17

AUTOMATA-THEORETIC TECHNIQUES FOR TEMPORAL REASONING

Moshe Y. Vardi

1 INTRODUCTION

This chapter describes an automata-theoretic approach to temporal reasoning. The basic idea underlying this approach is that for any temporal formula we can construct a finite-state automaton that accepts the computations that satisfy the formula. For linear temporal logics the automaton runs on infinite words while for branching temporal logics the automaton runs on infinite trees. The simple combinatorial structures that emerge from the automata-theoretic approach decouple the logical and algorithmic components of temporal reasoning and yield clear and asymptotically optimal algorithms.

Temporal logics, which are modal logics geared towards the description of the temporal ordering of events, have been adopted as a powerful tool for specifying behavior concurrent programs and for verifying that such programs meet their specifications [47, 57]. One of the most significant developments in this area is the discovery of algorithmic methods for verifying temporal logic properties of *finite-state* programs [15, 44, 60, 81].

(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, which points to the instruction currently being executed, and the like. Finite-state programs have finitely many possible states, which means that the variables range over finite domains and recursion, if allowed, is bounded in depth.) This derives its significance from the fact that many synchronization, coordination and communication protocols can be modeled as finite-state programs [46, 63]. Finite-state programs can be modeled by transition systems where each state has a bounded description, and hence can be characterized by a fixed number of Boolean atomic propositions. This means that a finite-state program can be viewed as a finite *propositional Kripke structure* and that its properties can be specified using *propositional* temporal logic. Thus, to verify the correctness of the program with respect to a desired behavior, one only has to check that the program, modeled as a finite Kripke structure, is a model of (satisfies) the propositional temporal logic formula that specifies that behavior. Hence the name *model checking* for the verification methods derived from this viewpoint. An extensive survey of model checking can be found in [16, 37]. Model checking is an *algorithmic* approach to program verification. It is different from the *deductive approach*, in which in which a person, perhaps aided by computers, use deductive techniques to prove that a program satisfy its specification [48].

We distinguish between two types of temporal logics: linear and branching (see [43] and general discussion of temporal structures in Chapter **??**). In linear temporal logics, each moment in time has a unique possible future, while in branching temporal logics, each moment in time may split into several possible futures. (For an extensive discussion of various temporal logics, see [22].) For both types of temporal logics, a close and fruitful connection with the theory of automata on infinite structures has been developed. The basic idea is to associate with each temporal logic formula a finite automaton on infinite structures that accepts the computations that satisfy the formula. For linear temporal logic the structures are infinite words [66, 45, 68, 83], while for branching temporal logic the structures are infinite trees [30, 69, 21, 26, 82]. This enables the reduction of temporal logic decision problems, such as satisfiability and model checking, to known automata-theoretic problems, such as nonemptiness, yielding clean and asymptotically optimal algorithms. This reduction is the subject matter of this chapter. (See also Chapter **??** for a general discussion of temporal reasoning and Chapters **??** and **??** for discussion of modal reasoning.)

Initially, the translations in the literature from temporal logic formulas to automata used *nondeterministic* automata (cf. [34, 82, 83]). These translations have two disadvantages. First, the translation itself is rather nontrivial; For example, in [82, 83] the translations go through a series of ad-hoc intermediate representations in an attempt to simplify the translation. Second, for both linear and branching temporal logics there is an exponential blow-up involved in going from formulas to automata. This suggests that any algorithm that uses these translations as one of its steps is going to be an exponential-time algorithm. Thus, the automata-theoretic approach did not seem to be applicable to branching-time model checking, which in many cases can be done in linear execution time [15, 17, 60].

In the mid 1990s, it was shown that if one uses *alternating* automata rather than *nondeterministic* automata, then these problems can be addressed [42, 74]. Alternating automata generalize the standard notion of nondeterministic automata by allowing several successor states to go down along the same word or the same branch of the tree. In this chapter we argue that *alternating automata* offer the key to a comprehensive and satisfactory automata-theoretic framework for temporal logics. We demonstrate this claim by showing how alternating automata can be used to derive model-checking and satisfiability algorithms for both linear and branching temporal logics. The key observation is that while the translation from temporal logic formulas to nondeterministic automata is exponential [82, 83], the translation to alternating automata is linear [27, 42, 53, 74]. Thus, the advantage of alternating automata is that they enable one to decouple the logic from the combinatorics. The translations from formulas to automata handle the logic, and the algorithms that handle the automata handles the combinatorics.

Historical Note: The connection between logic and automata goes back to work in the early 1960s [6, 20, 73] on monadic second-order logic and automata over finite words. This was extended in [7] to infinite words, in [19, 72] to finite trees, and in [61] to infinite trees. As temporal logics can be expressed in first-order or monadic second-order logic [38, 35], 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 [33]. A direct translation to automata was proposed first in [70] in the context of propositional dynamic logic. A direct translation from temporal logic to automata was first given in [85] (see also [83] for linear time and in [80] for branching time). The translation to alternating automata was first proposed in [53] and pursued further in [42, 74, 75].

2 AUTOMATA THEORY

2.1 Words and Trees

We are given a finite nonempty alphabet Σ. A *finite word* is an element of Σ^*, i.e., a finite sequence $a_0, \ldots, a_n$ of symbols from Σ. An *infinite word* is an element of Σ^ω, i.e., an infinite sequence $a_0, a_1, \ldots$ of symbols from Σ.

A *tree* is a (finite or infinite) connected directed graph, with one node designated as the *root* and denoted by ε, and in which every non-root node has a unique parent (s is the *parent* of t and t is a *child* of s if there is an edge from s to t) and the root ε has no parent. The *arity* of a node x in a tree τ, denoted $arity(x)$, is the number of children of x in τ. The *level* of a node x, denoted $|x|$, is its distance from the root; in particular, $|\varepsilon| = 0$. Let N denote the set of positive integers. A *tree τ over N* is a subset of N^*, such that if $x \cdot i \in \tau$, where $x \in N^*$ and $i \in N$, then $x \in \tau$, there is an edge from x to $x \cdot i$, and if $i > 1$ then also $x \cdot (i-1) \in \tau$. By definition, the empty sequence ε is the root of such a tree. Let $\mathcal{D} \subseteq N$. We say that a tree τ is a *$\mathcal{D}$-tree* if τ is a tree over N and $arity(x) \in \mathcal{D}$ for all $x \in \tau$. A tree is called *leafless* if every node has at least one child.

