Verification

Javier Esparza
TU München

Orna Kupferman
The Hebrew University

Moshe Y. Vardi
Rice University

1 Introduction

This chapter describes the automata-theoretic approach to the satisfiability and model-checking problems for temporal logics. In a nutshell, the approach reduces these problems to standard decision problems about automata, like nonemptiness, language containment, or membership (whether a given object is accepted by a given automaton). These problems are solved using results of automata theory, which leads to algorithms for satisfiability and model-checking.

Temporal logics are modal logics for the description of the temporal ordering of events. They have become one of the most popular and flexible formalisms for specifying the behavior of concurrent programs [Pnu77, MP92]. In the early 80s, algorithmic methods were proposed for automatically verifying temporal logic properties of *finite-state* programs [QS82, LP85, CES86, VW86a]. (A state of a program is a complete description of its status, including the assignment of values to variables, the value of the program counter, and the like. Finite-state programs have finitely many possible states. Many hardware designs, synchronization and communication protocols, abstract versions of device drivers and many other systems can be modeled as finite-state programs.) The behavior of a finite-state program can be formalized as a finite *propositional Kripke structure*, and its *desired* behavior as a formula of propositional temporal logic. In order to verify the correctness of the program, one checks that its associated Kripke structure is a model of (satisfies) the formula. In other words, the problem of verifying whether a given finite-state program behaves as desired is reduced to the *model-checking* problem for the temporal logic. Extensive introductions to model checking can be found in [CGP99, BK08].

Temporal logics can describe time as linear or branching. In linear-time logics, each moment in time has a unique possible future, while in branching-time logics, each moment in time may split into several possible futures. (For an extensive discussion of various temporal logics, see [Eme90].) For both types, a close and fruitful connection with the theory of automata on infinite structures has been developed. The central idea is to associate with each temporal logic formula a finite automaton on infinite structures recognizing the computations that satisfy the formula. For linear temporal logic the structures are infinite words [Sis83, LPZ85, SVW87, VW94], while for branching temporal logic the structures are infinite trees [ES84, SE84, Eme85, VW86b, EJ88]. Once this has been achieved, the satisfiability problem for a logic reduces to the nonemptiness problem for its corresponding class of automata. The model-checking problem reduces to the language containment problem or to the membership problem, depending on the logic.

In the 80s and the first half of the 90s, the literature produced direct translations from temporal logic formulas to nondeterministic automata (cf. [VW86b, VW94, GPVW95]). However, for branching time logics this translations did not lead to asymptotically optimal algorithms: in particular, algorithms for branching-time logics derived from these translations were exponential, while other approaches only required linear time. Work carried out since the second half of the 90s has solved this problem by splitting the translation into two steps: a first translation of temporal formulas into alternating automata, followed by a translation of alternating into nondeterministic automata [Var94, KVW00]. The existential and universal states of alternating automata match the disjunctive and conjunctive operators of the logic, which makes the first translation simple and succinct: the size of the alternating automaton is linear in

the size of the formula [MSS88, KVW00, EJ91, Var94]. The two steps also decouple the logical and the combinatorial part of the problem: the translations from formulas to automata handle the logical part, while the combinatorics are handled by automata constructions.

The chapter is divided into two parts, corresponding to linear-time and branching-time logics. In the first part we present a translation of the logic LTL [Pnu81] into alternating Büchi word automata. The second part contains translations of the logics CTL, CTL*, and the propositional μ-calculus into different classes of symmetric alternating tree automata: *weak* automata for CTL, *hesitant* automata for CTL*, and *parity* automata for the μ-calculus.

Historical Note: The connection between logic and automata goes back to work in the early 1960s [Büc60, Elg61, Tra62] on monadic second-order logic and automata over finite words. This was extended in [Büc62] to infinite words, in [Don65, TW68] to finite trees, and in [Rab69] to infinite trees. As temporal logics can be expressed in first-order or monadic second-order logic [Kam68, HT87], the connection between monadic second-order logic and automata yields a connection between temporal logics and automata. Developing decision procedures that go via monadic second-order logic was a standard approach in the 1970s, see [Gab72]. A direct translation to automata was proposed first in [Str82] in the context of propositional dynamic logic. A direct translation from temporal logic to automata was first given in [WVS83] (see also [VW94] for linear time and [VW84] for branching time). The translation to alternating automata was first proposed in [MSS88] and pursued further in [Var94, Var95, KVW00].

2 Linear-time Logics

2.1 Linear temporal logic

The logic *LTL* is a linear temporal logic [Pnu81]. Formulas of LTL are constructed from a set AP of atomic propositions using the usual Boolean operators and the temporal operators X ("next time") and U ("until"). Formally, an LTL formula over AP is defined as follows:

- **true**, **false**, or p, for $p \in AP$.
- $\neg\psi_1$, $\psi_1 \wedge \psi_2$, $X\psi_1$, or $\psi_1 U \psi_2$, where ψ_1 and ψ_2 are LTL formulas.

The logic LTL is used for specifying properties of reactive systems. The systems are modeled by *Kripke structures*, and the semantics of LTL is defined with respect to infinite *computations*, modeled by infinite paths in Kripke structures. Formally, a Kripke structure is $K = \langle AP, W, R, W_0, \ell\rangle$, where AP is the set of atomic propositions, W is a set of states, $R \subseteq W \times W$ is a total transition relation (that is, for every state $w \in W$ there is at least one state $w' \in W$ such that $R(w, w')$), $W_0 \subseteq W$ is a set of initial states, and $\ell : W \to 2^{AP}$ maps each state to the set of atomic propositions that hold in this state.

A *path* of the Kripke structure K is an infinite sequence $w_0, w_1, \ldots$ of states such that $w_0 \in W_0$ and $R(w_i, w_{i+1})$ for all $i \geq 0$. A *computation* over AP is an infinite word over the alphabet 2^{AP}, namely a sequence of truth assignments to the atomic propositions in AP. Every path $w_0, w_1, \ldots$ of K induces the computation $\ell(w_0), \ell(w_1), \ldots$ of K.

Consider a computation $\pi = \pi_0, \pi_1, \pi_2, \ldots$, where for every $j \geq 0$, the set $\pi_j \subseteq AP$ is the set of atomic propositions that hold in the j-th position of π. We denote the suffix $\pi_j, \pi_{j+1}, \ldots$ of π by π^j. We write $\pi \models \psi$ to denote that the computation π satisfies the LTL formula ψ. The relation $\models$ is inductively defined as follows:

- For all π, we have $\pi \models$ **true** and $\pi \not\models$ **false**.
- For an atomic proposition $p \in AP$, we have $\pi \models p$ iff $p \in \pi_0$.
- $\pi \models \neg\psi_1$ iff $\pi \not\models \psi_1$.

- $\pi \models \psi_1 \wedge \psi_2$ iff $\pi \models \psi_1$ and $\pi \models \psi_2$.
- $\pi \models X\psi_1$ iff $\pi^1 \models \psi_1$.
- $\pi \models \psi_1 U \psi_2$ iff there exists $k \geq 0$ such that $\pi^k \models \psi_2$ and $\pi^i \models \psi_1$ for all $0 \leq i < k$.

Each LTL formula ψ over AP defines a language $L_\psi \subseteq (2^{AP})^\omega$ of the computations that satisfy ψ, Formally,

$$L_\psi = \{\pi \in (2^{AP})^\omega | \pi \models \psi\}.$$

We use the following abbreviations in writing formulas:

- $\vee, \rightarrow$, and $\leftrightarrow$, interpreted in the usual way.
- $\psi_1 R \psi_2 = \neg((\neg\psi_1) U (\neg\psi_2))$. That is, $\psi_1 R \psi_2$ is such that the operator R ("release") dualizes the operator U.
- $F\psi = \mathbf{true} U \psi$ ("eventually", where "F" stands for "Future").
- $G\psi = \neg F \neg\psi$ ("always", where "G" stands for "Globally"). Equivalently, $G\psi = \mathbf{false} R \psi$.

Example 2.1 We use LTL to formalize some desirable properties of a mutual exclusion algorithm for two processes, Process 0 and Process 1. Let AP contain the atomic propositions (with $i \in \{0, 1\}$) cs_i (Process i is in its critical section) and try_i (Process i tries to enter its critical section).

- The *mutual exclusion* property states that Process 0 and Process 1 are never simultaneously in their critical sections. We can express it using the LTL formula

  $$\psi_{me} = G((\neg cs_0) \vee (\neg cs_1)).$$

  Note we could have used several equivalent formulas, like $G(cs_0 \rightarrow \neg cs_1)$ or $\neg F(cs_0 \wedge cs_1)$. The corresponding language L_{me} contains all computations having no occurrences of letters (elements of 2^{AP}) containing both cs_0 and cs_1. Thus,

  $$L_{me} = \{\pi \in (2^{AP})^\omega \mid \text{ for all } j \geq 0, \text{ we have } cs_0 \notin \pi_j \text{ or } cs_1 \notin \pi_j\}.$$

- The *finite waiting* property for Process i states that if Process i tries to access its critical section, it eventually will. In LTL, we have

  $$\psi^i_{fw} = G(try_i \rightarrow F\, cs_i).$$

  The corresponding language L^i_{fw} contains all computations in which every occurrence of a letter containing try_i is followed later by an occurrence of some letter containing cs_i.

  $$L^i_{fw} = \{\pi \in (2^{AP})^\omega \mid \text{ for all } j \geq 0, \text{ if } try_i \in \pi_j, \text{ then there is } k > j \text{ such that } cs_i \in \pi_k)\}.$$

- The *access only after trying* property for Process i states that Process i enters its critical section only after it has tried to enter it. In LTL,

  $$\psi^i_{at} = ((\neg cs_i) U\, try_i) \vee G \neg cs_i.$$

  The corresponding language L^i_{at} contains all computations in which every occurrence of a letter containing cs_i is preceded by an occurrence of some letter containing try_i.

  $$L^i_{at} = \{\pi \in (2^{AP})^\omega \mid \text{ for all } j \geq 0, \text{ if } cs_i \in \pi_j, \text{ then there is } k \leq j \text{ such that } try_i \in \pi_k\}.$$

□

2.2 Alternating Büchi word automata

For a finite alphabet Σ, a word $w = \sigma_0\sigma_1\cdots$ is a (finite or infinite) sequence of letters from Σ. A property of a system with a set AP of atomic propositions can be viewed as a language over the alphabet 2^{AP}. We have seen in Section 2.1 that LTL can be used to formalize properties. Another way to define properties is to use automata.

A *nondeterministic finite automaton* is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha\rangle$, where Σ is a finite nonempty alphabet, Q is a finite nonempty set of *states*, $Q_0 \subseteq Q$ is a nonempty set of *initial states*, $\delta : Q \times \Sigma \to 2^Q$ is a *transition function*, and α is an *acceptance condition*.

Intuitively, when an automaton $\mathcal{A}$ runs on an input word over Σ, it starts in one of the initial states, and it proceeds along the word according the transition function. Thus, $\delta(q, \sigma)$ is the set of states that $\mathcal{A}$ can move into when it is in state q and it reads the letter σ. Note that the automaton may be *nondeterministic*, since it may have many initial states and the transition function may specify many possible transitions for each state and letter. The automaton $\mathcal{A}$ is *deterministic* if $|Q_0| = 1$ and $|\delta(q, \sigma)| \leq 1$ for all states $q \in Q$ and symbols $\sigma \in \Sigma$.

Formally, a *run* r of $\mathcal{A}$ on a finite word $w = \sigma_1 \cdots \sigma_n \in \Sigma^*$ is a sequence $q_0, q_1, \ldots, q_n$ of $n + 1$ states in Q such that $q_0 \in Q_0$, and $q_{i+1} \in \delta(q_i, \sigma_{i+1})$ for all $0 \leq i < n$. Note that a nondeterministic automaton can have many runs on a given input word; in contrast, a deterministic automaton can have at most one. If the input word is infinite, then a run of $\mathcal{A}$ on it is an infinite sequence of states. The acceptance condition α determines which runs are accepting. For automata on finite words, $\alpha \subseteq Q$ and a run r is *accepting* if $q_n \in \alpha$. Otherwise, r is *rejecting*. For automata on infinite words, one can consider several acceptance conditions. In the Büchi acceptance condition, $\alpha \subseteq Q$, and a run r is accepting if it visits some state in α infinitely often. Formally, let $inf(r) = \{q : q_i = q \text{ for infinitely many } i\text{'s }\}$. Then, r is accepting iff $inf(r) \cap \alpha \neq \emptyset$. A nondeterministic automaton $\mathcal{A}$ accepts a word w if there is an accepting run of $\mathcal{A}$ on w. A *universal* automaton has the same components as a nondeterministic one, but it accepts a word w if all its runs on w are accepting.

We now turn to define alternating automata. We first need some notations. For a given set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$), where we also allow the formulas **true** and **false**. For $Y \subseteq X$, we say that Y *satisfies* a formula $\theta \in \mathcal{B}^+(X)$ iff the truth assignment that assigns *true* to the members of Y and assigns *false* to the members of $X \setminus Y$ satisfies θ. For example, the sets $\{q_1, q_3\}$ and $\{q_2, q_3\}$ both satisfy the formula $(q_1 \vee q_2) \wedge q_3$, while the set $\{q_1, q_2\}$ does not satisfy this formula.

Consider an automaton $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha\rangle$. We can represent δ using $\mathcal{B}^+(Q)$. For example, a transition $\delta(q, \sigma) = \{q_1, q_2, q_3\}$ of a nondeterministic automaton $\mathcal{A}$ can be written as $\delta(q, \sigma) = q_1 \vee q_2 \vee q_3$. If $\mathcal{A}$ is universal, the transition can be written as $\delta(q, \sigma) = q_1 \wedge q_2 \wedge q_3$. While transitions of nondeterministic and universal automata correspond to disjunctions and conjunctions, respectively, transitions of alternating automata can be arbitrary formulas in $\mathcal{B}^+(Q)$. We can have, for instance, a transition $\delta(q, \sigma) = (q_1 \wedge q_2) \vee (q_3 \wedge q_4)$, meaning that the automaton accepts a suffix w^i of w from state q, if it accepts w^{i+1} from both q_1 and q_2 or from both q_3 and q_4. Such a transition combines existential and universal choices.

Formally, an *alternating automaton on infinite words* is a tuple $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha\rangle$, where Σ, Q, and α are as in nondeterministic automata, $q_{in} \in Q$ is an initial state (we will later explain why it is technically easier to assume a single initial state), and $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$ is a transition function. In order to define runs of alternating automata, we first have to define trees and labeled trees. Given a set Υ of directions, an Υ-*tree* is a prefix-closed set $T \subseteq \Upsilon^*$. Thus, if $x \cdot c \in T$ where $x \in \Upsilon^*$ and $c \in \Upsilon$, then also $x \in T$. The elements of T are called *nodes*, and the empty word ε is the *root* of T. For every $x \in T$ and $c \in \Upsilon$, the node $x \cdot c$ is a *successor* of x. The number of successors of x is called the *degree* of x and is denoted by $d(x)$. A node is a *leaf* if it has no successors. We sometimes refer to the length $|x|$ of x as its *level* in the tree. A *path* of an Υ-tree T is a set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $x \in \pi$, either x is a leaf or there exists a unique $c \in \Upsilon$ such that $x \cdot c \in \pi$. Given an alphabet Σ, a Σ-*labeled* Υ-*tree* is a pair $\langle T, V\rangle$ where T is an Υ-tree and $V : T \to \Sigma$ maps each node of T to a letter of Σ.

While a run of a nondeterministic automaton on an infinite word is an infinite sequence of states, a run of an alternating automaton is a Q-labeled $\mathbb{N}$-tree. Formally, given an infinite word $w = \sigma_0\sigma_1\cdots$, a run of $\mathcal{A}$ on w is a Q-labeled $\mathbb{N}$-tree $\langle T_r, r\rangle$ such that the following two conditions hold:

- $\varepsilon \in T_r$ and $r(\varepsilon) = q_{in}$.
- If $x \in T_r$, $r(x) = q$, and $\delta(q, \sigma_{|x|}) = \theta$, then there is a (possibly empty) set $S = \{q_1, \ldots, q_k\}$ such that S satisfies θ and for all $1 \leq c \leq k$, we have $x \cdot c \in T_r$ and $r(x \cdot c) = q_c$.

For example, if $\delta(q_{in}, \sigma_0) = (q_1 \vee q_2) \wedge (q_3 \vee q_4)$, then every run of $\mathcal{A}$ on w has a root labeled q_{in}, a node in level 1 labeled q_1 or q_2, and another node in level 1 labeled q_3 or q_4. Note that if $\theta = \mathbf{true}$, then x need not have children. This is the reason why T_r may have leaves. Also, since no set S satisfies $\theta = \mathbf{false}$, no run ever takes a transition with $\theta = \mathbf{false}$.

