiability and model checking coincide. Amorphous automata and simultaneous trees, introduced in this work, suggest a solution for both problems.

Amorphous automata have a flexible transition relation and can be adjusted during their run to handle any input tree. Therefore, they are much stronger than known precedents ([Va89]), and can check for acceptance trees of any, and not necessarily fixed, branching degree, including infinite. Moreover, they accept exactly *all* the models of a formula. The latter is a strong property which is not required for satisfiability, where nonemptiness of the automaton's language is checked.

In [Ra69], Rabin uses the simultaneous run method, in which several copies of the automaton are simultaneously activated on a tree. We introduce a new technique in which simultaneity is handled via the model. Simultaneous trees (easily constructed from the original models) 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 simultaneously. As a result, the number of states required for the automaton is smaller. Satisfiability, however, cannot be reduced to the nonemptiness of the automaton language, and involves an exponential extension of the automaton. This solves the second problem.

With every CTL formula ψ, we associate an amorphous automaton U_ψ, that accepts *exactly* those simultaneous trees (of any branching degree) that originate from models that satisfy ψ. The construction of U_ψ 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 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. Thus, we avoid the

need to struggle with formulas of the form $\neg\varphi$, for which an expensive automata complementation is required.

The inheritance approach is incorporated into the tree automata-based framework, by defining the amorphous automaton to accept a certain tree, iff all the sub-trees rooted at successors of its root are accepted. Each state s_φ in U_ψ is associated with a *single sub-formula* φ of ψ, such that the automaton with an initial state s_φ accepts all trees satisfying φ. In order to check that a tree V satisfies φ, the automaton checks that the sub-trees of V satisfy formulas inherited to them, based on φ and on the labeling of the root of V. Note that in previous methods each state 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 enabled by the use of the simultaneous trees method.

Since U_ψ accepts exactly all the simultaneous computation trees that originate from models that satisfy ψ, deciding the satisfiability of ψ can be reduced to the nonemptiness problem of the language of U_ψ, when restricted only to simultaneous computation trees. This, however, can not be done directly within the lower bound complexity for satisfiability. Instead, we suggest an extension of U_ψ which does not assume simultaneous input trees. Using the extended automaton, satisfiability is reduced to the usual nonemptiness test. Each state in the new automaton is associated with a set of formulas. Each such set contains formulas that U_ψ, during its run on a simultaneous tree, inherits to different copies of a certain sub-tree. Conceptually, the resulted automaton is very similar to U_ψ and simplifies known automata for handling satisfiability.

Given a model K, we implement model checking via membership checking by a construction of a Büchi tree automaton U_K that serves as the model automaton. The language of U_K contains exactly one tree, the simultaneous computation tree induced by K. Its intersection with the language of U_ψ is proved to be nonempty if and only if K satisfies ψ.

The rest of this paper is organized as follows: In section 2 we present basic definitions concerning trees, amorphous automata, and temporal logics. In section 3 we show how to associate with each Kripke structure a simultaneous tree. In section 4 we introduce the amorphous automata for CTL. Sections 5 and 6 consider model checking and satisfiability using amorphous automata, and section 7 suggests directions for future research.

2 Basic Definitions

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 $c' < c$, $x \cdot c' \in T$. 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 number of different successors x has. A *path* π of an infinite tree T is a prefix-closed set $\pi \subseteq T$ such that for every $i \geq 0$, there exists a unique $x \in \pi$ with $|x| = i$. Thus, each path $\pi \subseteq T$ corresponds to a unique sequence $c_1, c_2, \ldots$ in $\mathbb{N}^\omega$, such that $c_1 \cdot c_2 \cdots c_i$ is the word of length i in π.

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 $y \in \mathbb{N}^*$, $y \in T^x \Leftrightarrow x \cdot y \in T$. Given an alphabet Σ and an infinite tree T, a Σ-*tree* is a mapping $V : T \to \Sigma$. If $\Sigma = 2^{AP}$ where AP is a set of atomic propositions, we say that V is a *computation tree*. Typically, a computation

tree maps each node of T, corresponding to a state in a computation, to the set of atomic propositions which are true in that state.

Given a Σ-tree $V : T \to \Sigma$, and $x \in T$, the sub-Σ-tree $V^x : T^x \to \Sigma$, is the Σ-tree in which for every $y \in T^x$, $V^x(y) = V(x \cdot y)$. If $x \in \mathbb{N}$, we say that V^x is an *immediate sub-tree* of V. If for each node $x \in T$, $V^x = V^{x \cdot 0} = V^{x \cdot 1}$, we say that V 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.

We introduce a new kind of tree automata, the *amorphous automata*. Amorphous automata extend conventional tree automata in that they can handle trees with various and not previously specified branching degrees. Their amorphous nature enables them to be adjusted during their run to any branching degree. In order to achieve this, each amorphous automaton has, in addition to the standard components of a tree automaton, also a *set of tuple representations* $\mathcal{R}$, and a function *strc* (stretch). Each $rep \in \mathcal{R}$ represents a set of d-tuples of automaton states, for an unspecified parameter d. Given $rep \in \mathcal{R}$ and a *stretching factor* d, $strc(d, rep)$ provides a set of d-tuples of states. For a set $R \subseteq \mathcal{R}$ of tuple representations, $strc[d, R] = \bigcup_{rep \in R} strc(d, rep)$.

An amorphous Büchi automaton (ABA) on Σ-trees is $U = \langle \mathcal{R}, strc, S, M, s_0, F \rangle$, where $\mathcal{R}$ and *strc* are as above, S is a finite set of states, $M : S \times \Sigma \to 2^{\mathcal{R}}$ is a nondeterministic table of moves, $s_0 \in S$ is an initial state, and $F \subseteq S$ is a set of designated states.

A run of an ABA U on a Σ-tree V, is a mapping $r : T \to S$ such that $r(\epsilon) = s_0$ and for every $x \in T$, $\langle r(x \cdot 0), r(x \cdot 1), \ldots, r(x \cdot (d(x) - 1)) \rangle \in strc[d(x), M(r(x), V(x))]$.

If, for instance, $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$, $r(0 \cdot 1) = s_1$, $V(0 \cdot 1) = \sigma_2$, $M(s_1, \sigma_2) = \{rep\}$, $d(0 \cdot 1) = 4$, and $strc(4, rep) = \{\langle s_2, s_2, s_1, s_4 \rangle, \langle s_1, s_3, s_1, s_4 \rangle, \langle s_2, s_0, s_4, s_0 \rangle\}$ then r may continue according to one of the following possibilities, each satisfies $\langle r(0 \cdot 1 \cdot 0), r(0 \cdot 1 \cdot 1), r(0 \cdot 1 \cdot 2), r(0 \cdot 1 \cdot 3) \rangle \in strc(4, rep)$.