A *branch* $\beta = x_0, x_1, \ldots$ of a tree is a maximal sequence of nodes such that x_0 is the root and x_i is the parent of x_{i+1} for all $i > 0$. Note that β can be finite or infinite; if it is finite, then the last node of the branch has no children. A *Σ-labeled tree*, for a finite alphabet Σ, is a pair $(\tau, \mathcal{T})$, where τ is a tree and $\mathcal{T}$ is a mapping $\mathcal{T} : nodes(\tau) \rightarrow \Sigma$ that assigns to every node a label. We often refer to $\mathcal{T}$ as the labeled tree, leaving its domain implicit. A branch $\beta = x_0, x_1, \ldots$ of $\mathcal{T}$ defines a word $\mathcal{T}(\beta) = \mathcal{T}(x_0), \mathcal{T}(x_1), \ldots$ consisting of the sequence of labels along the branch.

2.2 Nondeterministic Automata on Infinite Words

A *nondeterministic Büchi automaton* A on words is a tuple $(\Sigma, S, S^0, \rho, F)$, where Σ is a finite nonempty *alphabet*, S is a finite nonempty set of *states*, $S^0 \subseteq S$ is the set of *initial* states, $F \subseteq S$ is the set of *accepting* states, and $\rho : S \times \Sigma \rightarrow 2^S$ is a *transition function*. Intuitively, $\rho(s,a)$ is the set of states that A can move into when it is in state s and it reads the symbol a. 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 symbol.

A run r of A on an infinite word $w = a_0, a_1, \ldots$ over Σ is a sequence $s_0, s_1, \ldots$, where $s_0 \in S^0$ and $s_{i+1} \in \rho(s_i, a_i)$, for all $i \geq 0$. We define $\lim(r)$ to be the set $\{s \mid s = s_i$ for infinitely many i's$\}$, i.e., the set of states that occur in r infinitely often. Since S is finite, $\lim(r)$ is necessarily nonempty. The run r is *accepting* if there is some accepting state that repeats in r infinitely often, i.e., $\lim(r) \cap F \neq \emptyset$. The infinite word w is *accepted* by A if there is an accepting run of A on w. The set of infinite words accepted by A is denoted $L_\omega(A)$.

An important feature of nondeterministic Büchi automata is their closure under intersection.

PROPOSITION 1. [12] *Let A_1 and A_2 be nondeterministic Büchi automata with n_1 and n_2 states, respectively. Then there is a Büchi automaton A with $O(n_1 n_2)$ states such that $L_\omega(A) = L_\omega(A_1) \cap L_\omega(A_2)$.*

One of the most fundamental algorithmic issues in automata theory is testing whether a given automaton is "interesting", i.e., whether it accepts some input. A Büchi automaton A is *nonempty* if $L_\omega(A) \neq \emptyset$. The *nonemptiness problem* for automata is to decide, given an automaton A, whether A is nonempty. It turns out that testing nonemptiness for Büchi automata is easy A accepts some word iff in the graph $G_A = (S, E_A)$, where $E_A = \{(s,t) | t \in \rho(s,a)$ for some $a \in \Sigma\}$, there is a path from S_0 that reaches some state $f \in F$ and then cycles back to f. This can be checked using depth-first search [71] or space-efficient search [65].

PROPOSITION 2.

1. [29, 28] *The nonemptiness problem for nondeterministic Büchi automata is decidable in linear time.*
2. [83] *The nonemptiness problem for nondeterministic Büchi automata of size n is decidable in space $O(\log^2 n)$.*

2.3 Alternating Automata on Infinite Words

Nondeterminism gives a computing device the power of existential choice. Its dual gives a computing device the power of universal choice. It is therefore natural to consider computing devices that have the power of both existential choice and universal choice. Such devices are called *alternating*. Alternation was studied in [11] in the context of Turing machines and in [5, 11] for finite automata. The alternation formalisms in [5] and [11] are different, though equivalent. We follow here the formalism of [5], which was extended in [55] to automata on infinite structures.

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**. Let $Y \subseteq X$. We say that Y *satisfies* a formula $\theta \in \mathcal{B}^+(X)$ if the truth assignment that assigns *true* to the members of Y and assigns *false* to the members of $X - Y$ satisfes θ. For example, the sets $\{s_1, s_3\}$ and $\{s_1, s_4\}$ both satisfy the formula $(s_1 \vee s_2) \wedge (s_3 \vee s_4)$, while the set $\{s_1, s_2\}$ does not satisfy this formula.

Consider a nondeterministic automaton $A = (\Sigma, S, s^0, \rho, F)$. The transition function ρ maps a state $s \in S$ and an input symbol $a \in \Sigma$ to a set of states. Each element in this set is a possible nondeterministic choice for the automaton's next state. We can represent ρ using $\mathcal{B}^+(S)$; for example, $\rho(s,a) = \{s_1, s_2, s_3\}$ can be written as $\rho(s,a) = s_1 \vee s_2 \vee s_3$. In alternating automata, $\rho(s,a)$ can be an arbitrary formula from $\mathcal{B}^+(S)$. We can have, for instance, a transition

$$\rho(s,a) = (s_1 \wedge s_2) \vee (s_3 \wedge s_4),$$

meaning that the automaton accepts the word aw, where a is a symbol and w is a word, when it is in the state s if it accepts the word w from both s_1 and s_2 or from both s_3 and s_4. Thus, such a transition combines the features of existential choice (the disjunction in the formula) and universal choice (the conjunctions in the formula).

Formally, an *alternating Büchi automaton* is a tuple $A = (\Sigma, S, s^0, \rho, F)$, where Σ is a finite nonempty alphabet, S is a finite nonempty set of states, $s^0 \in S$ is an initial state, F is a set of accepting states, and $\rho : S \times \Sigma \to \mathcal{B}^+(S)$ is a transition function. As a convention, if $\rho(s,a)$ is not specified, we assume that it is **false**.

Because of the universal choice in alternating transitions, a run of an alternating automaton is a tree rather than a sequence. A run of A on an infinite word $w = a_0 a_1 \ldots$ is an S-labeled tree r such that $r(\varepsilon) = s^0$ and the following holds:

> if $|x| = i$, $r(x) = s$, and $\rho(s, a_i) = \theta$, then x has k children $x_1, \ldots, x_k$, for some $k \leq |S|$, and $\{r(x_1), \ldots, r(x_k)\}$ satisfies θ.