A run $\langle T_r, r\rangle$ is *accepting* iff all its infinite paths, which are labeled by words in Q^ω, satisfy the acceptance condition. A word w is accepted iff there exists an accepting run on it. Note that while conjunctions in the transition function of $\mathcal{A}$ are reflected in branches of $\langle T_r, r\rangle$, disjunctions are reflected in the fact we can have many runs on the same word. The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of infinite words that $\mathcal{A}$ accepts.

We define the *size* $|\mathcal{A}|$ of an automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha\rangle$ as $|Q| + |\delta| + |F|$, where $|Q|$ and $|\alpha|$ are the respective cardinalities of the sets Q and α, and where $|\delta|$ is the sum of the lengths of formulas that appear as $\delta(q, \sigma)$ for some $q \in Q$ and $\sigma \in \Sigma$.

Example 2.2 We describe an alternating Büchi automaton $\mathcal{A}_n$ over the alphabet $\Sigma_n = \{1, 2, \ldots, n\}$ such that $\mathcal{A}_n$ accepts exactly all words containing the subword i^3 for all letters $i \in \Sigma$.

Let $\mathcal{A}_n = \langle \Sigma_n, Q_n, q_{in}, \delta, \emptyset\rangle$, where

- $Q_n = \{q_{in}\} \cup (\Sigma \times \{3, 2, 1\})$. Thus, in addition to an initial state, the automaton $\mathcal{A}_n$ contains three states for each letter. Intuitively, the automaton is going to spawn into n different copies, with copy i waiting for the subword i^3 using the states $\langle i, 3\rangle$, $\langle i, 2\rangle$, and $\langle i, 1\rangle$.
- In its first transition, $\mathcal{A}_n$ spawn into n copies, taking the first letter into an account. Thus, for all $i \in \Sigma$, we have $\delta(q_{in}, i) = \langle i, 2\rangle \wedge \bigwedge_{j \neq i} \langle j, 3\rangle$. In addition, for all $i \in \Sigma$ and $c \in \{3, 2, 1\}$, we have

$$\delta(\langle i, c\rangle, j) = \begin{cases} \langle i, c-1\rangle & \text{if } j = i \text{ and } c \in \{3, 2\}, \\ \mathbf{true} & \text{if } j = i \text{ and } c = 1, \\ \langle i, 3\rangle & \text{if } j \neq i. \end{cases}$$

Observe that the number of states in a nondeterministic automaton for the language is exponential in n, as it has to remember the subsets of letters for which the required subword has already appeared. □

2.3 Alternation removal

The rich structure of alternating automata makes them exponentially more succinct than nondeterministic automata. On the other hand, reasoning about alternating automata is complicated. For example, while the nonemptiness of a nondeterministic automaton is independent of the labeling of its transitions, this is no longer true for alternating automata, and this fact has important consequences: nonemptiness is NLOGSPACE-complete for nondeterministic automata but PSPACE-complete for alternating automata. Thus, many algorithms for alternating automata involve alternation removal – a translation to an equivalent nondeterministic automaton. Below we describe such a translation for the case of Büchi automata.

Theorem 2.3 [MH84] *Let $\mathcal{A}$ be an alternating Büchi automaton. There is a nondeterministic Büchi automaton $\mathcal{A}'$, with exponentially many states, such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

Proof: The automaton $\mathcal{A}'$ guesses a run tree of $\mathcal{A}$ and simultaneously, checks if the run tree contains infinitely many "checkpoints": levels between which every path in the run of $\mathcal{A}$ visits α at least once. The first checkpoint is the first level. The states of $\mathcal{A}'$ are pairs $\langle S, O\rangle \in 2^Q \times 2^Q$, where S is used to store a whole level of the run tree of $\mathcal{A}$, and O to identify the checkpoints: $\mathcal{A}'$ keeps track in O of the states that "owe" a visit to α; i.e., states that have not visited α since the last checkpoint. The next checkpoint is reached when no state owes a visit.

Let $\mathcal{A} = \langle \Sigma, Q, q_{in}, \delta, \alpha\rangle$. Then, $\mathcal{A}' = \langle \Sigma, 2^Q \times 2^Q, \langle\{q_{in}\}, \emptyset\rangle, \delta', 2^Q \times \{\emptyset\}\rangle$, where δ' is defined, for all $\langle S, O\rangle \in 2^Q \times 2^Q$ and $\sigma \in \Sigma$, as follows.

- If $O \neq \emptyset$, then

$$\delta'(\langle S, O\rangle, \sigma) = \{\langle S', O' \setminus \alpha\rangle \mid S' \text{ satisfies } \bigwedge_{q\in S} \delta(q, \sigma), O' \subseteq S', \text{and } O' \text{ satisfies } \bigwedge_{q\in O} \delta(q, \sigma)\}.$$

- If $O = \emptyset$, then

$$\delta'(\langle S, O\rangle, \sigma) = \{\langle S', S' \setminus \alpha\rangle \mid S' \text{ satisfies } \bigwedge_{q\in S} \delta(q, \sigma)\}.$$

Note that all the states $\langle S, O\rangle$ in $\mathcal{A}'$ that are reachable from the initial state satisfy $O \subseteq S$. Accordingly, if the number of states in $\mathcal{A}$ is n, then the number of states in $\mathcal{A}'$ is at most 3^n. □

Note that the construction has the flavor of the subset construction [RS59], but in a dual interpretation: a set of states is interpreted conjunctively: the suffix of the word has to be accepted from all the states in S. While such a dual subset construction is sufficient for automata on finite words, the case of Büchi automata requires also the maintenance of a subset O of S, leading to a 3^n, rather than a 2^n, blow-up. As shown in [BKR10], this additional blow up can not be avoided.

3 Applications

In Section 2 we have seen two formalisms for specifying properties of nonterminating systems: LTL and automata. In this section we show that translate LTL formulas to alternating Büchi automata on infinite words with no blow up.

3.1 Translating LTL to alternating Büchi word automata

Consider an LTL formula ψ. For simplicity, we assume that ψ is in *positive normal form*, in which negation is applied only to atomic propositions. Formally, LTL formulas over AP in positive normal form have the following syntax:

- **true**, **false**, p, or $\neg p$, for $p \in AP$.
- ψ_1, $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $X\psi_1$, $\psi_1 U \psi_2$, or $\psi_1 R \psi_2$, where ψ_1 and ψ_2 are LTL formulas in positive normal form.

Note that restricting negation to atomic propositions requires to add not only the Boolean operator $\vee$ but also the temporal operator R. Still, it is easy to see that transforming an LTL formula into a formula in positive normal form involves no blow up.

The *closure* of an LTL formula ψ in positive normal form, denoted $cl(\psi)$, is the set of all its subformulas. Formally, $cl(\psi)$ is the smallest set of formulas that satisfy the following.

- $\psi \in cl(\psi)$.

- If $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $\psi_1 U \psi_2$ or $\psi_1 R \psi_2$ are in $cl(\psi)$, then $\psi_1 \in cl(\psi)$ and $\psi_2 \in cl(\psi)$.
- If $X\psi_1$ is in $cl(\psi)$, then $\psi_1 \in cl(\psi)$.

For example, $cl(p \wedge ((Xp)Uq))$ is $\{p \wedge ((Xp)Uq), p, (Xp)Uq, Xp, q\}$. It is easy to see that the size of $cl(\psi)$ is linear in the length of ψ. Accordingly, we define the size of ψ, denoted $|\psi|$, as $|cl(\psi)|$. Note that even though the number of elements in the closure of a formula can be logarithmic in the length of the formula if there are multiple occurrences of identical subformulas, our definition of size is legitimate since it corresponds to the number of nodes in a reduced DAG representation of the formula.

Theorem 3.1 [Var94] *For every LTL formula ψ, there is an alternating Büchi automaton $\mathcal{A}_\psi$ such that $L(\mathcal{A}_\psi) = L(\psi)$ and the number of states of $\mathcal{A}_\psi$ is linear in $|\psi|$.*

Proof: We assume that ψ is in positive normal form. We define $\mathcal{A}_\psi = \langle 2^{AP}, cl(\psi), \delta, Q_0, \alpha \rangle$, where

- for a state $\varphi \in cl(\psi)$ and letter $\sigma \in 2^{AP}$, the transition $\delta(\varphi, \sigma)$ is defined according to the form of φ as follows.
  - $\delta(\mathbf{true}, \sigma) = \mathbf{true}$ and $\delta(\mathbf{false}, \sigma) = \mathbf{false}$.
  - $\delta(p, \sigma) = \left[\begin{array}{ll} \mathbf{true} & \text{if } p \in \sigma \\ \mathbf{false} & \text{if } p \notin \sigma. \end{array} \right.$
  - $\delta(\neg p, \sigma) = \left[\begin{array}{ll} \mathbf{true} & \text{if } p \notin \sigma \\ \mathbf{false} & \text{if } p \in \sigma. \end{array} \right.$
  - $\delta(\varphi_1 \wedge \varphi_2, \sigma) = \delta(\varphi_1, \sigma) \wedge \delta(\varphi_2, \sigma)$.
  - $\delta(\varphi_1 \vee \varphi_2, \sigma) = \delta(\varphi_1, \sigma) \vee \delta(\varphi_2, \sigma)$.
  - $\delta(X\varphi, \sigma) = \varphi$.
  - $\delta(\varphi_1 U \varphi_2, \sigma) = \delta(\varphi_2, \sigma) \vee (\delta(\varphi_1, \sigma) \wedge \varphi_1 U \varphi_2)$.
  - $\delta(\varphi_1 R \varphi_2, \sigma) = \delta(\varphi_2, \sigma) \wedge (\delta(\varphi_1, \sigma) \vee \varphi_1 R \varphi_2)$.
- the set α of accepting states consists of all the formulas in $cl(\psi)$ of the form $\varphi_1 R \varphi_2$.

□

Example 3.2 We describe an alternating Büchi automaton for the LTL formula $\psi = p \wedge ((Xp)Ur))$. The alphabet of the automaton consists of the four letters in $2^{\{p,r\}}$. The states and transitions are described in the table below. No state is accepting.

state q	$\delta(q, \{p,r\})$	$\delta(q, \{p\})$	$\delta(q, \{r\})$	$\delta(q, \emptyset)$
ψ	**true**	$p \wedge ((Xp)Ur))$	**false**	**false**
p	**true**	**true**	**false**	**false**
$(Xp)Ur$	**true**	$p \wedge ((Xp)Ur))$	**true**	$p \wedge ((Xp)Ur))$

□

Example 3.3 We describe an alternating Büchi automaton for the LTL formula $\psi = GFp$. Note that $\psi = \mathbf{false} R (\mathbf{true} U p)$. In the example, we use the F and G abbreviations. The alphabet of the automaton consists of the two letters in $2^{\{p\}}$. The set of accepting states is $\{GFp\}$, and the states and transitions are described in the table below.

state q	$\delta(q, \{p\})$	$\delta(q, \emptyset)$
GFp	GFp	$GFp \wedge Fp$
Fp	**true**	Fp

□

Combining Theorems 3.1 and 2.3, we get the following.

Theorem 3.4 [VW94] *For every LTL formula ψ, there is a nondeterministic Büchi automaton $\mathcal{A}_\psi$ such that $L(\mathcal{A}_\psi) = L(\psi)$ and the number of states of $\mathcal{A}_\psi$ is exponential in $|\psi|$.*

As will be discussed in Section 3.8, the exponential blow-up in the translation cannot in general be avoided. Nevertheless, while the 3^n blow-up in Theorem 2.3 refers to general alternating Büchi automata, the automata obtained from LTL are of a special structure: all the cycles in the automata are self-loops of size 1. For such automata (termed *very-weak* alternating automata), alternation can be removed with only an $n2^n$ blow up [GO01, BKR10].

3.2 Automata-based satisfiability and model-checking procedures for LTL

Recall that the logic LTL is used for specifying properties of reactive systems. Two natural problems arise in this setting.

- *Satisfiability:* given an LTL formula ψ, is there a computation π such that $\pi \models \psi$?
- *Model Checking:* given a Kripke structure K and an LTL formula ψ, do all computations of K satisfy ψ?

In this section we describe the automata-theoretic approach to LTL satisfiability and model checking. We show how, using the construction described in Theorem 3.4, these problems can be reduced to problems about automata and their languages. We first describe the relevant problems from automata theory and their decision procedures.

- The *nonemptiness* problem is to decide, given an automaton $\mathcal{A}$, whether $L(\mathcal{A}) \neq \emptyset$.
- The *language-containment* problem is to decide, given automata $\mathcal{A}_1$ and $\mathcal{A}_2$, whether $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$.

Theorem 3.5 [VW94] *The nonemptiness problem for NBWs is decidable in linear time and is NLOGSPACE-complete.*

Proof: An NBW $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, \alpha \rangle$ induces a graph $G_\mathcal{A} = \langle Q, E_\delta \rangle$ in which $E_\delta(q, q')$ iff there is a letter $\sigma \in \Sigma$ such that $q' \in \delta(q, \sigma)$. We claim that $L(\mathcal{A})$ is nonempty if and only if there are states $q_0 \in Q_0$ and $q \in \alpha$ such that the graph $G_\mathcal{A}$ contains a path leading from q_0 to q and a circuit going though q. Suppose first that $L(\mathcal{A})$ is nonempty. Then there is an accepting run $r = q_0, q_1, \ldots$ of $\mathcal{A}$ on some input word, which corresponds to an infinite path of $G_\mathcal{A}$ that starts in Q_0. Since r is accepting, some $q \in \alpha$ occurs in r infinitely often; in particular, there are i, j, where $0 < i < j$, such that $q = q_i = q_j$. Thus, $q_0, \ldots, q_i$ corresponds to a path from Q_0 to q, and $q_i, \ldots, q_j$ to a circuit going through q.

Conversely, assume that $G_\mathcal{A}$ contains a path leading from q_0 to q and a circuit going though q. We can then construct an infinite path of $G_\mathcal{A}$ starting at q_0 and visiting q infinitely often. This path defines a run whose corresponding word is accepted by $\mathcal{A}$.

To check emptiness in linear time, we use a depth-first-search algorithm to construct a decomposition of $G_\mathcal{A}$ into maximal strongly connected components [CLR90]. It is easy to see that $\mathcal{A}$ is nonempty iff from a component that intersects Q_0 nontrivially it is possible to reach a nontrivial component that intersects α nontrivially. (A strongly connected component is nontrivial if it contains an edge, which means, since it is strongly connected, that it contains a cycle).

The algorithm proving membership in NLOGSPACE first guesses a state $q_0 \in Q_0$, then guesses a state q_1 that is directly connected to q_0, then guesses a state q_2 that is directly connected to q_1, etc., until it reaches a state $q \in \alpha$. At that point the algorithm remembers q and it continues to move nondeterministically from a state q_i to a state q_{i+1} that is directly connected to q until it reaches q again or i exceeds $|Q|$. Clearly, the algorithm needs only logarithmic memory, since it needs to remember (a description of) at most three states at each step and a counter that counts to $|Q|$.

NLOGSPACE-hardness follows from NLOGSPACE-hardness of graph reachability [Jon75]. □

The language-containment problem is PSPACE-complete [SVW87]. Membership in PSPACE follows from the fact $L(\mathcal{A}_1) \subseteq L(\mathcal{A}_2)$ iff $L(\mathcal{A}_1) \cap (\Sigma^\omega \setminus L(\mathcal{A}_2)) = \emptyset$. Thus, solving the problem has a flavor of complementation of $\mathcal{A}_2$, which involves an exponential blow-up. In the context of model checking, however, we would like to check the containment of the language of a Kripke structure in the language of an LTL formula. Since LTL is easy to complement, we can therefore trade the language-containment problem by the following easier problem.

- The *language-disjointness* problem is to decide, given automata $\mathcal{A}_1$ and $\mathcal{A}_2$, whether $L(\mathcal{A}_1) \cap L(\mathcal{A}_2) = \emptyset$. Since the intersection of nondeterministic Büchi automata can be defined on an augmented product construction [Cho74], the language-disjointness problem is decidable in quadratic time and is NLOGSPACE-complete.

Note that the construction of $\mathcal{A}_\psi$ can proceed *on-the-fly*. That is, given a state S of $\mathcal{A}_\psi$ and a letter $\sigma \in 2^{AP}$, it is possible to compute the set $\delta(S, \sigma)$ from the formulas in S. As we shall see below, this fact is very helpful, as it implies that, when reasoning about $\mathcal{A}_\psi$, one need not construct the whole state space of $\mathcal{A}_\psi$ up front, but can rather proceed in an on-demand fashion.