1. $r(0 \cdot 1 \cdot 0) = s_2$, $r(0 \cdot 1 \cdot 1) = s_2$, $r(0 \cdot 1 \cdot 2) = s_1$, and $r(0 \cdot 1 \cdot 3) = s_4$.
2. $r(0 \cdot 1 \cdot 0) = s_1$, $r(0 \cdot 1 \cdot 1) = s_3$, $r(0 \cdot 1 \cdot 2) = s_1$, and $r(0 \cdot 1 \cdot 3) = s_4$.
3. $r(0 \cdot 1 \cdot 0) = s_2$, $r(0 \cdot 1 \cdot 1) = s_0$, $r(0 \cdot 1 \cdot 2) = s_4$, and $r(0 \cdot 1 \cdot 3) = s_0$.

For a path π of T and a run r of U on V, we define $\mathrm{Inf}(r|\pi) = \{s : s \in S$ and for infinitely many $x \in \pi$, $r(x) = s\}$. Given an ABA U and a Σ-tree V, we say that U accepts V iff there exists a run r of U on V, such that for all paths π of T, $\mathrm{Inf}(r|\pi) \cap F \neq \phi$. That is, the ABAs use the simple Büchi acceptance condition. $\mathcal{L}(U)$ denotes the ABA language. I.e., $\mathcal{L}(U) = \{V : U$ accepts $V\}$.

2.2 The Temporal Logic CTL

The temporal logic CTL (Computation Tree Logic) provides branching time operators. A path quantifier, either A ("for all paths") or E ("for some path") must be immediately followed by exactly *one* of the linear-time operators X ("next time") or U ("until"). 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. We use the temporal operator $\tilde{U}$ (introduced as V at [CE81]) as a duality for the U operator, and assume that CTL formulas are given in 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 $E\varphi_1 \tilde{U} \varphi_2$, 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 $\varphi = E\varphi_1 U \varphi_2$. 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 $\varphi = \varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, $A\varphi_1 U \varphi_2$, $E\varphi_1 U \varphi_2$, $A\varphi_1 \tilde{U} \varphi_2$, or $E\varphi_1 \tilde{U} \varphi_2$ then $cl(\varphi) = \{\varphi\} \cup cl(\varphi_1) \cup cl(\varphi_2)$.
- If $\varphi = AX\varphi_1$ or $EX\varphi_1$ then $cl(\varphi) = \{\varphi\} \cup cl(\varphi_1)$.

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

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 $L : W \rightarrow 2^{AP}$ maps each state to a set of atomic propositions true in this state. A *path* in K is a sequence of states, $\pi = w_0, w_1, \ldots$ such that for every $i \geq 0$, $\langle w_i, w_{i+1} \rangle \in R$. We denote by $bd(w)$ the branching degree of the state w. Every computation tree can be associated with a Kripke structure in a straightforward way. When considering the satisfaction of a formula with respect to a computation tree, we refer to the Kripke structure associated with this tree. $w \models \varphi$ indicates that a formula φ holds at a state w (assuming an agreed structure K). For a Kripke structures K, $K \models \varphi$ iff $w_0 \models \varphi$. The formal definition of the relation $\models$ is omitted. Here we describe only the semantics of the $\tilde{U}$ operator. Typically, $w \models A\varphi_1 \tilde{U} \varphi_2$ 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 its size, U_ψ assumes simultaneous input trees. In this section we associate with each Kripke structure K, a simultaneous computation tree V_K.

Typically, given $K = \langle W, R, w_0, L \rangle$, V_K is obtained by unwinding K into a tree in which each node points, in addition to the simultaneous computation trees of its successors in K, also to two new copies of itself. In order to achieve this, we first construct a marked version R', of the transition relation of K. This involves adding two new self loops, marked 0 and 1, to every state $w \in W$, and marking the original transitions with $2, 3, \ldots, bd(w) + 1$. Formally, given R, R' is defined as follows: $R' = R'_1 \cup R'_2$, where,

$$R_1' = \bigcup_{w \in W} \{\langle w, w, 0\rangle, \langle w, w, 1\rangle\}$$

$$R_2' \subseteq \{\langle w_1, w_2, c\rangle : \langle w_1, w_2\rangle \in R\ ,\ c \in \{2, 3, \ldots, bd(w_1) + 1\}\}$$

such that for every $w_1 \in W$ and $c \in \{2, 3, \ldots, bd(w_1) + 1\}$, there exists a unique w_2 such that $\langle w_1, w_2, c\rangle \in R_2'$

Note that each Kripke structure may induce several marked versions, all having from each state w, $bd(w) + 2$ transitions, uniquely marked $0, 1, \ldots bd(w) + 1$. Below we define the simultaneous computation tree $V_{K'}$ of a marked Kripke structure $K' = \langle W, R', w_0, L\rangle$ of K. We start with the *state tree* of K' which is a Σ-tree $V_{st} : T_K \to W$, where T_K and V_{st} are inductively defined as follows:

- $\epsilon \in T_K$, and $V_{st}(\epsilon) = w_0$.
- For all $x \in T_K$ and for all $c \in \{0, 1, \ldots, bd(V_{st}(x)) + 1\}$, $x \cdot c \in T_K$, and $V_{st}(x \cdot c) = w$ iff $\langle V_{st}(x), w, c\rangle \in R'$.

$V_{K'} : T_K \to 2^{AP}$, the simultaneous computation tree of K', is a Σ-tree in which for every $x \in T_K$, $V_{K'}(x) = L(V_{st}(x))$. V_K, a simultaneous computation tree induced by K, is $V_{K'}$ for some possible marking K', of K. For our purposes, it does not matter which marking is chosen.

Example 3.1 Consider the Kripke structure K, presented in Figure 1-a. K has two possible marking versions, appearing in Figures 1-b and 1-c. Consider the

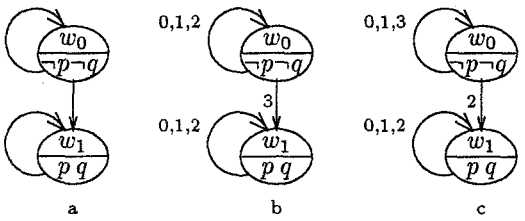

Figure 1: A Kripke structure K, and its possible marking versions.

version K', described at 1-b. T_K and $V_{st} : T_K \to W$, the state tree of K', appear in Figure 2. $V_{K'}$, the simultaneous computation tree of K', is obtained by replacing w_0 with $L(w_0)$, that is, with $\{\neg p, \neg q\}$, and replacing w_1 with $\{p, q\}$. $V_{K'}$ may serve as V_K, the simultaneous computation tree induced by K.

4 Representing a CTL Formula by an ABA

In this section, we present a method for associating with each CTL formula an ABA. Given a CTL formula ψ, we construct an ABA U_ψ, such that U_ψ accepts a simultaneous computation tree V_K iff K satisfies ψ. Thus, when considering simultaneous computation trees, the language of the ABA U_ψ is exactly the set of trees that originates from Kripke structures that satisfy ψ. Moreover, the size of U_ψ is linear in $|\psi|$.