For example, if $\rho(s_0, a_0)$ is $(s_1 \vee s_2) \wedge (s_3 \vee s_4)$, then the nodes of the run tree at level 1 include the label s_1 or the label s_2 and also include the label s_3 or the label s_4. Note that the run can also have finite branches; if $|x| = i$, $r(x) = s$, and $\rho(s, a_i) = \textbf{true}$, then x does not need to have any children. On the other hand, we cannot have $\rho(s, a_i) = \textbf{false}$, since **false** is not satisfiable. The run is *accepting* if $\lim(r(\beta)) \cap F \neq \emptyset$ for every branch $\beta = x_0, x_1, \ldots$ of the run. that is, for every infinite branch $\beta = x_0, x_1, \ldots$, we have that $r(x_i) \in F$ for infinitely many i's. A word w is accepted by A is A has an accepting run on w.

What is the relationship between alternating Büchi automata and nondeterministic Büchi automata? It is easy to see that alternating Büchi automata generalize nondeterministic Büchi automata; nondeterministic automata correspond to alternating automata where the transitions are pure disjunctions. It turns out that they have the same expressive power (although alternating Büchi automata are more succinct than nondeterministic Büchi automata).

PROPOSITION 3. [51] *Let A be an alternating Büchi automaton with n states. Then there is a nondeterministic Büchi automaton A_{nd} with 3^n states such that $L_\omega(A_{nd}) = L_\omega(A)$.*

By combining Propositions 2 and 3 (with its exponential blowup), we can obtain a nonemptiness test for alternating Büchi automata.

PROPOSITION 4.

1. *The nonemptiness problem for alternating Büchi automata is decidable in exponential time.*

2. *The nonemptiness problem for alternating Büchi automata is decidable in quadratic space.*

2.4 Nondeterministic Automata on Infinite Trees

We now consider automata on labeled leafless $\mathcal{D}$-trees. A *nondeterministic Büchi tree automaton* A is a tuple $(\Sigma, \mathcal{D}, S, S^0, \rho, F)$. Here Σ is a finite alphabet, $\mathcal{D} \subset N$ is a finite set of arities, S is a finite set of states, $S^0 \subseteq S$ is the set of initial states, $F \subseteq S$ is a set of accepting states, and $\rho : S \times \Sigma \times \mathcal{D} \rightarrow 2^{S^*}$ is a transition function, where $\rho(s, a, k) \subseteq S^k$ for each $s \in S$, $a \in \Sigma$, and $k \in \mathcal{D}$. Thus, $\rho(s, a, k)$ is a set of k-tuples of states. Intuitively, when the automaton is in state s and it is reading a k-ary node x of a tree $\mathcal{T}$, it nondeterministically chooses a k-tuple $\langle s_1, \ldots, s_k \rangle$ in $\rho(s, \mathcal{T}(x))$, makes k copies of itself, and then moves to the node $x \cdot i$ in the state s_i for $i = 1, \ldots, k$. A *run* $r : \tau \rightarrow S$ of A on a Σ-labeled $\mathcal{D}$-tree $\mathcal{T}$ is an S-labeled $\mathcal{D}$-tree such that the root is labeled by an initial state and the transitions obey the transition function ρ; that is, $r(\varepsilon) \in S^0$, and for each node x such that $arity(x) = k$, we have $\langle r(x \cdot 1), \ldots, r(x \cdot k) \rangle \in \rho(r(x), \mathcal{T}(x), k)$. The run is *accepting* if $\lim(r(\beta)) \cap F \neq \emptyset$ for every branch $\beta = x_0, x_1, \ldots$ of τ; that is, for every branch $\beta = x_0, x_1, \ldots$, we have that $r(x_i) \in F$ for infinitely many i's. The set of trees accepted by A is denoted $T_\omega(A)$. It is easy to see that nondeterministic Büchi automata on infinite words are essentially Büchi automata on $\{1\}$-trees.

Again, a key issue is testing emptiness.

PROPOSITION 5. [62, 82] *The nonemptiness problem for nondeterministic Büchi tree automata is decidable in quadratic time.*

2.5 Alternating Automata on Infinite Trees

An *alternating Büchi tree automaton* A is a tuple $(\Sigma, \mathcal{D}, S, s^0, \rho, F)$. Here Σ is a finite alphabet, $\mathcal{D} \subset N$ is a finite set of arities, S is a finite set of states, $s^0 \in S$ is an initial state, $F \subseteq S$ is a set of accepting states, and $\rho : S \times \Sigma \times \mathcal{D} \rightarrow \mathcal{B}^+(N \times S)$ is a partial transition function, where $\rho(s, a, k) \in \mathcal{B}^+(\{1, \ldots, k\} \times S)$ for each $s \in S$, $a \in \Sigma$, and $k \in \mathcal{D}$ such that $\rho(s, a, k)$ is defined. For example, $\rho(s, a, 2) = ((1, s_1) \vee (2, s_2)) \wedge ((1, s_3) \vee (2, s_1))$ means that the automaton can choose between four splitting possibilities. In the first possibility, one copy proceeds in direction 1 in the state s_1 and one copy proceeds in direction 1 in the state s_3. In the second possibility, one copy proceeds in direction 1 in the state s_1 and one copy proceeds in direction 2 in the state s_1. In the third possibility, one copy proceeds in direction 2 in the state s_2 and one copy proceeds in direction 1 in the state s_3. Finally, in the fourth possibility, one copy proceeds in direction 2 in the state s_2 and one copy proceeds in direction 2 in the state s_1. Note that it is possible for more than one copy to proceed in the same direction.

A run r of an alternating Büchi tree automaton A on a Σ-labeled leafless $\mathcal{D}$-tree $\langle \tau, \mathcal{T} \rangle$ is a $\tau \times S$-labeled tree. Each node of r corresponds to a node of τ. A node in r, labeled by (x, s), describes a copy of the automaton that reads the node x of τ in the state s.