Theorem 3.6 [SC85] *The LTL satisfiability problem is PSPACE-complete.*

Proof: An LTL formula ψ is satisfiable iff the automaton $\mathcal{A}_\psi$ is not empty. Indeed, $\mathcal{A}_\psi$ accepts exactly all the computations that satisfy ψ. Recall that the nonemptiness problem for nondeterministic Büchi automata is in NLOGSPACE. Since $\mathcal{A}_\psi$ can be constructed on-the-fly, and its states can be encoded using linear space, membership in PSPACE follows. Hardness in PSPACE is proved in [SC85] by a generic reduction from a PSPACE Turing machine. □

Theorem 3.7 [SC85] *The LTL model-checking problem is PSPACE-complete.*

Proof: Consider a Kripke structure $K = \langle AP, W, R, W_0, \ell \rangle$. We can construct a nondeterministic Büchi automaton $\mathcal{A}_K$ that accepts a computation $\pi \in (2^{AP})^\omega$ iff π is a computation of K. The construction of $\mathcal{A}_K$ essentially moves the labels of K from the states to the transitions and makes all states accepting. Thus, $\mathcal{A}_K = \langle 2^{AP}, W, W_0, \delta, W \rangle$, where for all $w \in W$ and $\sigma \in 2^{AP}$, we have

$$\delta(w, \sigma) = \begin{cases} \{w' : R(w, w')\} & \text{if } \sigma = \ell(w). \\ \emptyset & \text{if } \sigma \neq \ell(w). \end{cases}$$

Now, K satisfies ψ iff all the computations of K satisfy ψ, and so iff $L(\mathcal{A}_K) \subseteq L(\mathcal{A}_\psi)$. As discussed above, we can use the fact that LTL formulas are easy to complement and check instead that no computation of K violates ψ. Accordingly, the model-checking problem can be reduced to the language-disjointness problem for A_K and $\mathcal{A}_{\neg\psi}$, where $\mathcal{A}_{\neg\psi}$ is the nondeterministic Büchi automaton for $\neg\psi$. Let $\mathcal{A}_{K,\neg\psi}$ be a nondeterministic Büchi automaton accepting the intersection of the languages of $\mathcal{A}_K$ and $\mathcal{A}_{\neg\psi}$. Since $\mathcal{A}_K$ has no acceptance condition, the construction of $\mathcal{A}_{K,\neg\psi}$ can proceed by simply taking the product of $\mathcal{A}_K$ with $\mathcal{A}_{\neg\psi}$. Then, K satisfies ψ iff $\mathcal{A}_{K,\neg\psi}$ is empty. By Theorem 3.4, the number of states of $\mathcal{A}_{\neg\psi}$ is exponential in $|\psi|$. Also, the size of $\mathcal{A}_K$ is linear in $|K|$. Thus, the size of $\mathcal{A}_{K,\neg\psi}$ is $|K| \cdot 2^{O(|\psi|)}$. Since

the construction of $\mathcal{A}_{\neg\psi}$, and hence also of $\mathcal{A}_{K,\neg\psi}$ can be done on-the-fly, and the states of $\mathcal{A}_{\neg\psi}$ can be encoded in linear space, membership in PSPACE follows from the membership in NLOGSPACE of the nonemptiness problem for nondeterministic Büchi automata. Hardness in PSPACE is proved in [SC85]. □

As described in the proof of Theorem 3.7, the PSPACE complexity of the LTL model-checking problem follows from the exponential size of the product automaton $\mathcal{A}_{K,\neg\psi}$. Note that $\mathcal{A}_{K,\neg\psi}$ is exponential only in $|\psi|$, and is linear in $|K|$. Nevertheless, as K is typically much bigger than ψ, and the exponential blow up of the translation of ψ to $\mathcal{A}_{\neg\psi}$ rarely appears in practice, it is the linear dependency in $|K|$, rather than the exponential dependency in $|\psi|$ that makes LTL model-checking so challenging in practice [LP85].

3.3 Interesting special cases

3.3.1 From LTL to deterministic Büchi automata

Nondeterministic Büchi automata are more expressive than deterministic Büchi automata; for example, the property FGp ("eventually always p") can be specified by a nondeterministic Büchi automaton and not by a deterministic one [Lan69]. In model checking, the standard algorithmic framework uses nondeterministic Büchi automata [VW86a]. Many applications, however, require deterministic automata, for example, synthesis [PR89], probabilistic verification [CY95], and runtime monitoring [TV10]. In general, determinization of Büchi automata requires using more general acceptance conditions, e.g., Rabin acceptance conditions [Rab69]. Furthermore, known determinization constructions are quite intricate [Saf88, Pit06, KW08, Sch09b] and involve an exponential blow-up, implying that constructing deterministic automata for LTL formula involves a doubly exponential blow-up. In this section we show that even the translation of LTL to deterministic Büchi automata, when possible, is tightly doubly exponential, though the intricacy of general determinization can be avoided.

Theorem 3.8 [KV05] *The translation of LTL to deterministic Büchi automata, when possible, is tightly doubly exponential.*

Proof: We start with the upper bound. We present both the traditional translation, which uses standard determinization [KW08, Pit06, Saf88, Sch09a]. as well as a recent one [BK09], which avoids it. First, we describe the traditional approach [KV05]. Let ψ be an LTL formula of length n and let $\mathcal{B}_\psi$ be a nondeterministic Büchi automaton that recognizes ψ. By Theorem 3.4, the automaton $\mathcal{B}_\psi$ has $2^{O(n)}$ states. By determinizing $\mathcal{B}_\psi$, we get a deterministic Rabin automaton $\mathcal{R}_\psi$ with $2^{2^{O(n)}}$ states [Saf88]. By [KPB94], if $\mathcal{R}_\psi$ can be translated to a deterministic Büchi automaton, it can be translated to a deterministic Büchi automaton with the same structure, that is, the same state space and transition function, and hence with $2^{2^{O(n)}}$ states.

We now describe a more recent approach [BK09]. This approach uses automata with *co-Büchi* acceptance condition. A run r of a co-Büchi automaton with an acceptance condition $\alpha \subseteq Q$ is accepting iff $inf(r) \cap \alpha = \emptyset$; that is, if the run visits the set α of accepting states only finitely many times. Let ψ be an LTL formula of length n and let $\mathcal{B}_{\neg\psi}$ be a nondeterministic Büchi automaton that recognizes $\neg\psi$. By Theorem 3.4, the automaton $\mathcal{B}_{\neg\psi}$ has $2^{O(n)}$ states. By the duality between the Büchi and the co-Büchi conditions, we know that ψ can be recognized by a deterministic Büchi automaton iff $\neg\psi$ can be recognized by a deterministic co-Büchi automaton. In particular, given a deterministic co-Büchi automaton for $\neg\psi$, viewing it as a Büchi automaton complements its language and hence results in a deterministic Büchi automaton for ψ. It follows that in order to generate a deterministic Büchi automaton for ψ we only have to translate $\mathcal{B}_{\neg\psi}$ to a deterministic co-Büchi automaton. By [BK09], such a translation can avoid the Rabin acceptance condition and involves only a simple variant of the subset construction, resulting in a deterministic co-Büchi automaton with at most $2^{2^{O(n)}}$ states.

For the lower bound, described in [KV05], consider the regular language

$$\mathcal{L}_n = \{\{0,1,\#\}^* \cdot \# \cdot w \cdot \# \cdot \{0,1,\#\}^* \cdot \$ \cdot w \cdot \#^\omega \ : w \in \{0,1\}^n\}.$$

A word τ is in $\mathcal{L}_n$ iff the word in $\{0,1\}^n$ that comes after the single \$ in τ appears somewhere before the \$ between #-s. By [CKS81], the smallest deterministic automaton on finite words that accepts $\mathcal{L}_n$ has at least 2^{2^n} states. (The proof in [CKS81] considers the language of the finite words obtained from $\mathcal{L}_n$ by omitting the $\#^\omega$ suffix. The proof, however, is independent of this technical detail: reaching the \$, the automaton should remember the possible set of words in $\{0,1\}^n$ that have appeared before.) We can specify $\mathcal{L}_n$ with an LTL formula ψ_n that makes sure that there is only one \$ in the word and that eventually there exists a position in which # is true and the i-th letter from this position, for $1 \leq i \leq n$, agrees with the i-th letter after the \$. Formally,

$$\begin{array}{c}\psi_n = [(\neg\$)U(\$ \wedge X((0 \vee 1) \wedge (X(0 \vee 1) \wedge \overset{n}{\cdots} X((0 \vee 1) \wedge XG\#)\cdots)))] \wedge \\ F[\# \wedge \bigwedge_{1 \leq i \leq n}((X^i 0 \wedge G(\$ \rightarrow X^i 0)) \vee (X^i 1 \wedge G(\$ \rightarrow X^i 1)))].\end{array}$$

Note that the length of ψ_n is quadratic in n. By encoding the positions of subwords starting with #, it is possible to tighten the result to a language in which the LTL formula is of linear length [KR10]. Also note that while $\mathcal{L}_n$ refers to $\{0,1,\#,\$\}$ as an alphabet, the LTL formula ψ_n refers to them as atomic propositions. It is easy, however, to add to ψ_n a conjunct that requires exactly one atomic proposition to hold in each position. Finally, note that the argument about the number of states of the smallest deterministic automaton that recognizes $\mathcal{L}_n$ is independent of the acceptance condition of the automaton. Thus, the doubly-exponential lower bound is valid also for acceptance conditions. □

Note that Theorem 3.8 also implies an exponential lower bound for the translation of LTL to nondeterministic Büchi automata.

3.3.2 Safety properties

Of special interest in formal verification are safety properties, which assert that the system always stays within some allowed region. Thus, if a computation violates a safety property, then it has a finite prefix after which the violation can be detected. Consider a language $L \subseteq \Sigma^\omega$ of infinite words over the alphabet Σ. A finite word $x \in \Sigma^*$ is a *bad prefix* for L iff for all $y \in \Sigma^\omega$, we have $x \cdot y \notin L$. Thus, a bad prefix is a finite word that cannot be extended to an infinite word in L. Note that if x is a bad prefix, then all the finite extensions of x are also bad prefixes. A language L is a *safety* language iff every $w \notin L$ has a finite bad prefix. For a safety language L, we denote by $pref(L)$ the set of all bad prefixes for L. For a Büchi automaton $\mathcal{A}$, we say that $\mathcal{A}$ is a safety automaton if $L(\mathcal{A})$ is a safety language, and we use $pref(\mathcal{A})$ to denote the set of its bad prefixes. Similarly, a safety LTL formula ψ is one for which $L(\psi)$ is safe, and we use $pref(\psi)$ to denote the set of its bad prefixes.

A safety LTL formula can be translated to a Büchi automaton all of whose states are accepting (*a.k.a.* a *looping* automaton) [Sis94]. For the purpose of model checking, it is sufficient to translate an LTL formula ψ to an automaton $\mathcal{A}_{\neg\psi}$ on finite words that accepts all its bad prefixes. Indeed, it is easy to see that a Kripke structure K satisfies ψ iff no prefix of a computation of K is accepted by $\mathcal{A}_{\neg\psi}$. Also, the latter check involves reasoning about finite words and is simpler than reasoning about Büchi automata. It turns out, however, that translating LTL formulas to automata for their bad prefixes involves a doubly-exponential blow-up:

Theorem 3.9 [KV01] *The translation of a safety LTL formula to a nondeterministic automaton for its bad prefixes is tightly doubly exponential.*

Proof: We start with the upper bound. We show that translating a safety nondeterministic Büchi automaton $\mathcal{A}$ to an automaton for its bad prefixes can be done with an exponential blow up. The

doubly-exponential bound then follows from the exponential translation of LTL to nondeterministic Büchi automata. Let $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, \alpha \rangle$. Recall that $\mathit{pref}(\mathcal{A})$ contains exactly all prefixes $x \in \Sigma^*$ such that for all $y \in \Sigma^\omega$, we have $x \cdot y \notin \mathcal{L}(\mathcal{A})$. Accordingly, the automaton for $\mathit{pref}(\mathcal{A})$ accepts a prefix x iff the set of states that $\mathcal{A}$ could be in after reading x contains only states whose language is empty. Formally, we define the (deterministic) automaton $\mathcal{A}' = \langle \Sigma, 2^Q, \delta', \{Q_0\}, \alpha' \rangle$, where δ' and α' are as follows.

- The transition function δ' follows the subset construction induced by δ; that is, for every $S \in 2^Q$ and $\sigma \in \Sigma$, we have $\delta'(S, \sigma) = \bigvee_{s \in S} \delta(s, \sigma)$.
- For a state s, we say that s is empty if $\mathcal{A}$ with initial state s is empty. For a set $S \in 2^Q$, we say that S is null if all the states in S are empty. Now, $\alpha' = \{S : S \text{ is null}\}$.

Note that while the construction is doubly exponential, it avoids the need to determinize $\mathcal{A}$ and only applies the subset construction to it.

We now turn to prove the lower bound. The difficulty in proving the lower bound is that while the automaton constructed in the upper bound is deterministic, no such requirement is made, and it may seem like the (allowed) use of nondeterminism could save an exponent. We show that this is not the case. We define, for $n \geq 1$, the language L'_n of infinite words over $\{0, 1, \#, \$\}$ that contain at least one \$, and after the first \$ either there is a word in $\{0,1\}^n$ that has appeared before, or there is no word in $\{0,1\}^n$ (that is, there is at least one # or \$ in the first n positions after the first \$). The complement of the language L'_n, denoted $\mathit{comp}(L'_n)$, is a safety language. Consider a prefix of a word of the form $x\$$ such that $x \in \{0, 1, \#\}^*$ contains all the words in $\{0,1\}^n$, separated by #. Note that such a prefix is a bad prefix for the complement of L'_n. Indeed, no matter how the word continues, the first n bits in the suffix either constitute a word in $\{0,1\}^n$, in which case it has appeared before, or they do not constitute a word in $\{0,1\}^n$. In both cases, the obtained word is in L'_n. Also, a nondeterministic automaton needs 2^{2^n} states to detect bad prefixes of this form. This makes the automaton for $\mathit{pref}(\mathit{comp}(L'_n))$ doubly exponential. On the other hand, we can specify L'_n with an LTL formula ψ_n that is quadratic in n. The formula is similar to the one for L_n from Theorem 3.8, only that it is satisfied also by computations in which the first \$ is not followed by a word in $\{0,1\}^n$. Now, the LTL formula $\neg\psi_n$ is a safety formula of size quadratic in n and the number of states of the smallest nondeterministic Büchi automaton for $\mathit{pref}(\psi)$ is 2^{2^n}. □

The discouraging blow up in Theorem 3.9 suggests one should release the requirement on $\mathit{pref}(\psi)$ and seek, instead, a nondeterministic automaton on finite words that need not accept all the bad prefixes, yet must accept at least one bad prefix of every infinite computation that does not satisfy ψ. Such an automaton is said to be *fine* for ψ. For example, an automaton $\mathcal{A}$ that accepts all the finite words in $0^* \cdot 1 \cdot (0 + 1)$ does not accept all the bad prefixes of the safety language $\{0^\omega\}$; in particular, it does not accept the minimal bad prefixes in $0^* \cdot 1$. Yet, $\mathcal{A}$ is fine for $\{0^\omega\}$. Indeed, every infinite word that is different from 0^ω has a prefix in $0^* \cdot 1 \cdot (0 + 1)$. In practice, almost all the benefit that one obtains from an automaton for $\mathit{pref}(\psi)$ can also be obtained from a fine automaton. It is shown in [KL06] that the translation of safety LTL formulas to finite automata involves only an exponential blow up. The key idea behind the construction is that even though it is impossible to bound the length of a bad prefix, it is possible to bound the number of bad "events" in a run on it.

4 Branching-time Logics

4.1 The logics CTL* and CTL

The logic *CTL** combines both branching-time and linear-time operators [EH86]. A path quantifier, either A ("for all paths") or E ("for some path"), can prefix an assertion composed of an arbitrary combination of the linear-time operators. A *positive normal form* CTL* formula is a CTL* formula 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 of quantifiers and temporal connectives. For technical convenience, we write CTL* formulas in positive normal form.

There are two types of formulas in CTL*: *state formulas*, interpreted on states, and *path formulas*, interpreted on paths. Formally, let AP be a set of atomic proposition names. A CTL* state formula is either:

- **true**, **false**, p, or $\neg p$, for all $p \in AP$;
- $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are CTL* state formulas;
- $A\psi$ or $E\psi$, where ψ is a CTL* path formula.

A CTL* path formula is either:

- A CTL* state formula;
- $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $X\psi_1$, $\psi_1 U \psi_2$, or $\psi_1 R \psi_2$, where ψ_1 and ψ_2 are CTL* path formulas.