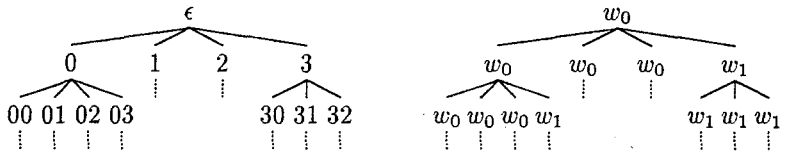

Figure 2: T_K and the simultaneous state tree of K.

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. Given a CTL formula ψ, we construct the ABA U_ψ in which each state is associated with a *single* formula $\varphi \in cl(\psi)$. Associating a state s_φ with a formula φ, means that the ABA, with s_φ as the initial state, accepts exactly all simultaneous trees satisfying φ. In order to determine whether a given tree V, satisfies φ, we check whether the immediate sub-trees of V satisfy one of the "inheritance packages" induced by $M(\varphi, \sigma)$. $M(\varphi, \sigma)$ associates with each immediate sub-tree of V, a formula that should hold on this sub-tree. Since V is a simultaneous tree, V^0 and V^1 are copies of V. Therefore, a requirement inherited to them is actually a requirement on V itself. $M(\varphi_1 \wedge \varphi_2, \sigma)$ for instance, inherits φ_1 to V^0, φ_2 to V^1, and t (i.e. no requirement) to all other immediate sub-trees.

The ABA associated with ψ is $U_\psi = \langle \mathcal{R}.strc, S, M, s_0, F \rangle$ where,

- $\mathcal{R} = S^3 \cup S^4$. That is, the ABAs for CTL use triples $\langle s_0, s_1, s_2 \rangle \in S^3$ or quadruples $\langle s_0, s_1, s_2, s_3 \rangle \in S^4$ as tuple representations.
- The function $strc : \mathbb{N} \times \mathcal{R} \to 2^{S^*}$, defined below, associates with each stretching factor d and such a representation, a set of d-tuples.

$$strc(d, \langle s_0, s_1, s_2 \rangle) = \{\langle \overbrace{s_0, s_1, s_2, s_2, \ldots, s_2}^{d} \rangle\}$$

$$\begin{aligned} strc(d, \langle s_0, s_1, s_2, s_3 \rangle) = \{ \; & \langle s_0, s_1, s_2, s_3, s_3, \ldots, s_3, s_3 \rangle, \\ & \langle s_0, s_1, s_3, s_2, s_3, \ldots, s_3, s_3 \rangle, \\ & \langle s_0, s_1, s_3, s_3, s_2, \ldots, s_3, s_3 \rangle, \\ & \vdots \\ & \langle s_0, s_1, s_3, s_3, s_3, \ldots, s_2, s_3 \rangle, \\ & \langle s_0, s_1, s_3, s_3, s_3, \ldots, s_3, s_2 \rangle \; \} \end{aligned}$$

  Note that $strc(d, \langle s_0, s_1, s_2, s_3 \rangle)$ contains all the $d-2$ possibilities of replacing a single s_3 with s_2 in $strc(d, \langle s_0, s_1, s_3 \rangle)$. Note also, that if required, the function $strc$, together with U_ψ, can be extended to handle also infinite stretching factors. Thus, amorphous automata gladly handle trees of infinite branching degrees.
- $S = cl(\psi)$. I.e., each state is a single formula from the closure of ψ.
- $s_0 = \psi$.
- $F = \{t\} \cup \{\varphi : \varphi \in cl(\psi) \text{ and } \varphi \text{ is an } \tilde{U}\text{-formula}\}$.
- $M : S \times \Sigma \to 2^{\mathcal{R}}$ where $\Sigma = 2^{AP}$.
  $M(\varphi, \sigma)$ is defined according to the structure of φ, as follows:
  - If φ is either $p, \neg p, t$, or f, where $p \in AP$,

$$M(\varphi, \sigma) = \begin{cases} \{\langle t, t, t \rangle\} & \text{if } \sigma \models \varphi \\ \{\langle f, f, f \rangle\} & \text{if } \sigma \not\models \varphi \end{cases}$$

- Otherwise, for all $\sigma \in \Sigma$,
  * $M(\varphi_1 \wedge \varphi_2, \sigma) = \{\langle \varphi_1, \varphi_2, t\rangle\}$
  * $M(\varphi_1 \vee \varphi_2, \sigma) = \{\langle \varphi_1, t, t\rangle, \langle \varphi_2, t, t\rangle\}$
  * $M(AX\varphi_1, \sigma) = \{\langle t, t, \varphi_1\rangle\}$
  * $M(EX\varphi_1, \sigma) = \{\langle t, t, \varphi_1, t\rangle\}$
  * $M(A\varphi_1 U \varphi_2, \sigma) = \{\langle \varphi_2, t, t\rangle, \langle \varphi_1, t, A\varphi_1 U \varphi_2\rangle\}$
  * $M(E\varphi_1 U \varphi_2, \sigma) = \{\langle \varphi_2, t, t\rangle, \langle \varphi_1, t, E\varphi_1 U \varphi_2, t\rangle\}$
  * $M(A\varphi_1 \tilde{U} \varphi_2, \sigma) = \{\langle \varphi_2, \varphi_1, t\rangle, \langle \varphi_2, t, A\varphi_1 \tilde{U} \varphi_2\rangle\}$
  * $M(E\varphi_1 \tilde{U} \varphi_2, \sigma) = \{\langle \varphi_2, \varphi_1, t\rangle, \langle \varphi_2, t, E\varphi_1 \tilde{U} \varphi_2, t\rangle\}$

  Note that when φ is not a proposition, $M(\varphi, \sigma)$ 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 ABAs 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 $\langle \varphi_2, \varphi_1, t\rangle$), 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 transitions induced by $\langle \varphi_2, t, E\varphi_1 \tilde{U} \varphi_2, t\rangle$). 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. In examples 4.1 and 4.2 below, we describe the ABAs of two CTL formulas.