Note that many nodes of r can correspond to the same node of τ; in contrast, in a run of a nondeterministic automaton on $\langle \tau, \mathcal{T} \rangle$ 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, r is a Σ_r-labeled tree $\langle \tau_r, \mathcal{T}_r \rangle$ where $\Sigma_r = \tau \times S$ and $\langle \tau_r, \mathcal{T}_r \rangle$ satisfies the following:

1. $\mathcal{T}_r(\varepsilon) = (\varepsilon, s^0)$.

2. Let $y \in \tau_r$, $\mathcal{T}_r(y) = (x, s)$, $arity(x) = k$, and $\rho(s, \mathcal{T}(x), k) = \theta$. Then there is a set $Q = \{(c_1, s_1), (c_1, s_1), \ldots, (c_n, s_n)\} \subseteq \{1, \ldots, k\} \times S$ such that

   - Q satisfies θ, and
   - for all $1 \leq i \leq n$, we have $y \cdot i \in \tau_r$ and $\mathcal{T}_r(y \cdot i) = (x \cdot c_i, s_i)$.

For example, if $\langle \tau, \mathcal{T} \rangle$ is a tree with $arity(\varepsilon) = 2$, $\mathcal{T}(\varepsilon) = a$ and $\rho(s^0, a) = ((1, s_1) \vee (1, s_2)) \wedge ((1, s_3) \vee (1, s_1))$, then the nodes of $\langle \tau_r, \mathcal{T}_r \rangle$ at level 1 include the label $(1, s_1)$ or $(1, s_2)$, and include the label $(1, s_3)$ or $(1, s_1)$.

As with alternating Büchi automata on words, alternating Büchi tree automata are as expressive as nondeterministic Büchi tree automata.

PROPOSITION 6. [52, 56] *Let A be an alternating Büchi tree automaton with n states. Then there is a nondeterministic Büchi tree automaton A_n with 3^n states such that $T_\omega(A_n) = T_\omega(A)$.*

By combining Propositions 5 and 6 (with its exponential blowup), we can obtain a nonemptiness test for alternating Büchi tree automata.

PROPOSITION 7. *The nonemptiness problem for alternating Büchi tree automata is decidable in exponential time.*

Does the size of the alphabet affect the complexity of the nonemptiness problem? For nondeterministic tree automata, the nonemptiness problem is reducible to the 1-letter nonemptiness problem, that is, to the nonemptiness problem for nondeterministic tree automata over 1-letter alphabets (i.e., $|\Sigma| = 1$). Indeed, instead checking the nonemptiness of an automaton $A = (\Sigma, \mathcal{D}, S, S^0, \rho, F)$, one can check the nonemptiness of the automaton $A' = (\{a\}, \mathcal{D}, S, S^0, \rho', F)$ where for all $s \in S$, we have $\rho'(s, a, k) = \bigcup_{a \in \Sigma} \rho(s, a, k)$. It is easy to see that A accepts some tree iff A' accepts some a-labeled tree. This can be viewed as if A' first guesses a Σ-labeling for the input tree and then proceeds like A on this Σ-labeled tree.

This reduction is not valid for alternating tree automata. Suppose that we defined A' by taking $\rho'(s, a, k) = \bigvee_{a \in \Sigma} \rho(s, a, k)$. Then, if A' accepts some a-labeled tree, it still does not guarantee that A accepts some tree. A necessary condition for the validity of the reduction is that different copies of A' that run on the same subtree guess the same Σ-labeling for this subtree. Nothing, however, prevents one copy of A' to proceed according to one labeling and another copy to proceed according to a different labeling. This problem does not occur when A is defined over a singleton alphabet. There, it is guaranteed that all copies proceed according to the same (single) labeling.

As we see later, in our applications we sometimes do have 1-letter alphabets, which makes the 1-letter nonemptiness problem for alternating automata of interest. It turns out that this problem is easier than the general nonemptiness problem. Actually, it is as easy as the nonemptiness problem for nondeterministic Büchi tree automata (Proposition 5). This result requires also *uniformity*, i.e., $|\mathcal{D}| = 1$.

PROPOSITION 8. [42] *The 1-letter nonemptiness problem for uniform alternating Büchi tree automata is decidable in quadratic time.*

As we shall see later, the alternating automata in our applications have a special structure, studied first in [54]. A *weak alternating tree automaton* (WAA) is an alternating Büchi tree automaton in which there exists a partition of the state set S into disjoint sets $S_1, \ldots, S_n$ such that for each set S_i, either $S_i \subseteq F$, in which case S_i is an *accepting set*, or $S_i \cap F = \emptyset$, in which case S_i is a *rejecting set*. In addition, there exists a partial order $\leq$ on the collection of the S_i's such that for every $s \in S_i$ and $s' \in S_j$ for which s' occurs in $\rho(s, a, k)$, for some $a \in \Sigma$ and $k \in \mathcal{D}$, we have $S_j \leq S_i$. Thus, transitions from a state in S_i lead to states in either the same S_i or a lower one. It follows that every infinite path of a run of a WAA ultimately gets "trapped" within some S_i. The path then satisfies the acceptance condition if and only if S_i is an accepting set. That is, a run visits infinitely many states in F if and only if it gets trapped in an accepting set. The number of sets in the partition of S is defined as the *depth* of the automaton.

It turns out that the nonemptiness problem for WAA on 1-letter alphabets is easier than nonemptiness problem for alternating Büchi automata on 1-letter alphabets.

PROPOSITION 9. [42] *The 1-letter nonemptiness problem for uniform weak alternating tree automata is decidable in linear time.*

As we shall see, the WAA that we use have an even more special structure. In these WAA, each set S_i can be classified as either *transient*, *existential*, or *universal*, such that for each set S_i and for all $s \in Q_i$, $a \in \Sigma$, and $k \in \mathcal{D}$, the following hold:

1. If S_i is transient, then $\rho(s, a, k)$ contains no elements of S_i.
2. If S_i is existential, then $\rho(s, a, k)$ only contains *disjunctively related* elements of S_i (i.e. if the transition is rewritten in disjunctive normal form, there is at most one element of S_i in each disjunct).
3. If Q_i is universal, then $\rho(s, a, k)$ only contains *conjunctively related* elements of S_i (i.e. if the transition is rewritten in conjunctive normal form, there is at most one element of Q_i in each conjunct).