CTL* is the set of state formulas generated by the above rules.

The logic *CTL* is a restricted subset of CTL* in which the temporal operators must be immediately preceded by a path quantifier. Formally, it is the subset of CTL* obtained by restricting the path formulas to be $X\varphi_1$, $\varphi_1 U \varphi_2$, or $\varphi_1 R \varphi_2$, where φ_1 and φ_2 are CTL state formulas. Note that LTL can also be viewed as a restricted subset of CTL*, where only one outermost path quantifier is allowed.

We say that a CTL formula φ is an *U-formula* if it is of the form $A\varphi_1 U \varphi_2$ or $E\varphi_1 U \varphi_2$. The subformula φ_2 is then called the *eventuality* of φ. Similarly, φ is a *R-formula* if it is of the form $A\varphi_1 R \varphi_2$ or $E\varphi_1 R \varphi_2$. The *closure* $cl(\varphi)$ of a CTL* (CTL) formula φ is the set of all CTL* (CTL) state subformulas of φ (including φ, but excluding **true** and **false**).

As in LTL, we define the size $|\varphi|$ of φ as the number of elements in $cl(\varphi)$. The semantics of CTL* is defined with respect to a *Kripke structure* $K = \langle AP, W, R, w^0, \ell \rangle$. Note that, for simplicity, we assume that the Kripke structure has a single initial state. The notation $K, w \models \varphi$ indicates that the state w of the Kripke structure K satisfies the CTL* state formula φ. Similarly, $K, \pi \models \psi$ indicates that the path π of the Kripke structure K satisfies the CTL* path formula ψ. When K is clear from the context, we write $w \models \varphi$ and $\pi \models \psi$. Also, $K \models \varphi$ if and only if $K, w^0 \models \varphi$.

The relation $\models$ is inductively defined as follows.

- For all w, we have $w \models$ **true** and $w \not\models$ **false**.
- $w \models p$ for $p \in AP$ iff $p \in \ell(w)$.
- $w \models \neg p$ for $p \in AP$ iff $p \notin \ell(w)$.
- $w \models \varphi_1 \wedge \varphi_2$ iff $w \models \varphi_1$ and $w \models \varphi_2$.
- $w \models \varphi_1 \vee \varphi_2$ iff $w \models \varphi_1$ or $w \models \varphi_2$.
- $w \models A\psi$ iff for every path $\pi = w_0, w_1, \ldots$, with $w_0 = w$, we have $\pi \models \psi$.
- $w \models E\psi$ iff there exists a path $\pi = w_0, w_1, \ldots$, with $w_0 = w$, such that $\pi \models \psi$.
- $\pi \models \varphi$ for a state formula φ, iff $w_0 \models \varphi$ where $\pi = w_0, w_1, \ldots$
- $\pi \models \psi_1 \wedge \psi_2$ iff $\pi \models \psi_1$ and $\pi \models \psi_2$.
- $\pi \models \psi_1 \vee \psi_2$ iff $\pi \models \psi_1$ or $\pi \models \psi_2$.

- $\pi \models X\psi$ iff $\pi^1 \models \psi$.
- $\pi \models \psi_1 U \psi_2$ iff there exists $i \geq 0$ such that $\pi^i \models \psi_2$ and for all $0 \leq j < i$, we have $\pi^j \models \psi_1$.
- $\pi \models \psi_1 R \psi_2$ iff for all $i \geq 0$ such that $\pi^i \not\models \psi_2$, there exists $0 \leq j < i$ such that $\pi^j \models \psi_1$.

For example, the CTL* formula $A(GFp \rightarrow GFq)$ states that in all computations, if the computation has infinitely many positions in which p holds, then it also has infinitely many positions in which q holds. The formula is not a CTL formula, and in fact has no equivalent CTL formula. The formula $AG(req \rightarrow EFgrant)$ states that in all computations, from every position in which a request is issued, there exists a path in in which a grant is eventually issued. This formula is in CTL.

In Section 2.2, we defined trees and labeled trees, and defined runs of alternating word automata as Q-labeled trees. In this section we define automata whose input are Σ-labeled trees. Such automata recognize languages like "the set of all $\{a, b\}$-labeled trees all of whose paths have infinitely many a's". Note that an infinite word in Σ^ω can be viewed as a Σ-labeled tree in which the degree of all nodes is 1.

Of special interest to us are Σ-labeled trees in which $\Sigma = 2^{AP}$ for some set AP of atomic propositions. We call such Σ-labeled trees *computation trees*. A Kripke structure $K = \langle AP, W, R, w^0, \ell \rangle$ can be viewed as a tree $\langle T_K, V_K \rangle$ that corresponds to the unwinding of K from w^0. Formally, for every node w, let $d(w)$ denote the degree of w (i.e., the number of successors of w, and note that for all w we have $d(w) \geq 1$), and let $succ_R(w) = \langle w_0, \ldots, w_{d(w)-1} \rangle$ be an ordered list of w's R-successors (we assume that the nodes of W are ordered). We define T_K and V_K inductively as follows:

1. $\varepsilon \in T_K$ and $V_K(\varepsilon) = w^0$.
2. For $y \in T_K$ with $succ_R(V_K(y)) = \langle w_0, \ldots, w_m \rangle$ and for $0 \leq i \leq m$, we have $y \cdot i \in T_K$ and $V_K(y \cdot i) = w_i$.

We sometimes view $\langle T_K, V_K \rangle$ as a computation tree over 2^{AP}, taking the label of a node to be $L(V_K(x))$ instead of $V_K(x)$. Which interpretation is intended will be clear from the context.

We sometimes refer to satisfaction of temporal logic formulas in computation trees, meaning their satisfaction in this Kripke structure. In particular, we use $L(\psi)$ to denote the set of 2^{AP}-labeled trees that satisfy ψ. Since satisfaction of CTL* formulas is preserved under unwinding, the identification of K with $\langle T_K, V_K \rangle$ is sound.

4.2 The propositional μ-calculus

The *propositional μ-calculus* is a propositional modal logic augmented with least and greatest fixed-point operators [Koz83]. Specifically, we consider a μ-calculus where formulas are constructed from Boolean propositions with Boolean connectives, the temporal operators EX and AX, as well as least (μ) and greatest (ν) fixed-point operators. We assume that μ-calculus formulas are written in positive normal form (negation only applied to atomic propositions constants and variables). Formally, given a set AP of atomic proposition constants and a set APV of atomic proposition variables, a μ-calculus formula is either:

- **true**, **false**, p or $\neg p$ for all $p \in AP$;
- y for all $y \in APV$;
- $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are μ-calculus formulas;
- $AX\varphi$ or $EX\varphi$, where φ is a μ-calculus formula;
- $\mu y.f(y)$ or $\nu y.f(y)$, where $y \in APV$ and $f(y)$ is a μ-calculus formula containing y.

We say that an atomic proposition variable $y \in APV$ is *free* in a formula ψ if y is not in the scope of a fixed-point operator. A *sentence* is a formula that contains no free atomic proposition variables. We call AX and EX *next modalities*, and we call μ and ν *fixed-point modalities*. We say that a μ-calculus formula is a *μ-formula* (*ν-formula*), if it is of the form $\mu y.f(y)$ ($\nu y.f(y)$). We use λ to denote a fixed-point modality μ or ν. For a λ-formula $\lambda y.f(y)$, the formula $f(\lambda y.f(y))$ is obtained from $f(y)$ by replacing each free occurrence of y with $\lambda y.f(y)$.

The closure, $cl(\varphi)$, of a μ-calculus sentence φ is the smallest set of μ-calculus sentences that satisfies the following:

- $\varphi \in cl(\varphi)$.
- If $\varphi_1 \wedge \varphi_2 \in cl(\varphi)$ or $\varphi_1 \vee \varphi_2 \in cl(\varphi)$, then $\varphi_1 \in cl(\varphi)$ and $\varphi_2 \in cl(\varphi)$.
- If $AX\varphi \in cl(\varphi)$ or $EX\varphi \in cl(\varphi)$, then $\varphi \in cl(\varphi)$.
- If $\mu y.f(y) \in cl(\varphi)$, then $f(\mu y.f(y)) \in cl(\varphi)$.
- If $\nu y.f(y) \in cl(\varphi)$, then $f(\nu y.f(y)) \in cl(\varphi)$.

For example, for $\varphi = \mu y.(q \vee (p \wedge EXy))$, $cl(\varphi) = \{\varphi, q \vee (p \wedge EX\varphi), q, p \wedge EX\varphi, p, EX\varphi\}$. It follows from a result of [Koz83] that for every μ-calculus formula φ, the number of elements in $cl(\varphi)$ is linear with respect to a reduced DAG representation of φ. Accordingly, as with CTL*, we define the size $|\varphi|$ of φ as the number of elements in $cl(\varphi)$.

Given a Kripke structure $K = \langle AP, W, R, w^0, \ell \rangle$, and a set $\{y_1, \ldots, y_n\}$ of atomic proposition variables, a *valuation* $\mathcal{V} : \{y_1, \ldots, y_n\} \to 2^W$ is an assignment of subsets of W to the variables $\{y_1, \ldots, y_n\}$. For a valuation $\mathcal{V}$, a variable y, and a set $W' \subseteq W$, we denote by $\mathcal{V}[y \leftarrow W']$ the valuation that assigns W' to y and otherwise coincides with $\mathcal{V}$. A formula φ with free variables $\{y_1, \ldots, y_n\}$ is interpreted over the structure K as a mapping φ^K from valuations to 2^W. Thus, $\varphi^K(\mathcal{V})$ denotes the set of states that satisfy φ with the valuation $\mathcal{V}$. The mapping φ^K is defined inductively as follows:

- $\mathbf{true}^K(\mathcal{V}) = W$ and $\mathbf{false}^K(\mathcal{V}) = \emptyset$;
- For $p \in AP$, we have $p^K(\mathcal{V}) = \{w \in W : p \in \ell(w)\}$ and $(\neg p)^K(\mathcal{V}) = \{w \in W : p \notin \ell(w)\}$;
- For $y_i \in APV$, we have $y_i^K(\mathcal{V}) = \mathcal{V}(y_i)$;
- $(\varphi_1 \wedge \varphi_2)^K(\mathcal{V}) = \varphi_1^K(\mathcal{V}) \cap \varphi_2^K(\mathcal{V})$;
- $(\varphi_1 \vee \varphi_2)^K(\mathcal{V}) = \varphi_1^K(\mathcal{V}) \cup \varphi_2^K(\mathcal{V})$;
- $(AX\varphi)^K(\mathcal{V}) = \{w \in W : \forall w' \text{ such that } \langle w, w' \rangle \in R, \text{ we have } w' \in \varphi^K(\mathcal{V})\}$;
- $(EX\varphi)^K(\mathcal{V}) = \{w \in W : \exists w' \text{ such that } \langle w, w' \rangle \in R \text{ and } w' \in \varphi^K(\mathcal{V})\}$;
- $(\mu y.f(y))^K(\mathcal{V}) = \bigcap\{W' \subseteq W : f^K(\mathcal{V}[y \leftarrow W']) \subseteq W'\}$;
- $(\nu y.f(y))^K(\mathcal{V}) = \bigcup\{W' \subseteq W : W' \subseteq f^K(\mathcal{V}[y \leftarrow W'])\}$.

Note that no valuation is required for a sentence. Thus, a sentence is interpreted over K as a predicate defining a subset of W. For a state $w \in W$ and a sentence φ, we say that $w \models \varphi$ iff $w \in \varphi^K$. For example, the μ-calculus formula $\mu y.(q \vee (p \wedge EXy))$ is equivalent to the CTL formula $EpUq$. Finally, as with CTL*, we use the fact that satisfaction of μ-calculus formulas is preserved under unwinding, and interpret them with respect to both Kripke structures and their computation trees. In particular, we use $L(\varphi)$, for a sentence φ, to denote the language of computation trees that correspond to Kripke structures whose initial state satisfies φ.

A μ-calculus formula is *alternation free* if, for all $y \in APV$, there are no occurrences of ν (μ) on any syntactic path from an occurrence of μy (νy) to an occurrence of y. For example, the formula $\mu x.(p \vee \mu y.(x \vee EXy))$ is alternation free and the formula $\nu x.\mu y.((p \wedge x) \vee EXy)$ is not alternation free. The *alternation-free μ-calculus* is a subset of μ-calculus containing only alternation-free formulas.

4.3 Symmetric alternating tree automata

Automata over infinite trees (tree automata) run over labeled trees that have no leaves [Tho90]. *Alternating automata* generalize nondeterministic tree automata and were first introduced in [MS87] (see [Slu85] for alternating automata on finite trees). We define here *symmetric* alternating tree automata [JW95, Wil99]. Let $\Omega = \{\varepsilon, \Box, \Diamond\}$. A *symmetric alternating tree automaton* is an automaton in which the transition function δ maps a state q and a letter σ to a formula in $\mathcal{B}^+(\Omega \times Q)$. Atoms of the form $\langle \varepsilon, q\rangle$ are called *ε-transitions*. Intuitively, an atom $\langle \varepsilon, q\rangle$ corresponds to a copy of the automaton in state q sent to the current node of the input tree. An atom $\langle \Box, q\rangle$ corresponds to copies of the automaton in state q sent to all the successors of the current node. An atom $\langle \Diamond, q\rangle$ corresponds to a copy of the automaton in state q, sent to some successor of the current node.

When, for instance, the automaton is in state q, reads a node x with successors $x \cdot \upsilon_1, \ldots, x \cdot \upsilon_n$, and

$$\delta(q, V(x)) = ((\Box, q_1) \wedge (\varepsilon, q_2)) \vee ((\Diamond, q_2) \wedge (\Diamond, q_3)),$$

it can either send n copies in state q_1 to the nodes $x \cdot \upsilon_1, \ldots, x \cdot \upsilon_n$ and send a copy in state q_2 to x, or send one copy in state q_2 to some node in $x \cdot \upsilon_1, \ldots, x \cdot \upsilon_n$ and send one copy in state q_3 to some node in $x \cdot \upsilon_1, \ldots, x \cdot \upsilon_n$. Thus, symmetric automata can send several copies to the same successor, and can also have ε-transitions. On the other hand, symmetric automata cannot distinguish between left and right and can send copies to successor nodes only in either a universal or an existential manner.

Formally, a symmetric automaton is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha\rangle$ where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+(\Omega \times Q)$ is a transition function, $q_0 \in Q$ is an initial state, and α specifies the acceptance condition (a condition that defines a subset of Q^ω). The automaton $\mathcal{A}$ is *ε-free* iff δ contains no ε-transitions. Let Υ be a finite set of directions. In Section 2.2 we defined Σ-labeled Υ-trees. For simplicity, we assume that automata run on *full* Σ-labeled Υ-trees $\langle T, V\rangle$, thus $T = \Upsilon^*$. A *run* of a symmetric automaton $\mathcal{A}$ on an input Σ-labeled Υ-tree $\langle T, V\rangle$ is a $(\Upsilon^* \times Q)$-labeled $\mathbb{N}$-tree $\langle T_r, r\rangle$. Unlike T, in which each node has exactly $|\Upsilon|$ children, the branching degree of the nodes in T_r may vary, and T_r may also have leaves (nodes with no children). Thus, $T_r \subset \mathbb{N}^*$ and a path in T_r may be either finite, in which case it ends in a leaf, or infinite. Each node of T_r corresponds to a node of T. A node in T_r, labeled by (x, q), describes a copy of the automaton that reads the node x of T and visits the state q. Note that many nodes of T_r can correspond to the same node of T; in contrast, in a run of a nondeterministic automaton on $\langle T, V\rangle$ there is a one-to-one correspondence between the nodes of the run and the nodes of the tree. The labels of a node and its children have to satisfy the transition function. Formally, the run $\langle T_r, r\rangle$ is an $(\Upsilon^* \times Q)$-labeled tree that satisfies the following:

1. $\varepsilon \in T_r$ and $r(\varepsilon) = (\varepsilon, q_0)$.
2. Let $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \theta$. Then there is a (possibly empty) set $S \subseteq \Omega \times Q$, such that S satisfies θ, and for all $(c, s) \in S$, the following hold:
   - If $c = \varepsilon$, then there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x, s)$.
   - If $c = \Box$, then for each $\upsilon \in \Upsilon$, there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot \upsilon, s)$.
   - If $c = \Diamond$, then for some $\upsilon \in \Upsilon$, there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot \upsilon, s)$.

For example, if $\langle T, V\rangle$ is a $\{1, 2\}$-tree with $V(\varepsilon) = a$ and $\delta(q_0, a) = \Diamond q_1 \wedge \Box q_2$, then level 1 of $\langle T_r, r\rangle$ includes a node labeled $(1, q_1)$ or $(2, q_1)$, and include two nodes labeled $(1, q_2)$ and $(2, q_2)$. Note that if $\theta = \mathbf{true}$, then y need not have children. This is the reason why T_r may have leaves. Also, since there exists no set S as required for $\theta = \mathbf{false}$, we cannot have a run that takes a transition with $\theta = \mathbf{false}$.