Example 4.1 Consider the formula $\psi' = A((\neg AXp)Uq)$. Its positive normal form is $\psi = A((EX\neg p)Uq)$. The ABA associated with ψ is $U_\psi = \langle \mathcal{R}, strc, S, M, s_0, F\rangle$ where:

- $\mathcal{R}$ and $strc$ are as defined for ABAs of CTL.
- $S = \{\psi, EX\neg p, \neg p, q, t, f\}$.
- $s_0 = \psi$.
- $F = \{t\}$.
- M is described in the following table:

S	$M(S, \sigma)$ (for any $\sigma \subseteq \{p, q\}$)
ψ	$\{\langle q, t, t\rangle, \langle EX\neg p, t, \psi\rangle\}$
$EX\neg p$	$\{\langle t, t, \neg p, t\rangle\}$
$\neg p$	$\{\langle t, t, t\rangle\}$ if $p \notin \sigma$ $\{\langle f, f, f\rangle\}$ if $p \in \sigma$
q	$\{\langle t, t, t\rangle\}$ if $q \in \sigma$ $\{\langle f, f, f\rangle\}$ if $q \notin \sigma$
t	$\{\langle t, t, t\rangle\}$
f	$\{\langle f, f, f\rangle\}$

In the state ψ, choosing $\langle q, t, t\rangle$ means that the ABA guesses that q, the eventuality of ψ is satisfied in the present. On the other hand, choosing $\langle EX\neg p, t, \psi\rangle$ means that the ABA 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 ABA 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 $\psi = A(tUA(f\tilde{U}p))$. The ABA associated with ψ is $U_\psi = \langle \mathcal{R}, strc, S, M, s_0, F \rangle$ where:

- $\mathcal{R}$ and *strc* are as defined for ABAs of CTL.
- $S = \{\psi,\ A(f\tilde{U}p),\ p,\ t,\ f\}$.
- $s_0 = \psi$.
- $F = \{t,\ A(f\tilde{U}p)\}$.
- M is described in the following table:

S	$M(S,\sigma)$ (for any $\sigma \subseteq \{p,q\}$)
ψ	$\{\langle A(f\tilde{U}p), t, t\rangle, \langle t, t, \psi\rangle\}$
$A(f\tilde{U}p)$	$\{\langle p, f, t\rangle, \langle p, t, A(f\tilde{U}p)\rangle\}$
p	$\{\langle t,t,t\rangle\}$ if $p \in \sigma$ $\{\langle f,f,f\rangle\}$ if $p \notin \sigma$
t	$\{\langle t,t,t\rangle\}$
f	$\{\langle f,f,f\rangle\}$

As in the previous example, in the state ψ, the ABA may either guess that $A(f\tilde{U}p)$, the eventuality of ψ, is satisfied in the present, or choose $\langle t, t, \psi\rangle$ 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 $A(f\tilde{U}p)$, the ABA should accept only trees in which p is always true in all paths. Since once reaching the state f, the ABA stays there forever, choosing $\langle p, f, t\rangle$ never leads to an accepting run. Therefore, in an accepted run, the ABA must choose the second option, $\langle p, t, A(f\tilde{U}p)\rangle$, guaranteeing for satisfaction of p in the present and satisfaction of $A(f\tilde{U}p)$ in all successors. Here, the ABA has no choice but to keep visiting $A(f\tilde{U}p)$ forever; yet, since $A(f\tilde{U}p) \in F$, this is permitted.

We now refer to the size of U_ψ. We define the size of an ABA as the length of its encoding. For every ψ, the number of states required for U_ψ, $|cl(\psi)|$, is bounded by $|\psi| + 2$. Recall that $M(\varphi, \sigma)$ for a formula φ, which is not a proposition, is independent of σ and contains at most two tuple representations. Also, when φ is a proposition, there is no need to keep $M(\varphi, \sigma)$ for all σ. Thus, $|M|$ is $\mathcal{O}(|S|)$. Encoding $\mathcal{R}$ and *strc* requires a constant place. Therefore, the size of U_ψ is linear in $|\psi|$.

Theorem 4.3 *For every CTL formula ψ and for every Kripke structure K,*

1. *U_ψ accepts V_K iff K satisfies ψ.*
2. *The size of U_ψ is linear in $|\psi|$.*

5 CTL Model Checking Using ABAs

In this section we introduce a polynomial automata-based algorithm for the model checking problem for CTL. The model checking problem for a temporal logic is: Given a Kripke structure K and a temporal logic formula ψ, determine whether $K \models \psi$. When sequential automata are used to describe linear time formulas ([VW86b]), each Kripke structure may correspond to infinity 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 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. Since nodes of a given model have various and unknown in advance branching degrees, the amorphous nature of the ABAs is essential.

Given a CTL formula ψ, the language of U_ψ contains exactly all the simultaneous computation trees which originate from Kripke structures that satisfy ψ. Given a Kripke structure K, we construct an automaton U_K such that the language of U_K contains exactly one tree, V_K, the simultaneous computation tree induced by K. The language of U_ψ, intersected with that of U_K, may either contain a single tree, V_K, in which case $K \models \psi$, or be empty, in which case $K \not\models \psi$.

U_K is defined over trees which coincide in their branching degree with T_K. Thus, its branching degree may vary. Yet, each state has a fixed and known in advance branching degree. Each state in U_K is associated with a pair of a model state $w \in W$ and an acceptance (a) / rejection (r) flag. Being at state $\langle w, a\rangle$, U_K expects a computation tree V with a root labeled with $L(w)$ and $bd(w)+2$ immediate sub-trees which correspond to the successors of w in K'. Once failing to meet the expected label, U_ψ reaches a tuple of rejection states and keeps visiting only rejection states thereafter.

Formally, given V_K, which originates from the marked Kripke structure $K' = \langle W, R', w^0, L\rangle$, $U_K = \langle S, M, s_0, F\rangle$ is defined as follows.

- $S = W \times \{a, r\}$.
- $M : S \times \Sigma \to 2^{S^*}$.
  For every $w \in W$ such that for every $0 \le c \le bd(w)+1$, $\langle w, w_c, c\rangle \in R'$,
  $$M(\langle w, a\rangle, \sigma) = \begin{cases} \langle\langle w_0, a\rangle, \langle w_1, a\rangle, \ldots, \langle w_{bd(w)+1}, a\rangle\rangle & \text{if } \sigma = L(w). \\ \langle\langle w_0, r\rangle, \langle w_1, r\rangle, \ldots, \langle w_{bd(w)+1}, r\rangle\rangle & \text{if } \sigma \neq L(w). \end{cases}$$
  $M(\langle w, r\rangle, \sigma) = \langle\langle w_0, r\rangle, \langle w_1, r\rangle, \ldots, \langle w_{bd(w)+1}, r\rangle\rangle$.
- $s_0 = \langle w^0, a\rangle$.
- $F = W \times \{a\}$.