This means that it is only when moving from one S_i to the next, that alternation actually occurs (alternation is moving from a state that is conjunctively related to states in its set to a state that is disjunctively related to states in its set, or vice-versa). In other words, when a copy of the automaton visits a state in some existential set S_i, then as long as it stays in this set, it proceeds in an "existential mode"; namely, it imposes only existential requirement on its successors in S_i. Similarly, when a copy of the automaton visits a state in some universal set S_i, then as long as it stays in this set, it proceeds in a "universal mode". Thus, whenever a copy alternates modes, it must be that it moves from one S_i to the next. We call a WAA that satisfies this property a *limited-alternation*[1] WAA.

PROPOSITION 10. [42] *The 1-letter nonemptiness problem for uniform limited-alternation WAA of size n and depth m can be solved in space $O(m \log^2 n)$.*

[1]The term used in [42] is *hesitant*.

3 TEMPORAL LOGICS AND ALTERNATING AUTOMATA

3.1 Linear Temporal Logic

Formulas of *linear temporal logic* (LTL) are built from a set *Prop* of atomic propositions and are closed under the application of Boolean connectives, the unary temporal connective X (*next*), and the binary temporal connectives U (*until*) and R (*release*) [22]. LTL is interpreted over *computations.* A computation is a function $\pi : \omega \to 2^{Prop}$, which assigns truth values to the elements of *Prop* at each time instant (natural number). (This corresponds to using the ordered natural numbers as the frame; see Chapter **??**). A computation π and a point $i \in \omega$ satisfies an LTL formula φ, denoted $\pi, i \models \varphi$, under the following conditions:

- $\pi, i \models p$ for $p \in Prop$ iff $p \in \pi(i)$.
- $\pi, i \models \xi \wedge \psi$ iff $\pi, i \models \xi$ and $\pi, i \models \psi$.
- $\pi, i \models \neg\varphi$ iff not $\pi, i \models \varphi$
- $\pi, i \models X\varphi$ iff $\pi, i+1 \models \varphi$.
- $\pi, i \models \xi U \psi$ iff for some $j \geq i$, we have $\pi, j \models \psi$ and for all k, $i \leq k < j$, we have $\pi, k \models \xi$.[2]
- $\pi, i \models \xi R \psi$ iff for all $j \geq i$, if $\pi, j \not\models \psi$, then for some k, $i \leq k < j$, we have $\pi, k \models \xi$.

Note that $\neg(X\varphi)$ is equivalent to $X(\neg\varphi)$ and $\neg(\xi U \psi)$ is equivalent to $(\neg\xi)R\psi$. This implies that we can assume that formulas are in *positive normal form*, in which negations are applied only to atomic propositions. This normal form is obtained by pushing negations inward as far as possible, using De Morgan's laws and dualities as above. For example, the formula $G(\neg\mathsf{request} \vee (\mathsf{request} U \mathsf{grant}))$ says that whenever a request is made it holds continuously until it is eventually granted. We say that π *satisfies* a formula φ, denoted $\pi \models \varphi$, iff $\pi, 0 \models \varphi$.

Computations can also be viewed as infinite words over the alphabet 2^{Prop}. It turns out that the computations satisfying a given formula are exactly those accepted by some finite automaton on infinite words. The following theorem establishes a very simple translation between LTL and alternating Büchi automata on infinite words.

THEOREM 11. [53, 74] *Given an LTL formula φ, one can build an alternating Büchi automaton $A_\varphi = (\Sigma, S, s^0, \rho, F)$, where $\Sigma = 2^{Prop}$ and $|S|$ is in $O(|\varphi|)$, such that $L_\omega(A_\varphi)$ is exactly the set of computations satisfying the formula φ.*

Proof. The set S of states consists of all subformulas of φ. The initial state s^0 is φ itself. The set F of accepting states consists of all formulas in S of the form $(\xi R \psi)$. It remains to define the transition function ρ.

- $\rho(p, a) = \mathbf{true}$ if $p \in a$,
- $\rho(\neg p, a) = \mathbf{true}$ if $p \notin a$,

[2] Note that our U operator is not strict; cf. Chapter **??**.

- $\rho(p, a) = \mathbf{false}$ if $p \notin a$,
- $\rho(\neg p, a) = \mathbf{false}$ if $p \in a$,
- $\rho(\xi \wedge \psi, a) = \rho(\xi, a) \wedge \rho(\psi, a)$,
- $\rho(\xi \vee \psi, a) = \rho(\xi, a) \vee \rho(\psi, a)$,
- $\rho(X\psi, a) = \psi$,
- $\rho(\xi U \psi, a) = \rho(\psi, a) \vee (\rho(\xi, a) \wedge \xi U \psi)$.
- $\rho(\xi R \psi, a) = \rho(\psi, a) \wedge (\rho(\xi, a) \vee \xi R \psi)$.

∎

By applying Proposition 3, we now get:

COROLLARY 12. [83] *Given an LTL formula φ, one can build a nondeterministic Büchi automaton $A_\varphi = (\Sigma, S, s^0, \rho, F)$, where $\Sigma = 2^{Prop}$ and $|S|$ is in $2^{O(|\varphi|)}$, such that $L_\omega(A_\varphi)$ is exactly the set of computations satisfying the formula φ.*

For a description of optimized translations from LTL to automata, seefor example, [32]. While our focus here is on LTL, the automata-theoretic approach applies also to more expressive, recent industrial specification languages such as ForSpec [2]; see [1, 10].

3.2 Branching Temporal Logic

The branching temporal logic CTL (Computation Tree Logic) provides temporal connectives that are composed of a path quantifier immediately followed by a single linear temporal connective [22]. The path quantifiers are A ("for all paths") and E ("for some path"). The linear-time connectives are X, U, and R. Thus, given a set $Prop$ of atomic propositions, a CTL formula in positive normal form is one of the following:

- p or $\neg p$, for all $p \in AP$,
- $\xi \wedge \psi$ or $\xi \vee \psi$, where ξ and ψ are CTL formulas.
- $EX\xi$, $AX\xi$, $E(\xi U \psi)$, $A(\xi U \psi)$, $E(\xi R \psi)$, or $A(\xi R \psi)$, where ξ and ψ are CTL formulas.