Each infinite path ρ in $\langle T_r, r\rangle$ is labeled by a word in Q^ω. Let $inf(\rho)$ denote the set of states in Q that appear in $r(\rho)$ infinitely often. A run $\langle T_r, r\rangle$ is accepting iff all its infinite paths satisfy the acceptance condition (in particular, if T_r is finite, then $\langle T_r, r\rangle$ is accepting). In *Büchi* automata, $\alpha \subseteq Q$, and an

infinite path ρ satisfies α iff $inf(\rho) \cap \alpha \neq \emptyset$. In *parity* automata, α is a partition $\{F_1, F_2, \ldots, F_k\}$ of Q and an infinite path ρ satisfies α iff the minimal index i for which $inf(\rho) \cap F_i \neq \emptyset$ is even.

An automaton accepts a tree iff there exists an accepting run on it. We denote by $\mathcal{L}(\mathcal{A})$ the language of the automaton $\mathcal{A}$; i.e., the set of all labeled trees that $\mathcal{A}$ accepts. We say that $\mathcal{A}$ is *nonempty* iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$. We denote by $\mathcal{A}^q$ the automaton obtained from $\mathcal{A}$ by making q the initial state.

Example 4.1 Consider the language of all $\{a, b\}$-labeled trees in which all paths have infinitely many nodes labeled a and there exists a path in which all nodes at odd positions in the path are labeled b. A symmetric alternating Büchi automaton for the language is $\mathcal{A} = \langle \{a, b\}, \{q_0, q_1, q_2, q_3, q_4\}, q_0, \{q_1, q_3, q_4\} \rangle$, where δ is described in the following table.

state q	$\delta(q, a)$	$\delta(q, b)$
q_0	$(\Box, q_1) \wedge (\Diamond, q_3)$	$(\Box, q_2) \wedge (\Diamond, q_3)$
q_1	$(\Box, q_1)$	$(\Box, q_2)$
q_2	$(\Box, q_1)$	$(\Box, q_2)$
q_3	**false**	$(\Diamond, q_4)$
q_4	$(\Diamond, q_3)$	$(\Diamond, q_3)$

In the states q_1 and q_2, the automaton checks that all paths have infinitely many nodes labeled by a. It does so by going, in all directions, to q_1 whenever it reads an a, and to q_2 whenever it reads a b. The acceptance condition than requires infinitely many visits in q_1. In the states q_3 and q_4, the automaton checks that there exists a path in which all nodes at odd positions are labeled b. It does so by alternating, in some direction, between these two states, and allowing state q_3, which is visited along the guessed path in exactly all odd positions, to read b only. Finally, in state q_0, the automaton sends copies that check both requirements. □

In [MSS86], Muller et al. introduce *weak alternating automata* (WAAs). In a WAA, we have a Büchi acceptance condition $\alpha \subseteq Q$ and there exists a partition of Q into disjoint sets, $Q_1, \ldots, Q_m$, such that for each set Q_i, either $Q_i \subseteq \alpha$, in which case Q_i is an *accepting set*, or $Q_i \cap \alpha = \emptyset$, in which case Q_i is a *rejecting set*. In addition, there exists a partial order $\leq$ on the collection of the Q_i's such that $Q_j \leq Q_i$ if q occurs in $\delta(q, \sigma)$ for some $q \in Q_i$, $q' \in Q_j$, and $\sigma \in \Sigma$. Thus, transitions from a state in Q_i lead to states in either the same Q_i or a lower one. It follows that every infinite path of a run of a WAA ultimately gets "trapped" within some Q_i. The path then satisfies the acceptance condition if and only if Q_i is an accepting set. Indeed, a run visits infinitely many states in α if and only if it gets trapped in an accepting set. We sometimes refer to the *type* of an automaton, meaning its acceptance condition, and its being weak or not weak.

We call the partition of Q into sets the *weakness partition* and we call the partial order over the sets of the weakness partition the *weakness order*. Often (in particular, in all the cases we consider in this work) a WAA is given together with its weakness partition and order. Otherwise, as we claim below, these can be induced by the partition of the graph of the WAA into *maximal strongly connected components* (MSCCs). Formally, given $\mathcal{A}$, let $G_\mathcal{A}$ be a directed graph induced by $\mathcal{A}$; that is, the vertices of $G_\mathcal{A}$ are the states of $\mathcal{A}$ and there is an edge from vertex q to vertex q' iff there is a transition in $\mathcal{A}$ from the state q that involves the state q'. Let $C_1, \ldots, C_n$ be a partition of $G_\mathcal{A}$ into MSCCs. That is, for every C_i and for every two vertices q and q' in C_i, there is a path from q to q' and from q' to q, and for every vertex $q'' \notin C_i$, the set $C_i \cup \{q''\}$ no longer satisfies this condition. Since the partition into MSCCs is maximal, there is a partial order $\leq$ between them so that $C_i \leq C_j$ iff C_i is reachable from C_j.

A *hesitant alternating automaton* (HAA) is an alternating automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$, where $\alpha = \langle G, B \rangle$ with $G \subseteq Q$ and $B \subseteq Q$, and the following condition on the structure of the transitions hold. As in WAAs, there exists a partition of Q into disjoint sets and a partial order $\leq$ such that transitions from a state in Q_i lead to states in either the same Q_i or a lower one. In addition, each set Q_i is classified as either *transient*, *existential*, or *universal*, and for each set Q_i and for all $q \in Q_i$ and $\sigma \in \Sigma$, the following hold:

1. If Q_i is a transient set, then $\delta(q, \sigma)$ contains no elements of Q_i.
2. If Q_i is an existential set, then $\delta(q, \sigma)$ only contains disjunctively related elements of Q_i.
3. If Q_i is a universal set, then $\delta(q, \sigma)$ only contains conjunctively related elements of Q_i.

It follows that every infinite path π of a run r gets trapped within some existential or universal set Q_i. The path then satisfies an acceptance condition $\langle G, B \rangle$ if and only if either Q_i is an existential set and $inf(\pi) \cap G \neq \emptyset$, or Q_i is a universal set and $inf(\pi) \cap B = \emptyset$. Note that the acceptance condition of HAAs combines the Büchi and the co-Büchi acceptance conditions: existential sets refer to a Büchi condition G whereas universal sets refer to a co-Büchi condition B. Note also that while the transition function of HAAs is more restricted than the one of WAAs, their acceptance condition is more expressive. We will need the stronger acceptance condition to handle CTL* formulas. We call the partition of Q into sets the *hesitation partition* and we call the partial order over the sets the *hesitation order*. The length of the longest descending chain in the hesitation order is defined as the *depth* of the HAA.

4.4 Nonemptiness and 1-letter nonemptiness for alternating tree automata

In this section we study both the nonemptiness and the 1-letter nonemptiness problems for alternating tree automata (that it, the nonemptiness problem for automata with $|\Sigma| = 1$). As we shall see in the sequel, the differences between these problems are analogous to the differences between the satisfiability and the model-checking problems for branching temporal logics.

For nondeterministic automata, the solution to the nonemptiness problem can ignore the alphabet of the automaton. For example, a nondeterministic automaton on finite words is not empty iff there is a path from an initial state to a final state. For alternating automata, the alphabet cannot be ignored as the nonemptiness algorithm has to make sure that different copies of the automaton follow the same input. For example, if one copy is sent in order to check that the input word has a suffix 0^ω and a second copy is sent in order to check that the input word has a suffix 1^ω, then the language of the automaton is empty, but ignoring the alphabet would cause the nonemptiness algorithm to check the nonemptiness of each copy independently, leading to a wrong answer.

We first consider the general nonemptiness problem. Its solution involves alternation removal – a translation of the automaton into an equivalent nondeterministic tree automaton. For weak automata, the translation is similar to the one described for word automata in Section 2.3. For parity and hesitant automata, the translation is more complicated and is described in [MS95]. Another issue that has to be addressed in the context of the nonemptiness problem is the *sufficient branching degree* property for branching temporal logics and tree automata. According to this property, if a branching-temporal logic formula is satisfiable, then there is a Kripke structure whose branching degree is bounded (and depends on the size of the formula) that satisfies it. A similar property holds for symmetric alternating tree automata. Here, we assume that the nonemptiness problem gets as input both the automaton and a branching degree. Applying the nonemptiness algorithm in order to check the satisfiability of branching temporal logics, we will then use the known branching degree property for the corresponding logics. Finally, for nondeterministic automata and a fixed branching degree, the non-emptiness problem is equivalent to the problem of deciding two-player games in which the winning condition corresponds to the acceptance condition of the automaton [GH82].

Theorem 4.2 [MSS86, MS95, FL79, Sei90] *Consider a symmetric alternating tree automaton $\mathcal{A}$ (weak, hesitant, or parity) and a branching degree d given in unary. The problem of deciding whether $\mathcal{A}$ accepts some tree of degree d is EXPTIME-complete.*

We now turn to study the 1-letter nonemptiness problem. Note that the 1-letter nonemptiness problem is really a membership problem, asking whether the tree all of whose nodes are labeled by the single letter is accepted by the alternating automaton. In fact, once we move to the 1-letter setting, the "directions"

$\Box, \Diamond$, and ε coincide, and one can check instead the 1-letter nonemptiness of a word automaton. We will discuss this point further in Section 5.2. As with nondeterministic tree automata, the nonemptiness problem for alternating word automata over a singleton alphabet is equivalent to the problem of deciding two-player games.

Theorem 4.3 [KVW00] *Consider an alternating word automaton $\mathcal{A}$ over a singleton alphabet. The problem of deciding whether $\mathcal{A}$ is not empty:*

- *can be solved in linear time for weak or hesitant automata.*
- *is in NP $\cap$ co-NP for parity automata.*
- *can be solved in space $O(m \log^2 n)$ for hesitant automata of size n and depth m.*

Proof: We describe the linear-time algorithm for weak automata. The other algorithms can be found in [KVW00]. Consider a WAA $\mathcal{A} = \langle \{a\}, Q, \delta, q_0, \alpha \rangle$. The algorithm labels the states of $\mathcal{A}$ with either 'T', standing for "not empty", or 'F', standing for "empty". The language of $\mathcal{A}$ is thus nonempty if and only if the initial state q_0 is labeled with 'T'.

As $\mathcal{A}$ is weak, there exists a partition of Q into disjoint sets Q_i such that there exists a partial order $\leq$ on the collection of the Q_i's and such that for every $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, a)$, we have that $Q_j \leq Q_i$. Thus, transitions from a state in Q_i lead to states in either the same Q_i or a lower one. In addition, each set Q_i is classified as accepting, if $Q_i \subseteq \alpha$, or rejecting, if $Q_i \cap \alpha = \emptyset$. Note that if the partition of Q is not given, one can find such a partition in linear time. The algorithm works in phases and proceeds up the partial order. We regard **true** and **false** as states with a self loop. The state **true** constitutes an accepting set and the state **false** constitutes a rejecting set, both minimal in the partial order. Let $Q_1 \leq \ldots \leq Q_n$ be an extension of the partial order to a total order. In each phase i, the algorithm handles states from the minimal set Q_i that still has not been labeled.

States that belong to the set Q_1 are labeled according to the classification of Q_1. Thus, they are labeled with 'T' if Q_1 is an accepting set and they are labeled with 'F' if it is a rejecting set. Once a state $q \in Q_i$ is labeled with 'T' or 'F', transition functions in which q occurs are simplified accordingly; i.e., a conjunction with a conjunct 'F' is simplified to 'F' and a disjunction with a disjunct 'T' is simplified to 'T'. Consequently, a transition function $\delta(q', \sigma)$ for some q' (not necessarily from Q_i) can be simplified to **true** or **false**. The state q' is then labeled, and simplification propagates further.

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

Correct operation of the algorithm can be understood as follows. As $\mathcal{A}$ is weak, it is guaranteed that once the automaton visits a state that belongs to Q_1, it visits only states from Q_1 thereafter. Similarly, when the automaton visits a state q whose labeling cannot be decided according to labeling of states in lower sets, this state leads to a cycle or belongs to a cycle of states of the same status. Hence the labeling of states according to the classification of the set to which they belong.

Formally, we prove that for all $1 \leq i \leq n$, all the states in Q_i are labeled correctly. The proof proceeds by induction on i. The case $i = 1$ is immediate. Assume that we have already labeled correctly all the states in all Q_j with $j < i$ and let $q \in Q_i$. We consider the case where Q_i is an accepting set. The proof is symmetric for the case where Q_i is a rejecting set. We distinguish between three possibilities of labeling q:

1. The state q is labeled 'T' before the phase i. Then, the value of $\delta(q, a)$, simplified according to the labeling already done, is **true**. Therefore, there exists a run of $\mathcal{A}^q$ in which every copy created in

the first step (i.e., every copy that is created in order to satisfy $\delta(q,a)$) reaches a state q' for which, by the induction hypothesis, the language of $\mathcal{A}^{q'}$ is not empty. Hence, the language of $\mathcal{A}^q$ is also not empty.

2. The state q is labeled 'F' before the phase i. The correctness proof is symmetric to the one of the previous case: The value of $\delta(q,a)$, simplified according to the labeling already done, is **false**. Therefore, every run of $\mathcal{A}^q$ has at least one copy created in the first step and reaches a state q' for which, by the induction hypothesis, the language of $\mathcal{A}^{q'}$ is empty. Hence, the language of $\mathcal{A}^q$ is also empty.

3. The state q is labeled 'T' during the phase i. Then, it must be the case that the simplification of $\delta(q,a)$ contains states of Q_i. Moreover, it contains only states of Q_i and they all have not been labeled before the phase i. Thus, there exists a run of $\mathcal{A}^q$ in which every copy created in the first step either reaches a state q' for which the language of $\mathcal{A}^{q'}$ is not empty, or stays forever in Q_i. Hence, the language of $\mathcal{A}^q$ is not empty.

Note that these are indeed the only possibilities of labeling q: a state in an accepting set Q_i cannot be labeled after the phase i and it cannot be labeled with 'F' during the phase i since we are dealing with an accepting Q_i.

As suggested in [BB79, Bee80, DG84], the algorithm can be implemented in linear running time using a data structure that corresponds to an and/or graph. □

5 Applications

5.1 Translating branching temporal logics to alternating tree automata

In this section we present translations of CTL and the alternation-free μ-calculus to alternating weak tree automata, of μ-calculus formulas to alternating parity automata, and of CTL* formulas to alternating hesitant automata.

Theorem 5.1 [KVW00] *For every CTL formula ψ, there is a symmetric alternating weak tree automaton $\mathcal{A}_\psi$ such that $L(\mathcal{A}_\psi) = L(\psi)$ and the number of states of $\mathcal{A}_\psi$ is linear in $|\psi|$.*

Proof: We define $\mathcal{A}_\psi = \langle 2^{AP}, cl(\psi), \delta, \psi, \alpha\rangle$, where the set α of accepting states consists of all the R-formulas in $cl(\psi)$, and the transition function δ is defined, for all $\sigma \in 2^{AP}$, as follows.

- $\delta(\mathbf{true}, \sigma) = \mathbf{true}$ and $\delta(\mathbf{false}, \sigma) = \mathbf{false}$.
- $\delta(p,\sigma) = \begin{cases} \mathbf{true} & \text{if } p \in \sigma \\ \mathbf{false} & \text{if } p \notin \sigma. \end{cases}$
- $\delta(\neg p,\sigma) = \begin{cases} \mathbf{true} & \text{if } p \notin \sigma \\ \mathbf{false} & \text{if } p \in \sigma. \end{cases}$
- $\delta(\varphi_1 \wedge \varphi_2, \sigma) = \delta(\varphi_1,\sigma) \wedge \delta(\varphi_2,\sigma)$.
- $\delta(\varphi_1 \vee \varphi_2, \sigma) = \delta(\varphi_1,\sigma) \vee \delta(\varphi_2,\sigma)$.
- $\delta(AX\varphi,\sigma) = (\Box,\varphi)$.
- $\delta(EX\varphi,\sigma) = (\Diamond,\varphi)$.
- $\delta(A\varphi_1 U \varphi_2,\sigma) = \delta(\varphi_2,\sigma) \vee (\delta(\varphi_1,\sigma) \wedge (\Box, A\varphi_1 U \varphi_2))$.