Example 5.1 The automaton associated with the Kripke structure K from example 3.1, is $U_K = \langle S, M, s_0, F\rangle$ where,

- $S = \{\langle w_0, a\rangle, \langle w_1, a\rangle, \langle w_0, r\rangle, \langle w_1, r\rangle\}$.
- $s_0 = \langle w_0, a\rangle$.
- $F = \{\langle w_0, a\rangle, \langle w_1, a\rangle\}$.
- $M : S \times \Sigma \to 2^{S^*}$ is described in the following table:

s	$M(s, \phi)$	$M(s, \{p, q\})$	$M(s, \{p\})$ and $M(s, \{q\})$
$s_0 = \langle w_0, a\rangle$	$\{\langle s_0, s_0, s_0, s_1\rangle\}$	$\{\langle s_2, s_2, s_2, s_3\rangle\}$	$\{\langle s_2, s_2, s_2, s_3\rangle\}$
$s_1 = \langle w_1, a\rangle$	$\{\langle s_3, s_3, s_3\rangle\}$	$\{\langle s_1, s_1, s_1\rangle\}$	$\{\langle s_3, s_3, s_3\rangle\}$
$s_2 = \langle w_0, r\rangle$	$\{\langle s_2, s_2, s_2, s_3\rangle\}$	$\{\langle s_2, s_2, s_2, s_3\rangle\}$	$\{\langle s_2, s_2, s_2, s_3\rangle\}$
$s_3 = \langle w_1, r\rangle$	$\{\langle s_3, s_3, s_3\rangle\}$	$\{\langle s_3, s_3, s_3\rangle\}$	$\{\langle s_3, s_3, s_3\rangle\}$

Lemma 5.2 *For every Kripke structure K,*

1. $\mathcal{L}(U_K) = \{V_K\}$.
2. *The size of U_K is linear in $|K|$.*

Given a Büchi tree automaton $U_1 = \langle S_1, M_1, s_{0_1}, F_1\rangle$ and an ABA $U_1 = \langle \mathcal{R}, strc, S_2, M_2, s_{0_2}, F_2\rangle$, we extend the known product of Büchi tree automata ([VW86a]), and define their product as a Büchi tree automaton $U = U_1 \times U_2$, that has the same branching degree as U_1. $U = \langle S, M, s_0, F\rangle$ is defined as follows:

- $S = S_1 \times S_2 \times \{1,2\}$.
- $s_0 = \langle s_{0_1}, s_{0_2}, 1\rangle$.
- $F = F_1 \times S_2 \times \{1\}$.
- $M : S \times \Sigma \to 2^{S^*}$, where for every $\langle s_1, s_2, l\rangle \in S$ and $\sigma \in \Sigma$,

$$\langle\langle s_1^1, s_2^1, l'\rangle, \langle s_1^2, s_2^2, l'\rangle, \ldots, \langle s_1^d, s_2^d, l'\rangle\rangle \in M(\langle s_1, s_2, l\rangle, \sigma) \text{ iff}$$

  - $\langle s_1^1, s_1^2, \ldots, s_1^d\rangle \in M_1(s_1, \sigma)$.
  - $\langle s_2^1, s_2^2, \ldots, s_2^d\rangle \in strc[d, M_2(s_2, \sigma)]$.
  - $l' = \begin{cases} 1 & \text{if either } (l = 1 \text{ and } s_1 \notin F_1) \text{ or } (l = 2 \text{ and } s_2 \in F_2). \\ 2 & \text{if either } (l = 1 \text{ and } s_1 \in F_1) \text{ or } (l = 2 \text{ and } s_2 \notin F_2). \end{cases}$

Lemma 5.3 *Given a Büchi tree automaton U_1 and an* ABA *U_2, their product automaton U satisfies:*

1. $\mathcal{L}(U) = \mathcal{L}(U_1) \cap \mathcal{L}(U_2)$.
2. $|U| = \mathcal{O}(|U_1| * |U_2|)$.

Given U_K and U_ψ, let $U = U_K \times U_\psi$. Note how the amorphous nature of U_ψ enables it to be intersected with any Büchi tree automaton U_K. Intuitively, U_ψ hides shrunk, lurking patiently for model automata, ready for everything ! When a victim gets close enough, it jumps on it, stretches properly, and quickly determines whether K satisfies ψ.

Theorem 5.4 *Given a CTL formula ψ and a Kripke structure K, $U = U_K \times U_\psi$ satisfies:*

1. *$\mathcal{L}(U) \neq \phi$ iff $K \models \psi$.*
2. *The nonemptiness of $\mathcal{L}(U)$ can be tested in time* poly$(|\psi| * |K|)$.

We conclude with the algorithm that follows. Given a CTL formula ψ and a Kripke structure K:

(1) Construct the ABA U_ψ.
(2) Construct V_K and accordingly, the Büchi tree automaton U_K.
(3) Construct $U = U_K \times U_\psi$.
(4) Output "Yes" if $\mathcal{L}(U) \neq \phi$, "No", otherwise.

6 CTL Satisfiability Decision Using ABAs

In this section we introduce a method for satisfiability decision of CTL formulas using ABAs . We present an extended automaton, D_ψ, that accepts exactly all trees that satisfy ψ. Satisfiability of ψ is reduced to the nonemptiness of $\mathcal{L}(D_\psi)$. Each state in the new automaton is associated with a set of formulas. Associating a state with a set of formulas s, means that D_ψ with s as the initial state, accepts exactly all trees satisfying all the formulas in s. Recall that in order to determine whether a given simultaneous tree V satisfies φ, U_ψ checks whether the immediate sub-trees of V satisfy one of the "inheritance packages" induced by $M(\varphi, \sigma)$. Since V^0 and V^1 are copies of V, requirements on V itself are handled easily. D_ψ does not have the luxuriance of these copies. Instead, $M(s, \sigma)$ of D_ψ, when run on V, imposes on the sub-trees of V both the requirements that U_ψ imposes on $V^2, \ldots, V^{d-1}$ and requirements that guarantee the satisfaction of those requirements imposed by U_ψ

on V^0 and V^1. This is done for every $\varphi \in s$. $M(\{p \vee AXp, q \wedge AXq\}, \sigma)$ for instance, inherits to all immediate sub-trees $\{f\}$, if $q \notin \sigma$, $\{q\}$ if both $p \in \sigma$ and $q \in \sigma$, and $\{p, q\}$ if $q \in \sigma$ and $p \notin \sigma$. This, however, results in a set of requirements for each sub-tree, instead of the single one imposed by U_ψ. Moreover, a more sophisticated handling of eventualities is needed. We first bring some definitions and notations required for the definition of D_ψ.

6.1 Definitions and Notations

In order to handle eventualities efficiently, D_ψ attributes the U-formulas of $cl(\psi)$ by either (a), noting the until is an *active* until, or by (d), noting it is a *dormant* until. We denote by U_a and U_d, active and dormant U-formulas, respectively. U-formulas which are not attributed, are treated as if attributed by (d). Semantically, U_a and U_d are both equal to a not attributed U.

$acl(\psi)$, the *attributed closure of* ψ, extends the regular closure of ψ by duplicating and attributing all the U-sub-formulas of ψ. That is, every $A\varphi_1 U \varphi_2 \in cl(\psi)$ is replaced by both $A\varphi_1 U_a \varphi_2$ and $A\varphi_1 U_d \varphi_2$ in $acl(\psi)$, and similarly for $E\varphi_1 U \varphi_2$. It is easy to see that even with this duplication of U-formulas, $|acl(\psi)| \leq |\psi| + 2$ for every ψ.