The semantics of CTL is defined with respect to *programs*. A program over a set $Prop$ of atomic propositions is a structure of the form $P = (W, w^0, R, V)$, where W is a set of states, $w^0 \in W$ is an initial state, $R \subseteq W^2$ is a total accessibility relation (i.e., every state can access at least one state), and $V : W \to 2^{Prop}$ assigns truth values to propositions in $Prop$ for each state in W. The intuition is that W describes all the states that the program could be in (where a state includes the content of the memory, registers, buffers, program counter, etc.), R describes all the possible transitions between states (allowing for nondeterminism), and V relates the states to the propositions (e.g., it tells us in what states the proposition `request` is true); see [47] for a discussion on modelling programs. The assumption that R is total (i.e., that every state has an R-successor) is for technical convenience. We can view a terminated execution as repeating forever its last state. We say that P is a *finite-state* program if W is finite. A *path* in P is a sequence of states, $\mathbf{u} = u_0, u_1, \ldots$ such that for every $i \geq 0$, we have that $u_i R u_{i+1}$ holds. Such a path is called a u_0-path.

A program $P = (W, w^0, R, V)$ and a state $u \in W$ satisfies a CTL formula φ, denoted $P, u \models \varphi$, under the following conditions:

- $P, u \models p$ for $p \in Prop$ if $p \in V(u)$.
- $P, u \models \neg p$ for $p \in Prop$ if $p \notin V(u)$.
- $P, u \models \xi \wedge \psi$ iff $P, u \models \xi$ and $P, u \models \psi$.
- $P, u \models \xi \vee \psi$ iff $P, u \models \xi$ or $P, u \models \psi$.
- $P, u \models EX\varphi$ if $P, v \models \varphi$ for some v such that uRv holds.
- $P, u \models AX\varphi$ if $P, v \models \varphi$ for all v such that uRv holds.
- $P, u \models E(\xi U \psi)$ if there exist a u-path π such that $\pi, 0 \models \xi U \psi$.
- $P, u \models A(\xi U \psi)$ if for every u-path π we have $\pi, 0 \models \xi U \psi$.
- $P, u \models E(\xi R \psi)$ if there exist a u-path π such that $\pi, 0 \models \xi R \psi$.
- $P, u \models A(\xi R \psi)$ if for every u-path π we have $\pi, 0 \models \xi R \psi$.

For example, the formula $AG(\mathsf{request} \rightarrow EF\mathsf{grant})$ says that whenever a request is made it is eventually granted in some possible future. We say that p satisfies φ, denoted $P \models \varphi$, if $P, w^0 \models \varphi$.

A program $P = (W, w^0, R, V)$ is a *tree program* if (W, R) is a tree and w^0 is its root. Note that in this case P is a leafless 2^{Prop}-labeled tree (it is leafless, since R is total). P is a $\mathcal{D}$-tree program, for $\mathcal{D} \subset N$, if (W, R) is a $\mathcal{D}$-tree. It turns out that the tree programs satisfying a given formula are exactly those accepted by some finite tree automaton. The following theorem establishes a very simple translation between CTL and weak alternating Büchi tree automata.

THEOREM 13. [42, 53] *Given a CTL formula φ and a finite set $\mathcal{D} \subset N$, one can build a limited-alternation WAA $A_\varphi = (\Sigma, \mathcal{D}, S, s^0, \rho, F)$, where $\Sigma = 2^{Prop}$ and $|S|$ is in $O(|\varphi|)$, such that $T_\omega(A_\varphi)$ is exactly the set of $\mathcal{D}$-tree programs satisfying φ.*

Proof. The set S of states consists of all subformulas of φ. The initial state s^0 is φ itself. The set F of accepting states consists of all formulas in S of the form $E(\xi R \psi)$ and $A(\xi R \psi)$. It remains to define the transition function ρ. In the following definition we use the notion of dual, defined in the proof of Theorem 11.

- $\rho(p, a, k) = \mathbf{true}$ if $p \in a$.
- $\rho(\neg p, a, k) = \mathbf{true}$ if $p \notin a$.
- $\rho(p, a, k) = \mathbf{false}$ if $p \notin a$.
- $\rho(\neg p, a, k) = \mathbf{false}$ if $p \in a$.
- $\rho(\xi \wedge \psi, a, k) = \rho(\xi, a, k) \wedge \rho(\psi, a, k)$.

- $\rho(\xi \vee \psi, a, k) = \rho(\xi, a, k) \vee \rho(\psi, a, k)$.
- $\rho(EX\psi, a, k) = \bigvee_{c=0}^{k-1}(c, \psi)$.
- $\rho(AX\psi, a, k) = \bigwedge_{c=0}^{k-1}(c, \psi)$.
- $\rho(E(\xi U\psi), a, k) = \rho(\psi, a, k) \vee (\rho(\xi, a, k) \wedge \bigvee_{c=0}^{k-1}(c, E(\xi U\psi)))$.
- $\rho(A(\xi U\psi), a, k) = \rho(\psi, a, k) \vee (\rho(\xi, a, k) \wedge \bigwedge_{c=0}^{k-1}(c, A(\xi U\psi)))$.
- $\rho(E(\xi R\psi), a, k) = \rho(\psi, a, k) \wedge (\rho(\xi, a, k) \vee \bigvee_{c=0}^{k-1}(c, E(\xi R\psi)))$.
- $\rho(A(\xi U\psi), a, k) = \rho(\psi, a, k) \wedge (\rho(\xi, a, k) \vee \bigwedge_{c=0}^{k-1}(c, A(\xi U\psi)))$.

Finally, we define a partition of S into disjoint sets and a partial order over the sets. Each formula $\psi \in S$ constitutes a (singleton) set $\{\psi\}$ in the partition. The partial order is then defined by $\{\xi\} \leq \{\psi\}$ iff ξ a subformula of ψ. Here, all sets are transient, expect for sets of the form $\{E(\xi U\psi)\}$ and $\{\neg A(\xi U\psi)\}$, which are existential, and sets of the form $\{A(\xi U\psi)\}$ and $\{\neg E(\xi U\psi)\}$, which are universal. Thus, A_φ is a limited-alternation WAA. ∎

While temporal logic was introduced in both a branching-time setting and linear-time setting [59], its introduction to computer science was in a linear-time setting [57], to be followed soon by a branching-time setting [14]. The debate in the computer-science literature regarding the relative merits of the linear-time and branching-time goes back to the early 1980s [3, 13, 23, 25, 28, 43, 58, 76, 78]. For a description of earlier discussions of this distinction in the philosophical-logic literature, see [9]. For a more recent account, see [79].