- $\delta(E\varphi_1 U \varphi_2, \sigma) = \delta(\varphi_2, \sigma) \vee (\delta(\varphi_1, \sigma) \wedge (\Diamond, E\varphi_1 U \varphi_2))$.
- $\delta(A\varphi_1 R \varphi_2, \sigma) = \delta(\varphi_2, \sigma) \wedge (\delta(\varphi_1, \sigma) \vee (\Box, A\varphi_1 R \varphi_2))$.
- $\delta(E\varphi_1 R \varphi_2, \sigma) = \delta(\varphi_2, \sigma) \wedge (\delta(\varphi_1, \sigma) \vee (\Diamond, E\varphi_1 R \varphi_2))$.

The weakness partition and order of $\mathcal{A}_\psi$ are defined as follows. 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 α or disjoint from α. □

Example 5.2 Consider the CTL formula $\psi = A(\mathbf{true}\, U(A(\mathbf{false}\, R\, p)))$. Note that $\psi = AFAGp$. The WAA associated with ψ is $\mathcal{A}_\psi = \langle\{\{p\}, \emptyset\}, \{\psi, A(\mathbf{false}\, R\, p)\}, \delta, \psi, \{A(\mathbf{false}\, R\, p)\}\rangle$, where δ is described in the following table (we restrict $\mathcal{A}_\psi$ to the reachable states; in particular, the state p is not reachable from the state ψ).

state q	$\delta(q, \{p\})$	$\delta(q, \emptyset)$
ψ	$(\Box, A(\mathbf{false}\, R\, p)) \vee (\Box, \psi)$	$(\Box, \psi)$
$A\mathbf{false}Rp$	$(\Box, A(\mathbf{false}\, R\, p))$	**false**

In the state ψ, if p holds in the present, then $\mathcal{A}_\psi$ may either guess that $A(\mathbf{false}\, R\, p)$, the eventuality of ψ, is satisfied in the present, or proceed with $(\Box, \psi)$, which means that the requirement for fulfilling the eventuality of ψ is postponed to the future. The crucial point is that since $\psi \notin \alpha$, infinite postponing is impossible. In the state $A(\mathbf{false}\, R\, p)$, $\mathcal{A}_\psi$ expects a tree in which p is always true in all paths. Then, it keeps visiting $A(\mathbf{false}\, R\, p)$ forever. Since $A(\mathbf{false}\, R\, p) \in \alpha$, this is permitted. □

Example 5.3 Consider the CTL formula $\psi = A((EX\neg p)Ur)$. The WAA associated with ψ is $\mathcal{A}_\psi = \langle 2^{\{p,r\}}, \{\psi, \neg p\}, \delta, \psi, \emptyset\rangle$, where δ is described in the following table (we restrict $\mathcal{A}_\psi$ to its reachable states).

state q	$\delta(q, \{p, r\})$	$\delta(q, \{p\})$	$\delta(q, \{r\})$	$\delta(q, \emptyset)$
ψ	**true**	$(\Diamond, \neg p) \wedge (\Box, \psi)$	**true**	$(\Diamond, \neg p) \wedge (\Box, \psi)$
$\neg p$	**false**	**false**	**true**	**true**

In the state ψ, if b does not hold on the present, then $\mathcal{A}_\psi$ requires both $EX\neg p$ to be satisfied in the present (that is, $\neg p$ to be satisfied in some successor), and ψ to be satisfied by all the successors. As $\psi \notin \alpha$, the WAA $\mathcal{A}_\psi$ should eventually reach a node that satisfies r. □

We now present a similar translation for the alternation-free μ-calculus. The WAA we construct use ε-transitions. As we explain below when we handle μ-calculus formulas, ε-transitions enable us to decompose a formula into its subformulas before we read the next letter in the input. In the case of μ-calculus formulas, this is essential for a sound definition of the acceptance condition of the automaton. In the case of alternation-free μ-calculus, this guarantees that the transitions from μ and ν formulas are well defined. We will elaborate on this point in Remark 5.6.

Theorem 5.4 [KVW00] *For every alternation-free μ-calculus formula ψ, there is a symmetric alternating weak tree automaton $\mathcal{A}_\psi$ such that $L(\mathcal{A}_\psi) = L(\psi)$ and the number of states of $\mathcal{A}_\psi$ is linear in $|\psi|$.*

Proof: We define $\mathcal{A}_\psi = \langle 2^{AP}, cl(\psi), \delta, \psi, \alpha\rangle$. For atomic proposition constants and for formulas of the forms $AX\varphi$ or $EX\varphi$, the transition function δ is equal to the one described for CTL. For formulas of the form $\varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2$, as well as for μ and ν formulas, we use ε-transitions and define, for all $\sigma \in 2^{AP}$, the transition function as follows. Note that while the transitions are independent of σ, the fact they are ε-transitions imply that σ is eventually read.

- $\delta(\varphi_1 \wedge \varphi_2, \sigma) = (\varepsilon, \varphi_1) \wedge (\varepsilon, \varphi_2)$.
- $\delta(\varphi_1 \vee \varphi_2, \sigma) = (\varepsilon, \varphi_1) \vee (\varepsilon, \varphi_2)$.
- $\delta(\mu y.f(y), \sigma) = (\varepsilon, f(\mu y.f(y)))$.
- $\delta(\nu y.f(y), \sigma) = (\varepsilon, f(\nu y.f(y)))$.

In order to define α, we introduce an equivalence relation $\mathcal{R}$ over $cl(\varphi)$ given by

$$\varphi_1 \mathcal{R} \varphi_2 \text{ iff } \varphi_1 \in cl(\varphi_2) \text{ and } \varphi_2 \in cl(\varphi_1).$$

Since ψ is alternation free, it is guaranteed that an equivalence class of $\mathcal{R}$ cannot contain both a ν-formula and a μ-formula. A state $\varphi \in cl(\psi)$ belongs to α if and only if it belongs to an equivalence class that contains a ν-formula.

The weakness partition and order of $\mathcal{A}_\psi$ are induced by $\mathcal{R}$ as follows. Each equivalence class of $\mathcal{R}$ constitutes a set Q_i. The partial order is defined by $Q_1 \leq Q_2$ if $\varphi_1 \in cl(\varphi_2)$ for some $\varphi_1 \in Q_1$ and $\varphi_2 \in Q_2$. As in CTL, 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 α or disjoint from α. □

Example 5.5 Consider the formula $\psi = \mu y.(p \vee EXAXy)$. The WAA associated with ψ is $\mathcal{A}_\psi = \langle\{\{p\}, \emptyset\}, \{\psi, p \vee EXAX\psi, p, EXAX\psi, AX\psi\}, \delta, \psi, \emptyset\rangle$, where δ is described below.

state q	$\delta(q, \{p\})$	$\delta(q, \emptyset)$
ψ	$(\varepsilon, p \vee EXAX\psi)$	$(\varepsilon, p \vee EXAX\psi)$
$p \vee EXAX\psi$	$(\varepsilon, p) \vee (\varepsilon, EXAX\psi)$	$(\varepsilon, p) \vee (\varepsilon, EXAX\psi)$
p	**true**	**false**
$EXAX\psi$	$(\Diamond, AX\psi)$	$(\Diamond, AX\psi)$
$AX\psi$	$(\Box, \psi)$	$(\Box, \psi)$

In the state ψ, we take an ε-transition and decompose ψ to a disjunction. In the decomposed state $p \vee EXAX\psi$, we again take ε-transitions and decompose it further to the state p and the state $EXAX\psi$ Since the state set of $\mathcal{A}_\psi$ constitutes a single rejecting set, the proposition p should eventually hold. □

Remark 5.6 Recall that WAAs for CTL formulas do not have ε-transitions. A natural definition of a transition function that does not use ε-transitions in WAAs for the alternation-free μ-calculus would have the same transitions as in the case of CTL for states associated with Boolean assertions, and the following transitions for states associated with μ and ν formulas.

- $\delta(\mu y.f(y), \sigma) = \delta(f(\mu y.f(y)), \sigma)$.
- $\delta(\nu y.f(y), \sigma) = \delta(f(\nu y.f(y)), \sigma)$.

For example, giving up ε-transitions in the WAA for the formula $\psi = \mu y.(p \vee EXAXy)$ from Example 5.5, we would have obtained the two-state WAA $\mathcal{A}_\psi = \langle\{\{p\}, \emptyset\}, \{\psi, AX\psi\}, \delta, \psi, \emptyset\rangle$, where δ is described below (we restrict $\mathcal{A}_\psi$ to its reachable states).

state q	$\delta(q, \{p\})$	$\delta(q, \emptyset)$
ψ	**true**	$(\Diamond, AX\psi)$
$AX\psi$	$(\Box, \psi)$	$(\Box, \psi)$

While this simplifies the automaton, a transition function without ε-tranistions may result in a circular definition. For example, if $\psi = \mu y.(p \vee y)$, then $\delta(\psi, \sigma) = \delta(p \vee \psi, \sigma) = \delta(p, \sigma) \vee \delta(\psi, \sigma)$. A μ-calculus formula is *guarded* if for all $y \in APV$, all the occurrences of y that are in a scope of a fixed-point modality λ are also in a scope of a next modality which is itself in the scope of λ. Thus, a μ-calculus sentence is *guarded* if for all $y \in APV$, all the occurrences of y are in the scope of a next modality. For example, the formula $\mu y.(p \vee EXy)$ is guarded and the formula $EX\mu y.(p \vee y)$ is not guarded. The above circularity in the definition occurs only with formulas are not guarded. This, for guarded formulas, one can simplify the state space of the WAA and proceed without ε-transitions. In [KVW00], the translation to AWWs assumes that the formulas are guarded, and no ε-transitions are used. Since a translation to guarded normal form may involve a blow-up, it is sometimes better to use a translation that involves ε-transitions. □

We now continue to full μ-calculus formulas. Here, unlike the case of CTL or guarded alternation-free μ-calculus formulas, the translation must include ε-transitions. The problem with a transition function that does not use ε-tranistion is that it does not allow the necessary acceptance conditions to be defined. Indeed, if one looks carefully at the transition relation obtained for a CTL or a guarded alternation-free μ-calculus formula ψ (see Remark 5.6), not all elements of $cl(\psi)$ are reachable from the initial state. Specifically, a Boolean combination can be reachable without its constituents being reachable. This makes it impossible to express an acceptance condition involving a formula appearing only as a constituent of a Boolean state. In the case of CTL this was of no consequence since the acceptance condition involves exclusively R formulas and these do by construction appear as states. Similarly, for the alternation-free μ calculus, the problem was worked around by using the absence of alternation to define equivalence classes of formulas in such a way that each class contains at least one formula that is a state of the automaton. For the full μ-calculus, such short cuts are not possible and we thus need to decompose states that are Boolean combinations of formulas of the form $p, \neg p, AX\varphi, EX\varphi, \mu y.f(y)$, or $\nu y.f(y)$ to states associated with formulas of these forms. In order to achieve this, we need both the richer parity condition and ε-transitions. Formally, we have the following result.

Theorem 5.7 [EJ91, BC96b, KVW00] *For every μ-calculus formula ψ, there is a symmetric alternating parity tree automaton $\mathcal{A}_\psi$ such that $L(\mathcal{A}_\psi) = L(\psi)$ and the number of states of $\mathcal{A}_\psi$ is linear in $|\psi|$.*

Proof: For a μ-calculus sentence ψ and a subformula $\varphi = \lambda y.f(y)$ of ψ, we define the *alternation level* of φ in ψ, denoted $al_\psi(\varphi)$, as follows [BC96a].

- If φ is a sentence, then $al_\psi(\varphi) = 1$.
- Otherwise, let $\xi = \lambda' x.g(x)$ be the innermost μ or ν subformula of ψ that has φ as a strict subformula. Then, if x is free in φ and $\lambda' \neq \lambda$, we have $al_\psi(\varphi) = al_\psi(\xi) + 1$. Otherwise, we have $al_\psi(\varphi) = al_\psi(\xi)$.

Intuitively, the alternation level of φ in ψ is the number of alternating fixed-point operators we have to "wrap φ with" in order to reach a sub-sentence of ψ. For example, the alternation level of the subformula $\mu y.((p \vee AXy) \wedge AXz)$ in the formula $\nu z.\mu y.((p \vee AXy) \wedge AXz)$ is 2.

Now, given ψ, we define the parity automaton $\mathcal{A}_\psi = \langle 2^{AP}, cl(\psi), \delta, \psi, \alpha \rangle$, where

- The transition function δ is exactly as in the automata for the alternation-free μ-calculus. We describe it here for completeness.
  - $\delta(\mathbf{true}, \sigma) = \mathbf{true}$ and $\delta(\mathbf{false}, \sigma) = \mathbf{false}$.
  - $\delta(p, \sigma) = \begin{cases} \mathbf{true} & \text{if } p \in \sigma \\ \mathbf{false} & \text{if } p \notin \sigma. \end{cases}$

- $\delta(\neg p, \sigma) = \begin{cases} \textbf{true} & \text{if } p \notin \sigma \\ \textbf{false} & \text{if } p \in \sigma. \end{cases}$
- $\delta(\varphi_1 \wedge \varphi_2, \sigma) = (\varepsilon, \varphi_1) \wedge (\varepsilon, \varphi_2)$.
- $\delta(\varphi_1 \vee \varphi_2, \sigma) = (\varepsilon, \varphi_1) \vee (\varepsilon, \varphi_2)$.
- $\delta(AX\varphi, \sigma) = (\Box, \varphi)$.
- $\delta(EX\varphi, \sigma) = (\Diamond, \varphi)$.
- $\delta(\mu y.f(y), \sigma) = (\varepsilon, f(\mu y.f(y)))$.
- $\delta(\nu y.f(y), \sigma) = (\varepsilon, f(\nu y.f(y)))$.

- Let d be the maximal alternation level of subformulas of ψ. Denote by G_i the set of all the ν-formulas in $cl(\psi)$ of alternation level i in ψ. Denote by B_i the set of all μ-formulas in $cl(\psi)$ whose alternation level in ψ is less than or equal to i. The parity acceptance condition forces each path in the run of $\mathcal{A}_\psi$ to visit some ν-formula infinitely often and visit μ-formulas of smaller alternation levels only finitely often. Formally, let $F_0 = \emptyset$, and for every $1 \leq i \leq d$, let $F_{2i-1} = B_i$ and $F_{2i} = G_i$.

□

In Theorem 5.1 we presented a translation of CTL formulas to WAAs. Since the size of each set in the weakness partition is 1, it is easy to see that the resulting WAAs have the restricted structure of HAAs, and that we can redefine their acceptance condition to be $\langle G, B\rangle$, where G is the set of all ER-formulas in $cl(\psi)$ and B is the set of all AU-formulas in $cl(\psi)$. Also, since each set in the HAA corresponds to a single formula in $cl(\psi)$, the depth of the HAA is at most $|\psi|$.

We now present a translation of CTL* formulas to HAAs. Weak alternating automata define exactly the set of weakly definable languages [Rab70, MSS86]. The logic CTL* can define languages that are not weakly definable. For example, the set of trees that satisfy the CTL* formula $AFGp$ is not weakly definable [Rab70]. Therefore, a stronger acceptance condition is required for automata corresponding to formulas of CTL*, which is why hesitant alternating automata need a combination of Büchi and co-Büchi conditions.

For two HAAs $\mathcal{A}_1$ and $\mathcal{A}_2$ over the same alphabet Σ, we say that $\mathcal{A}_1$ is the *complement* of $\mathcal{A}_2$ iff $\mathcal{L}(\mathcal{A}_1)$ includes exactly all the Σ-labeled trees that are not in $\mathcal{L}(\mathcal{A}_2)$.

Given a transition function δ, let $\tilde{\delta}$ denote the dual function of δ. That is, for every q and σ with $\delta(q, \sigma) = \theta$, let $\tilde{\delta}(q, \sigma) = \tilde{\theta}$, where $\tilde{\theta}$ is obtained from θ by swapping $\vee$ and $\wedge$, switching $\Box$ and $\Diamond$, and switching **true** and **false**. If, for example, $\theta = (\Box, p) \vee (\textbf{true} \wedge (\Diamond, q))$ then $\tilde{\theta} = (\Diamond, p) \wedge (\textbf{false} \vee (\Box, q))$. Since $\mathcal{A}$ has a single initial state, dualizing the transition function and the acceptance condition amounts to complementing the automaton. Formally, we have the following result.

Lemma 5.8 [MS87, KVW00] *Given an HAA $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \langle G, B\rangle\rangle$, the alternating automaton $\tilde{\mathcal{A}} = \langle \Sigma, Q, \tilde{\delta}, q_0, \langle B, G\rangle\rangle$ is an HAA that complements $\mathcal{A}$.*

For an HAA $\mathcal{A}$, we say that $\tilde{\mathcal{A}}$ is the *dual* HAA of $\mathcal{A}$.