Intuitively, D_ψ handles eventualities by maintaining the following principle:

(1) Each dormant U-formula should eventually become active.
(2) Each active U-formula should eventually reach its eventuality.

The automaton keeps inheriting an active U-formula until it is satisfied. By the acceptance condition of D_ψ, a path is accepted only if the run on that path visits infinitely often states containing no active U-formulas. This ensures that (2) holds. Once a state with no active U-formula is visited, all its dormant U-formulas become active. This, together with the acceptance condition guarantee that (1) holds.

The principle expressed in (1) and (2) is similar to the one used in the sequential automaton for LTL presented in [WVS83]. However, while with each state in their automaton, a maximal consistent set of formulas is associated, our automaton directly implements the inheritance approach used by U_ψ, and with each state a minimal set of requirements is associated. Technically, their automaton is a composition of two automata over the alphabet $2^{cl(\psi)}$; one checks the local requirements and the other checks the eventuality requirements. D_ψ, on the other hand, handles both local and eventual requirements and is originally defined over the alphabet 2^{AP}.

To maintain (1), we define the function $awake : 2^{acl(\psi)} \rightarrow 2^{acl(\psi)}$. $awake$ activates all the U_d-formulas (if exist) in a set of attributed formulas, provided that no U_a-formula exists in the set. Formally, we first define $awake$ for singletons. $awake(\{A\varphi_1 U_d \varphi_2\}) = \{A\varphi_1 U_a \varphi_2\}$, $awake(\{E\varphi_1 U_d \varphi_2\}) = \{E\varphi_1 U_a \varphi_2\}$, and $awake(\{\varphi\}) = \{\varphi\}$ for all other formulas. We now extend it to sets of formulas as follows:

$$awake(s) = \begin{cases} s & \text{if there exists an } U_a\text{-formula } \varphi \in s. \\ \bigcup_{\varphi \in s} awake(\{\varphi\}) & \text{otherwise.} \end{cases}$$

Example 6.1

- $awake(\{A\varphi_1 U_a \varphi_2, E\varphi_3 U_d \varphi_4\}) = \{A\varphi_1 U_a \varphi_2, E\varphi_3 U_d \varphi_4\}$.
- $awake(\{AX A\varphi_1 U_a \varphi_2, E\varphi_3 U_d \varphi_4\}) = \{AX A\varphi_1 U_a \varphi_2, E\varphi_3 U_a \varphi_4\}$.

As tuple representations, D_ψ uses pairs of sets of attributed formulas from the attributed closure of ψ. Each tuple representation $\langle P, Q\rangle$, induces a nondeterministic choice for the next automaton's step. P contains requirements that should be inherited to *all* successors and Q contains requirements that should be inherited to *some* successor. The formulas in P originate from formulas of type $A\varphi$, while those in Q originate from formulas of type $E\varphi$.

We define the binary operators $\vee$ and $\wedge$ over sets of representations as follows ($\vee$ and $\wedge$ are commutative):

$$S_1 \vee S_2 = \begin{cases} \{\langle\{t\},\{t\}\rangle\} & \text{if } S_2 = \{\langle\{t\},\{t\}\rangle\}. \\ S_1 & \text{if } S_2 = \{\langle\{f\},\{f\}\rangle\}. \\ S_1 \cup S_2 & \text{otherwise.} \end{cases}$$

$$S_1 \wedge S_2 = \begin{cases} \{\langle\{f\},\{f\}\rangle\} & \text{if } S_2 = \{\langle\{f\},\{f\}\rangle\}. \\ S_1 & \text{if } S_2 = \{\langle\{t\},\{t\}\rangle\}. \\ \{\langle P_1 \cup P_2, Q_1 \cup Q_2\rangle : \langle P_1, Q_1\rangle \in S_1 \text{ and } \langle P_2, Q_2\rangle \in S_2\} & \text{otherwise.} \end{cases}$$

Note that $|S_1 \vee S_2| \leq |S_1| + |S_2|$ and $|S_1 \wedge S_2| \leq |S_1| * |S_2|$.

Example 6.2

Let $S_1 = \{\langle\{\varphi_1, \varphi_2\}, \{\varphi_3\}\rangle, \langle\{\varphi_4\}, \{\varphi_5\}\rangle\}$ and $S_2 = \{\langle\{\psi_1\}, \phi\rangle, \langle\{\psi_2, \psi_3\}, \{\psi_4\}\rangle\}$.
$S_1 \vee S_2 = \{\langle\{\varphi_1, \varphi_2\}, \{\varphi_3\}\rangle, \langle\{\varphi_4\}, \{\varphi_5\}\rangle, \langle\{\psi_1\}, \phi\rangle, \langle\{\psi_2, \psi_3\}, \{\psi_4\}\rangle\}$.
$S_1 \wedge S_2 = \{\langle\{\varphi_1, \varphi_2, \psi_1\}, \{\varphi_3\}\rangle, \langle\{\varphi_1, \varphi_2, \psi_2, \psi_3\}, \{\varphi_3, \psi_4\}\rangle, \langle\{\varphi_4, \psi_1\}, \{\varphi_5\}\rangle,$
$\langle\{\varphi_4, \psi_2, \psi_3\}, \{\varphi_5, \psi_4\}\rangle\}$.

6.2 The Extended Automaton

The extended ABA associated with ψ is $D_\psi = \langle \mathcal{R}, strc, S, M, s_0, F\rangle$ where,

- $\mathcal{R} = 2^{acl(\psi)} \times 2^{acl(\psi)}$.
- The function $strc : \mathbb{N} \times \mathcal{R} \to 2^{S^*}$, defined below, associates with each stretching factor d and such a representation, a set of d-tuples of states.

$$strc(d, \langle P, Q\rangle) = \{ \langle P \cup Q_1, P \cup Q_2, \ldots, P \cup Q_d\rangle : \bigcup_i Q_i = Q \text{ and for every } q \in Q \text{ there exists a unique } 1 \leq i \leq d \text{ such that } q \in Q_i \}$$

- $S = 2^{acl(\psi)}$. [1]
- $s_0 = \{\psi\}$.
- $F = \{s : s \neq \{f\}$ and contains no U_a-formulas$\}$.
- $M : S \times \Sigma \to 2^{\mathcal{R}}$ is defined by $M(s, \sigma) = \bigwedge_{\varphi \in awake(s)} h(\varphi, \sigma)$, where $h : acl(\psi) \times \Sigma \to 2^{\mathcal{R}}$ is inductively defined by the structure of φ as follows, [2]
  - If φ is either $p, \neg p, t$, or f, where $p \in AP$,
    $$h(\varphi, \sigma) = \begin{cases} \{\langle\{t\},\{t\}\rangle\} & \text{if } \sigma \models \varphi \\ \{\langle\{f\},\{f\}\rangle\} & \text{if } \sigma \not\models \varphi \end{cases}$$
  - Otherwise, for all $\sigma \in \Sigma$,
    - $h(\varphi_1 \wedge \varphi_2, \sigma) = h(\varphi_1, \sigma) \wedge h(\varphi_2, \sigma)$
    - $h(\varphi_1 \vee \varphi_2, \sigma) = h(\varphi_1, \sigma) \vee h(\varphi_2, \sigma)$
    - $h(AX\varphi_1, \sigma) = \{\langle\{\varphi_1\},\{t\}\rangle\}$
    - $h(EX\varphi_1, \sigma) = \{\langle\{t\},\{\varphi_1\}\rangle\}$
    - $h(A\varphi_1 U_a \varphi_2, \sigma) = h(\varphi_2, \sigma) \vee (h(\varphi_1, \sigma) \wedge \{\langle\{A\varphi_1 U_a \varphi_2\},\{t\}\rangle\})$

[1] Actually, the set of reachable states consists of only a small fragment of $2^{acl(\psi)}$.