4 MODEL CHECKING

In this section we focus on model checking finite-state programs. Our computational-complexity results are stated in terms of the size of the programs and the temporal properties being checked. The size $|\varphi|$ of a temporal formula φ is simply its length (as a character string). The size of a finite-state program $P = (W, w^0, R, V)$ is the size of its encoding, which is proportional to the number of states in W and the number of transitions in R. Of course, the size of P can be rather large. For example, if P is a computer circuit with n memory bits, then there are 2^n possible states. In general, the number of states of a program is at least exponential in the size of its description by means of a programing language or a hardware-description languages. This blow-up is referred to as the *state-explosion problem*. Much of the research on model checking is focused on dealing with the state-explosion problem, including on-the-fly search techniques, which search through the state space in a demand-driven fashion [36], and symbolic techniques, which represent large state spaces compactly [4]. See [16, 37] for extensive discussions.

4.1 Linear Temporal Logic

We assume that we are given a finite-state program and an LTL formula that specifies the legal computations of the program. The problem is to check whether all computations of the program are legal.

Let $\mathbf{u} = w_0, w_1 \ldots$ be a w_0-path of a finite-state program $P = (W, w^0, R, V)$. The sequence $V(w_0), V(w_1) \ldots$ is a *computation of* P (note that such a sequence can indeed be viewed as a function $\pi_{\mathbf{u}} : \omega \to 2^{Prop}$, which is how we described computations earlier). We say that P *satisfies* an LTL formula φ if *all* computations of P satisfy φ. The *LTL verification problem* is to check whether P satisfies φ.

We now describe the automata-theoretic approach to the LTL verification problem. A finite-state program $P = (W, w^0, R, V)$ can be viewed as a nondeterministic Büchi automaton $A_P = (\Sigma, W, \{w^0\}, \rho, W)$, where $\Sigma = 2^{Prop}$ and $v \in \rho(u, a)$ iff uRv holds and $a = V(u)$. As this automaton has a set of accepting states equal to the whole set of states, any infinite run of the automaton is accepting. Thus, $L_\omega(A_P)$ is the set of computations of P.

Hence, for a finite-state program P and an LTL formula φ, the verification problem is to verify that all infinite words accepted by the automaton A_P satisfy the formula φ. By Corollary 12, we know that we can build a nondeterministic Büchi automaton A_φ that accepts exactly the computations satisfying the formula φ. The verification problem thus reduces to the automata-theoretic problem of checking that all computations accepted by the automaton A_P are also accepted by the automaton A_φ, that is $L_\omega(A_P) \subseteq L_\omega(A_\varphi)$. Equivalently, we need to check that the automaton that accepts $L_\omega(A_P) \cap L_\omega(\overline{A_\varphi})$ is empty, where $L_\omega(\overline{A_\varphi}) = \overline{L_\omega(A_\varphi)} = \Sigma^\omega - L_\omega(A_\varphi)$.

First, note that, by Corollary 12, $L_\omega(\overline{A_\varphi}) = L_\omega(A_{\neg\varphi})$ and the automaton $A_{\neg\varphi}$ has $2^{O(|\varphi|)}$ states. (A straightforward approach, starting with the automaton A_φ and then complementing it, would result in a doubly exponential blow-up, since complementation of nondeterministic Büchi automata is exponential [41, 50]). To get the intersection of the two automata, we use Proposition 1. Consequently, we can build an automaton for $L_\omega(A_P) \cap L_\omega(A_{\neg\varphi})$ having $|W| \cdot 2^{O(|\varphi|)}$ states. We need to check this automaton for emptiness. Using Proposition 2, we get the following results.

THEOREM 14. [44, 67, 81] *Checking whether a finite-state program P satisfies an LTL formula φ can be done in time $O(|P| \cdot 2^{O(|\varphi|)})$ or in space $O((|\varphi| + \log |P|)^2)$.*

We note that a time upper bound that is polynomial in the size of the program and exponential in the size of the specification is considered here to be reasonable, since the specification is usually rather short [44]. For practical verification algorithms that are based on the automata-theoretic approach see [8, 18].

4.2 Branching Temporal Logic

For linear temporal logic, each program may correspond to infinitely many computations. Model checking is thus reduced to checking inclusion between the set of computations allowed by the program and the language of an automaton describing the formula. For branching temporal logic, each program corresponds to a single "computation tree". On that account, model checking is reduced to checking acceptance of this computation tree by the automaton describing the formula.

A program $P = (W, w^0, R, V)$ can be viewed as a W-labeled tree $\langle \tau_P, \mathcal{T}_P \rangle$ that corresponds to the unwinding of P from w^0. For every node $w \in W$, let $arity(w)$ denote the number of R-successors of w and let $succ_R(w) = \langle w_1, \ldots, w_{arity(w)} \rangle$ be an ordered list of w's R-successors (we assume that the nodes of W are ordered). τ_P and $\mathcal{T}_P$ are defined inductively:

1. $\varepsilon \in \tau_P$ and $\mathcal{T}_P(\varepsilon) = w^0$.

2. For $y \in \tau_P$ with $succ_R(\mathcal{T}_P(y)) = \langle w_1, \ldots, w_k \rangle$ and for all $1 \leq i \leq k$, we have $y \cdot i \in \tau_P$ and $\mathcal{T}_P(y \cdot i) = w_i$.

Let $\mathcal{D}$ be the set of arities of states of P, i.e., $\mathcal{D} = \{arity(w) \ : \ w \in W\}$. Clearly, τ_P is a $\mathcal{D}$-tree. If P is finite, then $\mathcal{D}$ is finite.

Let $\langle \tau_P, V \cdot \mathcal{T}_P \rangle$ be the 2^{Prop}-labeled $\mathcal{D}$-tree defined by $V \cdot \mathcal{T}_P(y) = V(\mathcal{T}_P(y))$ for $y \in \tau_P$. Let φ be a CTL formula. Suppose that $A_{\mathcal{D},\varphi}$ is an alternating automaton that accepts exactly all $\mathcal{D}$-tree programs that satisfy φ. It can easily be shown that $\langle \tau_P, V \cdot \mathcal{T}_P \rangle$ is accepted by $A_{\mathcal{D},\varphi}$ iff $P \models \varphi$. We now show that by taking the product of P and $A_{\mathcal{D},\varphi}$ we get an alternating Büchi tree automaton on a 1-letter alphabet that is empty iff $\langle \tau_P, V \cdot \mathcal{T}_P \rangle$ is accepted by $A_{\mathcal{D},\varphi}$.