Theorem 5.9 *For every CTL* formula ψ, there is a symmetric alternating hesitant tree automaton $\mathcal{A}_\psi$ such that $L(\mathcal{A}_\psi) = L(\psi)$, the number of states of $\mathcal{A}_\psi$ is exponential in $|\psi|$, and its depth is linear in $|\psi|$.*

Proof: Before defining $\mathcal{A}_\psi$ we need the following definitions and notations. For two CTL* formulas θ and φ, we say that θ is *maximal* in φ, if and only if θ is a strict state subformula of φ and there exists no state formula "between them", namely, there exists no strict subformula ξ of φ such that θ is a strict subformula of ξ. We denote by $max(\varphi)$ the set of all formulas maximal in φ. For example, $max(A((Xp)U(EXq))) = \{p, EXq\}$.

We construct $\mathcal{A}_\psi$ by induction on the structure of ψ. With each formula $\varphi \in cl(\psi)$, we associate an HAA $\mathcal{A}_\varphi$ composed from HAAs associated with formulas maximal in φ. We assume that the state sets of composed HAAs are disjoint (otherwise, we rename states) and that for all the HAAs we have $\Sigma = 2^{AP}$ (that is, an HAA associated with a subformula that does not involve all of AP is extended in a straightforward way). For φ with $max(\varphi) = \{\varphi_1, \ldots, \varphi_n\}$ and for all $1 \leq i \leq n$, let $\mathcal{A}_{\varphi_i} = \langle \Sigma, Q^i, \delta^i, q_0^i, \langle G^i, B^i \rangle \rangle$ be the HAA associated with φ_i and let $\tilde{\mathcal{A}}_{\varphi_i} = \langle \Sigma, \tilde{Q^i}, \tilde{\delta^i}, \tilde{q_0^i}, \langle \tilde{G^i}, \tilde{B^i} \rangle \rangle$ be its dual HAA. We define $\mathcal{A}_\varphi$ as follows.

- If $\varphi = p$ or $\varphi = \neg p$ for some $p \in AP$, then $\mathcal{A}_\varphi$ is a one-state HAA.
- If $\varphi = \varphi_1 \wedge \varphi_2$, then $\mathcal{A}_\varphi = \langle \Sigma, Q^1 \cup Q^2 \cup \{q_0\}, \delta, q_0, \langle G^1 \cup G^2, B^1 \cup B^2 \rangle \rangle$, where q_0 is a new state and δ is defined as follows. For states in Q^1 and Q^2, the transition function δ agrees with δ^1 and δ^2. For the state q_0 and for all $\sigma \in \Sigma$, we have $\delta(q_0, \sigma) = \delta(q_0^1, \sigma) \wedge \delta(q_0^2, \sigma)$. Thus, in the state q_0, $\mathcal{A}_\varphi$ sends all the copies sent by both $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$. The singleton $\{q_0\}$ constitutes a transient set, with the ordering $\{q_0\} > Q_i$ for all the sets Q_i in Q^1 and Q^2.

  The construction for $\varphi = \varphi_1 \vee \varphi_2$ is similar, with $\delta(q_0, \sigma) = \delta(q_0^1, \sigma) \vee \delta(q_0^2, \sigma)$.
- If $\varphi = E\xi$, where ξ is a CTL* path formula, we first build an HAA $\mathcal{A}'_\varphi$ over the alphabet $\Sigma' = 2^{max(\varphi)}$. That is, $\mathcal{A}'_\varphi$ regards the formulas maximal in φ as atomic propositions. Let $\mathcal{U}_\xi = \langle \Sigma', Q, \eta, q_0, \alpha \rangle$ be a nondeterministic Büchi automaton on infinite words such that $\mathcal{U}_\xi$ accepts exactly all the word models of ξ (see Theorem 3.4) where the maximal subformulas are regarded as atomic propositions. Then, $\mathcal{A}'_\varphi = \langle \Sigma', Q, \delta', q_0, \langle \alpha, \emptyset \rangle \rangle$ extends $\mathcal{U}_\xi$ to trees by simulating it along a single branch. That is, for all $q \in Q$ and $\sigma' \in \Sigma'$, we have

  $$\delta'(q, \sigma') = \bigvee_{q' \in \eta(q, \sigma')} (\Diamond, q').$$

  If $\eta(q, \sigma') = \emptyset$, then $\delta'(q, \sigma') = \mathbf{false}$. Note that Q constitutes a single existential set. The HAA $\mathcal{A}'_\varphi$ accepts exactly all the Σ'-labeled tree models of φ.

  We now adjust $\mathcal{A}'_\varphi$ to the alphabet Σ. The resulting automaton is $\mathcal{A}_\varphi$. Intuitively, $\mathcal{A}_\varphi$ starts additional copies of the HAAs associated with formulas in $max(\varphi)$. These copies guarantee that whenever $\mathcal{A}'_\varphi$ assumes that a formula in $max(\varphi)$ holds, then it indeed holds, and that whenever $\mathcal{A}'_\varphi$ assumes that a formula does not hold, then the negation of the formula holds. Formally, $\mathcal{A}_\varphi = \langle \Sigma, Q \cup \bigcup_i (Q^i \cup \tilde{Q^i}), \delta, q_0, \langle \alpha \cup \bigcup_i (G^i \cup \tilde{G^i}), \bigcup_i (B^i \cup \tilde{B^i}) \rangle \rangle$, where δ is defined as follows. For states in $\bigcup_i (Q^i \cup \tilde{Q^i})$, the transition function δ agrees with the corresponding δ^i and $\tilde{\delta^i}$. For $q \in Q$ and for all $\sigma \in \Sigma$, we have

  $$\delta(q, \sigma) = \bigvee_{\sigma' \in \Sigma'} (\delta'(q, \sigma') \wedge (\bigwedge_{\varphi_i \in \sigma'} \delta^i(q_0^i, \sigma)) \wedge (\bigwedge_{\varphi_i \notin \sigma'} \tilde{\delta^i}(\tilde{q_0^i}, \sigma))).$$

  Each conjunction in δ corresponds to a label $\sigma' \in \Sigma'$. Some copies of $\mathcal{A}_\varphi$ (these originated from $\delta'(q, \sigma')$) proceed as $\mathcal{A}'_\varphi$ when it reads σ'. Other copies guarantee that σ' indeed holds in the current node. The set Q constitutes an existential set, with the ordering $Q > Q'$ for all the sets Q' in $\bigcup_i (\{Q^i\} \cup \{\tilde{Q^i}\})$.
- If $\varphi = A\xi$, we construct and dualize the HAA of $E\neg\xi$.

We prove the correctness of the construction by induction on the structure of φ. The proof is immediate for the case φ is of the form p, $\neg p$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, or $A\xi$. We consider here the case where $\varphi = E\xi$. If a tree $\langle T_K, V_K \rangle$ satisfies φ, then there exists a path π in it such that $\pi \models \xi$. Thus, there exists an accepting run r of $\mathcal{U}_\xi$ on a word that agrees with π on the formulas in $max(\varphi)$. It is easy to see that a run of $\mathcal{A}_\varphi$ that proceeds on π according to r can accept $\langle T_K, V_K \rangle$. Indeed, by the definition of $\mathcal{A}'_\varphi$, the

copies that proceed according to δ' satisfy the acceptance condition. In addition, by the adjustment of $\mathcal{A}'_\varphi$ to the alphabet 2^{AP} and by the induction hypothesis, copies that take care of the maximal formulas can fulfill the acceptance condition. Now, if a run r of $\mathcal{A}_\varphi$ accepts a tree $\langle T_K, V_K\rangle$, then there must be a path π in this tree such that $\mathcal{A}_\varphi$ proceeds according to an accepting run of $\mathcal{U}_\xi$ on a word that agrees with π on the formulas in $max(\varphi)$. Thus, $\pi \models \xi$ and $\langle T_K, V_K\rangle$ satisfies φ.

We now consider the size of $\mathcal{A}_\psi$. For every φ, we prove, by induction on the structure of φ, that the size of $\mathcal{A}_\varphi$ is exponential in $|\varphi|$.

- Clearly, for $\varphi = p$ or $\varphi = \neg p$ for some $p \in AP$, the size of $\mathcal{A}_\varphi$ is constant.
- For $\varphi = \varphi_1 \wedge \varphi_2$ or $\varphi = \varphi_1 \vee \varphi_2$, we have $|\mathcal{A}_\varphi| = O(|\mathcal{A}_{\varphi_1}| + |\mathcal{A}_{\varphi_2}|)$. By the induction hypothesis, $|\mathcal{A}_{\varphi_1}|$ is exponential in $|\varphi_1|$ and $|\mathcal{A}_{\varphi_2}|$ is exponential in $|\varphi_2|$. Thus, $|\mathcal{A}_\varphi|$ is surely exponential in $|\varphi|$.
- For $\varphi = E\xi$, we know, by Theorem 3.4, that the number of states of the word automaton $\mathcal{U}_\xi$ is exponential in $|\xi|$. Therefore, $\mathcal{A}'_\varphi$ is exponential in $|\varphi|$. Also, $|\Sigma'|$ is exponential in $|max(\varphi)|$ and, by the induction hypothesis, for all $\varphi_i \in max(\varphi)$, the size of $\mathcal{A}_{\varphi_i}$ is exponential in $|\varphi_i|$. Therefore, $\mathcal{A}_\varphi$ is also exponential in $|\varphi|$.
- For $\varphi = A\xi$, we know, by the above, that $|\mathcal{A}_{E\neg\xi}|$ is exponential in $|\varphi|$. Since complementing an HAA does not change its size, the result for φ follows.

Finally, since each subformula of ψ induces exactly one set, the depth of $\mathcal{A}_\psi$ is linear in $|\psi|$. □

Example 5.10 Consider the CTL* formula $\psi = AGF(p \vee AXp)$. We describe the construction of $\mathcal{A}_\psi$ step by step. Since ψ is of the form $A\xi$, we need to construct and dualize the HAA of $EFG((\neg p) \wedge EX\neg p)$. We start with the HAAs $\mathcal{A}_\varphi$ and $\tilde{\mathcal{A}}_\varphi$ for $\varphi = (\neg p) \wedge EX\neg p$.

$\mathcal{A}_\varphi = \langle\{\{p\},\emptyset\}, \{q_2, q_3\}, \delta, q_2, \langle\emptyset,\emptyset\rangle\rangle$, with

state q	$\delta(q,\{p\})$	$\delta(q,\emptyset)$
q_2	**false**	$(\Diamond, q_3)$
q_3	**false**	**true**

$\tilde{\mathcal{A}}_\varphi = \langle\{\{p\},\emptyset\}, \{\tilde{q_2}, \tilde{q_3}\}, \tilde{\delta}, \tilde{q_2}, \langle\emptyset,\emptyset\rangle\rangle$, with

state q	$\tilde{\delta}(q,\{p\})$	$\tilde{\delta}(q,\emptyset)$
$\tilde{q_2}$	**true**	$(\Box, \tilde{q_3})$
$\tilde{q_3}$	**true**	**false**

Starting with a Büchi word automaton $\mathcal{U}_\xi$ for $\xi = FG\varphi$, we construct $\mathcal{A}'_{E\xi}$.

$\mathcal{U}_\xi = \langle\{\{\varphi\},\emptyset\}, \{q_0, q_1\}, \eta, q_0, \{q_1\}\rangle$, with

- $\eta(q_0,\{\varphi\}) = \{q_0, q_1\}$
- $\eta(q_0,\emptyset) = \{q_0\}$
- $\eta(q_1,\{\varphi\}) = \{q_1\}$
- $\eta(q_1,\emptyset) = \emptyset$

Hence, $\mathcal{A}'_{E\xi} = \langle\{\{\varphi\},\emptyset\}, \{q_0, q_1\}, \delta', q_0, \langle\{q_1\},\emptyset\rangle\rangle$, with

state q	$\delta'(q,\{\varphi\})$	$\delta'(q,\emptyset)$
q_0	$(\Diamond, q_0) \vee (\Diamond, q_1))$	$(\Diamond, q_0)$
q_1	$(\Diamond, q_1)$	**false**

We are now ready to compose the automata into an automaton $\mathcal{A}_{E\xi}$ over the alphabet $\{\{p\},\emptyset\}$.

We define $\mathcal{A}_{E\xi} = \langle\{\{p\},\emptyset\}, \{q_0, q_1, q_3, \tilde{q_3}\}, \delta, q_0, \langle\{q_1\},\emptyset\rangle\rangle$, with (note that we simplify the transitions, replacing **true** $\wedge\, \theta$ or **false** $\vee\, \theta$ by θ, replacing **true** $\vee\, \theta$ by **true**, and replacing **false** $\wedge\, \theta$ by **false**).

state q	$\delta(q,\{p\})$	$\delta(q,\emptyset)$
q_0	$(\Diamond, q_0)$	$[((\Diamond, q_0) \vee (\Diamond, q_1)) \wedge (\Diamond, q_3)] \vee [(\Diamond, q_0) \wedge (\Box, \tilde{q_3})]$
q_1	**false**	$(\Diamond, q_1) \wedge (\Diamond, q_3)$
q_3	**false**	**true**
$\tilde{q_3}$	**true**	**false**

Consider $\delta(q_0, \emptyset)$. The first disjunct corresponds to the case where $\mathcal{A}'_{E\xi}$ guesses that φ holds in the present. Then, $\mathcal{A}_{E\xi}$ proceeds with $\delta'(q_0, \{\varphi\}) = (\Diamond, q_0) \vee (\Diamond, q_1)$ conjuncted with $\delta(q_2, \emptyset) = (\Diamond, q_3)$. The later guarantees that φ indeed holds in the present. The second disjunct corresponds to the case where φ does not hold in the present. Then, $\mathcal{A}_{E\xi}$ proceeds with $\delta'(q_0, \emptyset) = (\Diamond, q_0)$ conjuncted with $\delta(\tilde{q_2}, \emptyset) = (\Box, \tilde{q_3})$.

We obtain $\mathcal{A}_\psi$ by dualizing $\mathcal{A}_{E\xi}$. Hence, $\mathcal{A}_\psi = \langle\{\{p\}, \emptyset\}, \{\tilde{q_0}, \tilde{q_1}, \tilde{q_3}, q_3\}, \tilde{\delta}, \tilde{q_0}, \langle\emptyset, \{\tilde{q_1}\}\rangle\rangle$, with

state q	$\tilde{\delta}(q,\{p\})$	$\tilde{\delta}(q,\emptyset)$
$\tilde{q_0}$	$(\Box, \tilde{q_0})$	$((\Box, \tilde{q_0}) \wedge (\Box, \tilde{q_1})) \vee (\Box, \tilde{q_3})) \wedge ((\Box, \tilde{q_0}) \vee (\Diamond, q_3))$
$\tilde{q_1}$	**true**	$(\Box, \tilde{q_1}) \vee (\Box, \tilde{q_3})$
$\tilde{q_3}$	**true**	**false**
q_3	**false**	**true**

Consider the state $\tilde{q_1}$. A copy of $\mathcal{A}_\psi$ that visits $\tilde{q_1}$ keeps creating new copies of $\mathcal{A}_\psi$, all visiting $\tilde{q_1}$, unless it reaches a node that satisfies p or AXp. Since $\tilde{q_1} \in B$, all the copies should eventually reach such a node. So, by sending a copy that visits the state $\tilde{q_1}$ to a node x, the HAA $\mathcal{A}_\psi$ guarantees that all the paths in the subtree with x as root eventually reach a node satisfying $p \vee AXp$. Hence, in the state $\tilde{q_0}$, unless $\mathcal{A}_\psi$ gets convinced that $p \vee AXp$ holds in the present, it sends copies that visit $\tilde{q_1}$ to all the successors. In addition, it always send copies visiting $\tilde{q_0}$ to all the successors. □

5.2 Automata-based satisfiability and model-checking procedures for branching temporal logics

In Section 3.2 we have seen how the satisfiability and model-checking problems for LTL can be reduced to the nonemptiness and the language-disjointness problems for nondeterministic Büchi word automata. In this section we reduce the satisfiability and model-checking problems for branching-time temporal logics to the nonemptiness and 1-letter nonemptiness problems for alternating tree automata. The type of the automaton depends on the logic considered.

We start with the satisfiability problem.