[2] Recall that non-attributed U-formulas are treated by *awake* and h, as if attributed by (d).

* $h(A\varphi_1 U_d \varphi_2, \sigma) = h(\varphi_2, \sigma) \vee (h(\varphi_1, \sigma) \wedge \{\langle\{A\varphi_1 U_d \varphi_2\}, \{t\}\rangle\})$
* $h(E\varphi_1 U_a \varphi_2, \sigma) = h(\varphi_2, \sigma) \vee (h(\varphi_1, \sigma) \wedge \{\langle\{t\}, \{E\varphi_1 U_a \varphi_2\}\rangle\})$
* $h(E\varphi_1 U_d \varphi_2, \sigma) = h(\varphi_2, \sigma) \vee (h(\varphi_1, \sigma) \wedge \{\langle\{t\}, \{E\varphi_1 U_d \varphi_2\}\rangle\})$
* $h(A\varphi_1 \tilde{U} \varphi_2, \sigma) = h(\varphi_2, \sigma) \wedge (h(\varphi_1, \sigma) \vee \{\langle\{A\varphi_1 \tilde{U} \varphi_2\}, \{t\}\rangle\})$
* $h(E\varphi_1 \tilde{U} \varphi_2, \sigma) = h(\varphi_2, \sigma) \wedge (h(\varphi_1, \sigma) \vee \{\langle\{t\}, \{E\varphi_1 \tilde{U} \varphi_2\}\rangle\})$

Intuitively, each pair $\langle P, Q\rangle$, in $h(\varphi, L(w))$ is an inheritance package for a single formula. M both awakes the state s and composes the inheritance packages corresponding to all its awaked formulas. Following is the formal justification.

Lemma 6.3 *For every φ and w such that $h(\varphi, L(w)) = \bigcup_{i=1\ldots n}\{\langle P_i, Q_i\rangle\}$,*

$$w \models \varphi \Leftrightarrow \exists 1 \leq i \leq n: \quad \begin{array}{l} \forall \xi \in P_i, \forall w' : w \frown w', w' \models \xi \\ \forall \xi \in Q_i, \exists w' : w \rightarrow w', w' \models \xi \end{array}$$

Example 6.4 Consider the formula $\psi = A((EX\neg p)Uq)$ from example 4.1.

- $acl(\psi) = \{\psi_a, \psi_d, EX\neg p, \neg p, q, t, f\}$.
- We describe here the function h when operates on formulas from $acl(\psi)$.

φ	$h(\varphi, \{p, q\})$	$h(\varphi, \{p, \neg q\})$	$h(\varphi, \{\neg p, q\})$	$h(\varphi, \{\neg p, \neg q\})$
ψ_a	$\{\langle\{t\}, \{t\}\rangle\}$	$\{\langle\{\psi_a\}, \{\neg p\}\rangle\}$	$\{\langle\{t\}, \{t\}\rangle\}$	$\{\langle\{\psi_a\}, \{\neg p\}\rangle\}$
$EX\neg p$	$\{\langle\{t\}, \{\neg p\}\rangle\}$	$\{\langle\{t\}, \{\neg p\}\rangle\}$	$\{\langle\{t\}, \{\neg p\}\rangle\}$	$\{\langle\{t\}, \{\neg p\}\rangle\}$
$\neg p$	$\{\langle\{f\}, \{f\}\rangle\}$	$\{\langle\{f\}, \{f\}\rangle\}$	$\{\langle\{t\}, \{t\}\rangle\}$	$\{\langle\{t\}, \{t\}\rangle\}$
q	$\{\langle\{t\}, \{t\}\rangle\}$	$\{\langle\{f\}, \{f\}\rangle\}$	$\{\langle\{t\}, \{t\}\rangle\}$	$\{\langle\{f\}, \{f\}\rangle\}$

The extended ABA associated with ψ is $D_\psi = \langle \mathcal{R}, strc, S, M, s_0, F\rangle$ where:

- $\mathcal{R}$ and *strc* are as defined for extended ABAs of CTL.
- $S = 2^{acl(\psi)}$ and the set of reachable states is $\{\{\psi_a\}, \{\psi_a, \neg p\}, \{t\}, \{f\}\}$.
- $s_0 = \{\psi\}$.
- $F = \{\{t\}\}$.
- M is described in the following table:

s	$M(s, \{p, q\})$	$M(s, \{p, \neg q\})$	$M(s, \{\neg p, q\})$	$M(s, \{\neg p, \neg q\})$
$\{\psi_a\}$	$\{\langle\{t\}, \{t\}\rangle\}$	$\{\langle\{\psi_a\}, \{\neg p\}\rangle\}$	$\{\langle\{t\}, \{t\}\rangle\}$	$\{\langle\{\psi_a\}, \{\neg p\}\rangle\}$
$\{\psi_a, \neg p\}$	$\{\langle\{f\}, \{f\}\rangle\}$	$\{\langle\{f\}, \{f\}\rangle\}$	$\{\langle\{t\}, \{t\}\rangle\}$	$\{\langle\{\psi_a\}, \{\neg p\}\rangle\}$
$\{t\}$	$\{\langle\{t\}, \{t\}\rangle\}$	$\{\langle\{t\}, \{t\}\rangle\}$	$\{\langle\{t\}, \{t\}\rangle\}$	$\{\langle\{t\}, \{t\}\rangle\}$
$\{f\}$	$\{\langle\{f\}, \{f\}\rangle\}$	$\{\langle\{f\}, \{f\}\rangle\}$	$\{\langle\{f\}, \{f\}\rangle\}$	$\{\langle\{f\}, \{f\}\rangle\}$

When D_ψ is in state $\{\psi_a, \neg p\}$ for instance, it accepts only trees that satisfy both ψ_a and $\neg p$. According to the label of the root of the tree, which express the present state, it inherits either $\{t\}$ to all its successors, if both formulas are satisfied in the present, $\{f\}$ to all its successors, if $\neg p$ is not satisfied in the present, or $\{\psi_a\}$ to all its successors and, in addition, $\{\neg p\}$ to one of them (for guaranteeing satisfaction of $EX\neg p$ in the present), if $\neg p$ is satisfied in present, but q, the eventuality of ψ_a is not. The last possibility is represented by $\langle\{\psi_a\}, \{\neg p\}\rangle$, for which $strc(d, \langle\{\psi_a\}, \{\neg p\}\rangle) = \{\langle\{\psi_a, \neg p\}, \{\psi_a\}, \overset{d}{\ldots}, \{\psi_a\}\rangle, \overset{d}{\ldots}, \langle\{\psi_a\}, \{\psi_a\}, \overset{d}{\ldots}, \{\psi_a, \neg p\}\rangle\}$.