Theorem 5.11 *The satisfiability problem is*

- *EXPTIME-complete for CTL, alternation-free μ-calculus, and μ-calculus.*
- *2EXPTIME-complete for CTL*.*

Proof: By [ES84], all the logics considered in the theorem satisfy the linear sufficient branching degree property. More specifically, is ψ is satisfiable, then there is Kripke structure of branching degree $|\psi| + 1$ that satisfies ψ. The upper bounds then follow from the translations described in Section 5.1 and the complexities of the nonemptiness problem specified in Theorem 4.2, applied to the corresponding symmetric alternating automata and branching degree. The lower bounds are proven in [FL79, SV89]. □

We continue with the model-checking problem. Given the alternating tree automaton $\mathcal{A}_\psi$ and a Kripke structure K, we define their product $\mathcal{A}_{K,\psi}$ as a 1-letter alternating word automaton (see below). Thus, the product with K takes us from a tree automaton to a word automaton, and from an automaton over an alphabet 2^{AP} to a 1-letter automaton. Obviously, the nonemptiness problem for tree automata can not, in general, be reduced to the nonemptiness problem of word automata. Also, as discussed above, the nonemptiness problem for alternating word automata cannot, in general, be reduced to the 1-letter nonemptiness problem. It is taking the product with K that makes both reductions valid here. Since each state in $\mathcal{A}_{K,\psi}$ is associated with a state w of K, then each state has the exact information as to which subtree of $\langle T_K, V_K\rangle$ it is responsible for (i.e., which subtree it would have run over if $\mathcal{A}_{K,\psi}$ had not been a 1-letter word automaton). The branching structure of $\langle T_K, V_K\rangle$ and its 2^{AP}-labeling are thus embodied in the states of $\mathcal{A}_{K,\psi}$. In particular, it is guaranteed that all the copies of the product automaton that start in a certain state, say one associated with w, follow the same labeling: the one that corresponds to computations of K that start in w.

Let $\mathcal{A}_\psi = \langle 2^{AP}, Q_\psi, \delta_\psi, q_0, \alpha_\psi\rangle$ be a symmetric alternating tree automaton for $L(\psi)$ and let $K = \langle AP, W, R, w^0, \ell\rangle$ be a Kripke structure. The product automaton of $\mathcal{A}_\psi$ and K is an alternating word automaton $\mathcal{A}_{K,\psi} = \langle\{a\}, W \times Q_\psi, \delta, \langle w^0, q_0\rangle, \alpha\rangle$ where δ and α are defined as follows:

- Let $q \in Q_\psi$, $w \in W$, and $\delta_\psi(q, \ell(w)) = \theta$. Then $\delta(\langle w, q\rangle, a) = \theta'$, where θ' is obtained from θ by replacing:
  - each atom (ε, q') in θ by the atom $\langle w, q'\rangle$,
  - each atom $(\Box, q')$ in θ by the conjunction $\bigwedge_{w':R(w,w')} \langle w', q'\rangle$, and
  - each atom $(\Diamond, q')$ in θ by the disjunction $\bigvee_{w':R(w,w')} \langle w', q'\rangle$.
- The acceptance condition α is defined according to the acceptance condition α_ψ of $\mathcal{A}_\psi$: each set F in α_ψ is replaced by the set $W \times F$.

It is easy to see that $\mathcal{A}_{K,\psi}$ is of the same type as $\mathcal{A}_\psi$. In particular, if $\mathcal{A}_\psi$ is a WAA (with a partition $\{Q_1, Q_2, \ldots, Q_n\}$), then so is $\mathcal{A}_{K,\psi}$ (with a partition $\{W \times Q_1, W \times Q_2, \ldots, W \times Q_n\}$).

Proposition 5.12 [KVW00] *Consider a Kripke structure K and a branching temporal logic ψ.*

- $|\mathcal{A}_{K,\psi}| = O(|K| \cdot |\mathcal{A}_\psi|)$.
- $\mathcal{L}(\mathcal{A}_{K,\psi})$ *is nonempty if and only if* $K \models \psi$.

Proof: The claim about the size of $\mathcal{A}_{K,\psi}$ follows easily from the definition of $\mathcal{A}_{K,\psi}$. Indeed, $|W \times Q_\psi| = |W| \cdot |Q_\psi|$, $|\delta| = |W| \cdot |\delta_\psi|$, and $|\alpha| = |W| \cdot |\alpha_\psi|$.

To prove the correctness of the reduction, we show that $\mathcal{L}(\mathcal{A}_{K,\psi})$ is nonempty if and only if $\mathcal{A}_\psi$ accepts $\langle T_K, V_K\rangle$. Since $\mathcal{A}_\psi$ accepts $L(\psi)$, the latter holds if and only if $K \models \psi$. Given an accepting run of $\mathcal{A}_\psi$ over $\langle T_K, V_K\rangle$, we construct an accepting run of $\mathcal{A}_{K,\psi}$. Also, given an accepting run of $\mathcal{A}_{K,\psi}$, we construct an accepting run of $\mathcal{A}_\psi$ over $\langle T_K, V_K\rangle$.

Assume first that $\mathcal{A}_\psi$ accepts $\langle T_K, V_K\rangle$. Thus, there exists an accepting run $\langle T_r, r\rangle$ of $\mathcal{A}_\psi$ over $\langle T_K, V_K\rangle$. Recall that T_r is labeled with $\mathbb{N}^* \times Q_\psi$. A node $y \in T_r$ with $r(y) = (x, q)$ corresponds to a copy of $\mathcal{A}_\psi$ that is in the state q and reads the tree obtained by unwinding K from $V_K(x)$. Consider the tree $\langle T_r, r'\rangle$ where T_r is labeled with $0^* \times W \times Q_\psi$ and for every $y \in T_r$ with $r(y) = (x, q)$, we have $r'(y) = (0^{|x|}, V_K(x), q)$. We show that $\langle T_r, r'\rangle$ is an accepting run of $\mathcal{A}_{K,\psi}$. In fact, since $\alpha = W \times \alpha_\psi$, we only need to show that $\langle T_r, r'\rangle$ is a run of $\mathcal{A}_{K,\psi}$; acceptance follows from the fact that $\langle T_r, r\rangle$ is accepting. Intuitively, $\langle T_r, r'\rangle$ is a "legal" run, since the W-component in r' always agrees with V_K. This agreement is the only additional requirement of δ with respect to δ_ψ. Consider a node $y \in T_r$ with $r(y) = (x, q)$, $V_K(x) = w$. Let $\delta_\psi(q, w) = \theta$. Since $\langle T_r, r\rangle$ is a run of $\mathcal{A}_\psi$, there exists a set $\{(c_0, q_0), (c_1, q_1), \ldots, (c_n, q_n)\}$ satisfying θ, such that the successors of y in T_r are $y \cdot i$, for $1 \leq i \leq n$, each

labeled with $(x \cdot c_i, q_i)$. In $\langle T_r, r' \rangle$, by its definition, $r'(y) = (0^{|x|}, w, q)$ and the successors of y are $y \cdot i$, each labeled with $(0^{|x+1|}, w_{c_i}, q_i)$, where $\{w_0, \ldots, w_n\}$ is the set of successors of w in K. Let $\delta(q, a) = \theta'$. By the definition of δ, the set $\{(w_{c_0}, q_0), (w_{c_1}, q_1), \ldots, (w_{c_n}, q_n)\}$ satisfies θ'. Thus, $\langle T_r, r' \rangle$ is a run of $\mathcal{A}_{K,\psi}$.

Assume now that $\mathcal{A}_{K,\psi}$ accepts a^ω. Thus, there exists an accepting run $\langle T_r, r \rangle$ of $\mathcal{A}_{K,\psi}$. Recall that T_r is labeled with $0^* \times W \times Q_\psi$. Consider the tree $\langle T_r, r' \rangle$ labeled with $\mathbb{N}^* \times Q_\psi$, where $r'(\varepsilon) = (\varepsilon, q_0)$ and for every $y \cdot c \in T_r$ with $r'(y) \in \{x\} \times Q_\psi$ and $r(y \cdot c) = (0^{|x+1|}, w, q)$, we have $r'(y \cdot c) = (x \cdot i, q)$, where i is such that $V_K(x \cdot i) = w$. As in the previous direction, it is easy to see that $\langle T_r, r' \rangle$ is an accepting run of $\mathcal{A}_\psi$ over $\langle T_K, V_K \rangle$. □

Proposition 5.12 can be viewed as an automata-theoretic generalization of Theorem 4.1 in [EJS93].

Theorems 5.1, 5.4, 5.7, and 5.9, together with Theorem 4.3, then imply the following.

Theorem 5.13 [KVW00] *The model-checking problem*

- *for CTL can be solved in linear time and in space $O(m \log^2(mn))$, where m is the length of the formula and n is the size of the Kripke structure.*
- *for the alternation-free μ-calculus can be solved in linear time.*
- *for the μ-calculus is in NP $\cap$ co-NP.*
- *for CTL* can be solved in space $O(m(m + \log n)^2)$, where m is the length of the formula and n is the size of the Kripke structure.*

Now, let us define the *program complexity* [VW86a] of model checking as the complexity of this problem in terms of the size of the input Kripke structure; i.e., assuming the formula fixed. Consider the translation of CTL* formulas to HAA. Fixing the formula, we get an HAA of a fixed depth, making their nonemptiness test in NLOGSPACE. On the other hand, the 1-letter nonemptiness problem for weak alternating word automata is PTIME-complete [KVW00]. Thus, the restricted structure of HAAs is essential for a space-efficient nonemptiness test, inducing the following complexities for the program complexity of model checking.

Theorem 5.14 [KVW00] *The program complexity of the model-checking problem is*

- *NLOGSPACE-complete for CTL and CTL*.*
- *PTIME-complete for the alternation-free μ-calculus.*

References

[BB79] C. Beeri and P.A. Bernstein. Computational problems related to the design of normal form relational schemas. *ACM Trans. on Database Systems*, 4:30–59, 1979.

[BB87] B. Banieqbal and H. Barringer. Temporal logic with fixed points. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, volume 398 of *Lecture Notes in Computer Science*, pages 62–74. Springer, 1987.

[BC96a] G. Bhat and R. Cleaveland. Efficient local model-checking for fragments of the modal μ-calculus. In *Proc. 2nd Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*. Springer, 1996.

[BC96b] G. Bhat and R. Cleavland. Efficient model checking via the equational μ-calculus. In *Proc. 11th IEEE Symp. on Logic in Computer Science*, pages 304–312, 1996.

[Bee80] C. Beeri. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Trans. on Database Systems*, 5:241–259, 1980.

[BK08] C. Baier and J-P. Katoen. *Principles of model checking*. MIT Press, 2008.

[BK09] U. Boker and O. Kupferman. Co-ing Büchi made tight and helpful. In *Proc. 24th IEEE Symp. on Logic in Computer Science*, pages 245–254, 2009.

[BKR10] U. Boker, O. Kupferman, and A. Rosenberg. Alternation removal in Büchi automata. In *Proc. 37th Int. Colloq. on Automata, Languages, and Programming*, volume 6199, pages 76–87, 2010.

[Büc60] J.R. Büchi. Weak second-order arithmetic and finite automata. *Zeit. Math. Logik und Grundl. Math.*, 6:66–92, 1960.

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Int. Congress on Logic, Method, and Philosophy of Science. 1960*, pages 1–12. Stanford University Press, 1962.

[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 Languagues and Systems*, 8(2):244–263, 1986.

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[Cho74] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *Journal of Computer and Systems Science*, 8:117–141, 1974.

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1990.

[CY95] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42:857–907, 1995.

[DG84] W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[Don65] J.E. Doner. Decidability of the weak second-order theory of two successors. *Notices Amer. Math. Soc.*, 12:819, 1965.

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

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

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

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

[Elg61] C. Elgot. Decision problems of finite-automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, 1961.

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

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

[ES84] E.A. Emerson and A. P. Sistla. Deciding branching time logic. In *Proc. 16th ACM Symp. on Theory of Computing*, pages 14–24, 1984.

[FL79] M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and Systems Science*, 18:194–211, 1979.

[Gab72] D.M. Gabbay. Applications of trees to intermediate logics i. *Journal of Symbolic Logic*, 37:135–138, 1972.

[GH82] Y. Gurevich and L. Harrington. Trees, automata, and games. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 60–65. ACM Press, 1982.

[GO01] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *Proc. 13th Int. Conf. on Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2001.

[GPVW95] R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In P. Dembiski and M. Sredniawa, editors, *Protocol Specification, Testing, and Verification*, pages 3–18. Chapman & Hall, 1995.

[HT87] T. Hafer and W. Thomas. Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In *Proc. 14th Int. Colloq. on Automata, Languages, and Programming*, volume 267 of *Lecture Notes in Computer Science*, pages 269–279. Springer, 1987.

[Jon75] N.D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and Systems Science*, 11:68–75, 1975.

[JW95] D. Janin and I. Walukiewicz. Automata for the modal μ-calculus and related results. In *20th Int. Symp. on Mathematical Foundations of Computer Science*, volume 969 of *Lecture Notes in Computer Science*, pages 552–562. Springer, 1995.

[Kam68] J.A.W. Kamp. *Tense Logic and the Theory of Order*. PhD thesis, UCLA, 1968.

[KL06] O. Kupferman and R. Lampert. On the construction of fine automata for safety properties. In *4th Int. Symp. on Automated Technology for Verification and Analysis*, volume 4218 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2006.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KPB94] S.C. Krishnan, A. Puri, and R.K. Brayton. Deterministic ω-automata vis-a-vis deterministic Büchi automata. In *Algorithms and Computations*, volume 834 of *Lecture Notes in Computer Science*, pages 378–386. Springer, 1994.

[KR10] O. Kupferman and A. Rosenberg. The blow-up in translating LTL to deterministic automata. In *Proc. 6th Workshop on Model Checking and Artificial Intelligence*, volume 6572 of *Lecture Notes in Artificial Intelligence*, pages 85–94. Springer, 2010.

[KV01] O. Kupferman and M.Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[KV05] O. Kupferman and M.Y. Vardi. From linear time to branching time. *ACM Transactions on Computational Logic*, 6(2):273–294, 2005.

[KVW00] O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[KW08] D. Kähler and T. Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In *Proc. 35th Int. Colloq. on Automata, Languages, and Programming*, volume 525 of *Lecture Notes in Computer Science*, pages 724–735. Springer, 2008.

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

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

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer, 1985.

[MH84] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

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

[MS87] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.

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

[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. Colloq. on Automata, Languages, and Programming*, volume 226 of *Lecture Notes in Computer Science*, pages 275 – 283. Springer, 1986.

[MSS88] D.E. Muller, A. Saoudi, and P. E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proc. 3rd IEEE Symp. on Logic in Computer Science*, pages 422–427, 1988.

[Pit06] N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 255–264. IEEE press, 2006.

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

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

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

[QS82] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 8th ACM Symp. on Principles of Programming Languages*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

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

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

[RS59] M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.

[Saf88] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, 1988.

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

[Sch09a] S. Schewe. Büchi complementation made tight. In *Proc. 26th Symp. on Theoretical Aspects of Computer Science*, volume 3 of *LIPIcs*, pages 661–672. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.

[Sch09b] S. Schewe. Tighter bounds for the determinisation of büchi automata. In *Proc. 12th Int. Conf. on Foundations of Software Science and Computation Structures*, volume 5504 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2009.

[SE84] R.S. Street and E.A. Emerson. An elementary decision procedure for the μ-calculus. In *Proc. 11th Int. Colloq. on Automata, Languages, and Programming*, volume 172, pages 465–472. Springer, 1984.

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

[Sis83] A.P. Sistla. *Theoretical issues in the design of distributed and concurrent systems.* PhD thesis, Harward University, 1983.

[Sis94] A.P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6:495–511, 1994.

[Slu85] G. Slutzki. Alternating tree automata. *Theoretical Computer Science*, 41:305–318, 1985.

[Str82] R.S. Streett. Propositional dynamic logic of looping and converse. *Information and Control*, 54:121–141, 1982.

[SV89] S. Safra and M.Y. Vardi. On ω-automata and temporal logic. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 127–137, 1989.

[SVW87] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 133–191, 1990.

[Tra62] B.A. Trakhtenbrot. Finite automata and monadic second order logic. *Siberian Math. J*, 3:101–131, 1962. Russian; English translation in: AMS Transl. 59 (1966), 23-55.

[TV10] D. Tabakov and M.Y. Vardi. Monitoring temporal systemc properties. In *Proc. 6th ACM&IEEE Int'l Conf. on Formal Methods and Models for Co-Design*, pages 123–132, 2010.

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

[Var94] M.Y. Vardi. Nontraditional applications of automata theory. In *Proc. 11th Symp. on Theoretical Aspects of Computer Science*, volume 789 of *Lecture Notes in Computer Science*, pages 575–597. Springer, 1994.

[Var95] M.Y. Vardi. Alternating automata and program verification. In *Computer Science Today –Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 471–485. Springer, 1995.

[VW84] M.Y. Vardi and P. Wolper. Yet another process logic. In *Logics of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 501–512. Springer, 1984.

[VW86a] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st IEEE Symp. on Logic in Computer Science*, pages 332–344, 1986.

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

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

[Wil99] T. Wilke. CTL^+ is exponentially more succinct than CTL. In *Proc. 19th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 1999.

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