Note how the function *strc* distributes each inheritance package among the d successors so that exactly all the possibilities for checking the satisfaction of the formulas in s (according to Lemma 6.3) are considered.

Theorem 6.5 *For every CTL formula ψ and for every computation tree V, D_ψ accepts V iff V satisfies ψ.*

Note that by its amorphous nature, the extended ABA D_ψ, accepts all trees that satisfy ψ and therefore may be used for model checking. This however is of no practical interest since it induces an exponential time procedure.

By the "sufficient branching degree property" of CTL ([Wo87], [Em90]), a CTL formula ψ is satisfiable iff it is satisfiable in an infinite tree of a branching degree $|\psi|$. We can therefore ignore the amorphous nature of D_ψ and restrict the discussion to a conventional Büchi tree automaton defined over computation trees of degree $|\psi|$. The transition from an ABA to a Büchi automaton is straightforward. Let $\mathcal{L}^d(D_\psi)$ denote the ABA language when restricted only to Σ-trees with a fixed branching degree d. Theorem 6.5 together with the sufficient branching degree property imply the following Theorem.

Theorem 6.6 *For every CTL formula ψ, ψ is satisfiable iff $\mathcal{L}^{|\psi|}(D_\psi) \neq \phi$.*

We now refer to the size of D_ψ when adjusted to trees of branching degree $|\psi|$. Since for every ψ, $|acl(\psi)| \leq |\psi| + 2$, the number of states required for D_ψ is exponential in $|\psi|$. Therefore, so is $|M|$, and consequently, also $|D_\psi|$. Since the nonemptiness problem for Büchi tree automata is in PTIME ([VW86a]), we can conclude with the following theorem:

Theorem 6.7 *Satisfiability of CTL, using Büchi tree automata, is one exponential decidable.*

7 Directions for Future Research

The two new notions, amorphous automaton and simultaneous tree, introduced in this paper both raise some interesting questions. We discuss here briefly those related to amorphous automata.

In this work we demonstrate the essence of amorphous automata for an automata-based model checking for the branching-time temporal logic CTL. Nevertheless, the idea of an automaton with a flexible branching degree is applicable to any branching-time temporal logic. The automata in [VW86a] and [ES84] that handle satisfiability of variants of PDL and CTL*, respectively, can be easily redefined, adopting the amorphous nature of the ABAs. This however, just like our D_ψ, results in automata which are too big to handle model checking efficiently. The interesting point is to combine the automata with an appropriate representation of the model, as the simultaneous tree method introduced in this work.

Each amorphous automaton U has a stretching function *strc*. In this work we used a simple *strc* functions and apply them with finite stretching factors. This, however, is not essential. We can, for instance, define $strc(d, rep)$ to include d-tuples induced by all the words of length d derived by the context-free language *rep* (possibly, $d = \omega$). What are the connections between the complexity of computing *strc* and solving the nonemptiness problem of U ? Are there cases in which the nonemptiness problem is "cheaper" ? Are there cases in which the nonemptiness problem is decidable while *strc* is not ?

In [Ha82], Harel introduces the infinite tree recurrence lemma, stating that well-founded ω-branching recursive trees are isomorphic to recurrence-free finite-branching marked recursive trees. Can we associate with each finite-branching automaton U_1, over recurrence-free finite-branching marked recursive trees, an amorphous automaton U_2, suitable for finite paths but with a stretching factor ω, such

that $\mathcal{L}(U_2)$ contains exactly all the trees isomorphic to those in $\mathcal{L}(U_1)$? What are the connections between the acceptance condition of U_1 and the function *strc* of U_2 ?

As discussed in [ES84], the difference in the objects over which branching time logics and conventional tree automata are interpreted (trees with possibly infinite branching degree, versus trees of fixed and finite branching degree), impose technical problems in finding automata-theoretic characterization of the expressive power of branching time logics. Currently, discussion is restricted to trees of a fixed branching degree. Amorphous automata enable a non-restricted discussion.

Acknowledgment

We thank Moshe Vardi for many helpful suggestions and discussions.

References

[CE81] Clarke E.M. and Emerson E.A., Design and synthesis of synchronization skeletons using branching time temporal logic. *Proc. Workshop on Logic of Programs, LNCS 131* (1981) 52-71

[Em90] Emerson,E.A., Temporal and modal logic. *Handbook of theoretical computer science.* (1990) 997-1072 North-Holland.

[ES84] Emerson,E.A. and Sistla A.P., Deciding full Branching Time Logics. *Inform. and Control 61(3)* (1984) 175-201.

[Ha82] Harel D., Effective Transformations on Infinite Trees, with Applications to High Undecidability, Dominoes and Fairness. *J. Assoc. Comput. Mach. 33* (1986) 224-248.

[Pn77] Pnueli A., The temporal logic of programs. *Proc. 18th Symp. on Foundation of Computer Science.* (1977) 46-57.

[Ra69] Rabin M.O., Decidability of second order theories and automata on infinite trees. *Trans. AMS, 141* (1969) 1-35.

[Th90] Thomas W., Automata on Infinite Objects. *Handbook of theoretical computer science.*(1990) 165-191, North-Holland.

[Va89] Vardi M.Y., Automata theory for database theoreticians. *Proc. 8th ACM Symp. on Principles of Data Systems,* (1989) 83-92.

[Va91] Vardi M.Y., Verification of concurrent programs: the automata-theoretic framework. *Annals of Pure and Applied Logic 51* (1991) 79-98.

[VW86a] Vardi M.Y. and Wolper P., Automata theoretic techniques for modal logics of programs. *Journal of computer and system science 32.* (1986) 183-221

[VW86b] Vardi M.Y. and Wolper P., An Automata theoretic approach to automatic program verification. *Proc. Symp. on Logic in Computer Science,* (1986) 322-331

[Wo87] Wolper P., On the relations of programs and computations to models of temporal logic, *Temporal Logic in Specification, LNCS 398,* (1989) 75-123

[WVS83] Wolper P., Vardi M.Y., and Sistla P., Reasoning about infinite computation paths, *Proc. 24th Symp. on Foundations of Computer Science,* (1983) 185-194.