Let $A_{\mathcal{D},\varphi} = (2^{AP}, \mathcal{D}, S, \varphi, \rho, F)$ be a limited-alternation WAA that accepts exactly all $\mathcal{D}$-tree programs that satisfy φ, and let $S_1, \ldots, S_n$ be the partition of S. The *product automaton* of P and $A_{\mathcal{D},\varphi}$ is the limited-alternation WAA $A_{P,\varphi} = (\{a\}, \mathcal{D}, W \times S, \delta, \langle w^0, \varphi \rangle, G)$, where δ and G are defined as follows:

- Let $s \in S$, $w \in W$, $succ_R(w) = \langle w_1, \ldots, w_k \rangle$, and $\rho(s, V(w), k) = \theta$. Then $\delta(\langle w, s \rangle, a, k) = \theta'$, where θ' is obtained from θ by replacing each atom (c, s') in θ by the atom $(c, \langle w_c, s' \rangle)$.

- $G = W \times F$

- $W \times S$ is partitioned to $W \times S_1, W \times S_2, \ldots, W \times S_n$.

- $W \times S_i$ is transient (resp., existential, universal) if S_i is transient (resp., existential, universal), for $1 \leq i \leq n$.

Note that if P has m_1 states and $A_{\mathcal{D},\varphi}$ has m_2 states then $A_{P,\varphi}$ has $O(m_1 m_2)$ states.

PROPOSITION 15. *$A_{P,\varphi}$ is nonempty if and only if $P \models \varphi$.*

We can now put together Propositions 9, 10, and 15 to get a model-checking algorithm for CTL.

THEOREM 16. [15, 42] *Checking whether a finite-state program P satisfies a CTL formula φcan be done in time $O(|P| \cdot |\varphi|)$ or in space $O(|\varphi| \log^2 |P|)$.*

See also [8, 84] for description of practical algorithms. For an extension of automata-theoretic branching-time model checking to more expressive branching-time logics, such as the branching-time logic CTL*, which merges CTL and LTL and is more expressive than both [25], or the modal fixpoint logic, which is more expressive than CTL* [39], see [42].

5 VALIDITY CHECKING

5.1 Linear Temporal Logic

We are given an LTL formula φ. We say that φ is *valid* iff it is true in all computations. By Corollary 12, we know that we can build a nondeterministic Büchi automaton A_φ that accepts exactly the computations in which φ is true. In other words, φ is valid iff $L_\omega(A_\varphi) = \Sigma^\omega$, where $\Sigma = 2^{Prop}$, which holds iff $\Sigma^\omega - L_\omega(A_\varphi) = \emptyset$. Since $\Sigma^\omega - L_\omega(A_\varphi) = L_\omega(A_{\neg\varphi})$, we have that φ is valid iff $L_\omega(A_{\neg\varphi}) = \emptyset$. Thus, validity checking is been reduced to emptiness checking. We can now combine Proposition 2 with Corollary 12:

THEOREM 17. [67] *Checking whether an LTL formula φ is valid can be done in time $O(2^{O(|\varphi|)})$ or in space $O((|\varphi|)^2)$.*

We note that the upper space bound of Theorem 17 is essentially optimal, since the validity problem for LTL is PSPACE-hard [67].

5.2 Branching Temporal Logic

We are given a CTL formula φ. we say that φ is *valid* iff it is true in all programs. For LTL, Theorems 11 and 12 provided automata-theoretic characterizations of all models of the formula. This is not the case for CTL, as Theorem 13 provides only a characterization of tree models. Fortunately, this suffices for validity checking due to the following proposition.

PROPOSITION 18. [22] *Let φ be a CTL formula. Then φ is valid iff φ is true in all $|\varphi|$-tree programs.*

Let A_φ be the automaton $A_\varphi^{\{|\varphi|\}}$, i.e., it is the automaton $A_\varphi^{\mathcal{D}}$ of Theorem 13, with $\mathcal{D} = \{|\varphi|\}$. It follows from Proposition 18 that a CTL formula is valid iff $T_\omega(A_{\neg\varphi}) = \emptyset$. Combining this with Proposition 7, we get:

THEOREM 19. [24] *Checking whether a CTL formula φ is valid can be done in time $O(2^{O(|\varphi|)})$.*

We note that the upper time bound of Theorem 19 is essentially optimal, since the validity problem for CTL is EXPTIME-hard [31]. For a practical algorithm to decide CTL validity, see [49]. For extension of automata-theoretic validity checking to expressive modal logics, see [40, 64, 77].

ACKNOWLEDGEMENTS

Work supported in part by NSF grants CCR-9988322, CCR-0124077, CCR-0311326, IIS-9908435, IIS-9978135, EIA-0086264, and ANI-0216467, by BSF grant 9800096, by Texas ATP grant 003604-0058-2003, and by a grant from the Intel Corporation. Part of this work was done while the author was visiting the Isaac Newton Institute for Mathematical Science, Cambridge, UK, as part of a Special Programme on Logic and Algorithms.

BIBLIOGRAPHY

[1] R. Armoni, D. Bustan, O. Kupferman, and M.Y. Vardi. Resets vs. aborts in linear temporal logic. In *Proc. 9th International Conference on Tools and algorithms for the construction and analysis of systems*, number 2619 in Lecture Notes in Computer Science, pages 65 – 80. Springer-Verlag, 2003.

[2] R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M.Y. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification logic. In *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *Lecture Notes in Computer Science*, pages 296–211, Grenoble, France, April 2002. Springer-Verlag.
[3] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.
[4] R.E. Bryant. Graph-based algorithms for boolean-function manipulation. *IEEE Trans. on Computers*, C-35(8), 1986.
[5] J.A. Brzozowski and E. Leiss. Finite automata and sequential networks. *Theoretical Computer Science*, 10:19–35, 1980.
[6] J.R. Büchi. Weak second-order arithmetic and finite automata. *Zeit. Math. Logik und Grundl. Math.*, 6:66–92, 1960.
[7] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. International Congress on Logic, Method, and Philosophy of Science. 1960*, pages 1–12, Stanford, 1962. Stanford University Press.
[8] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.
[9] J. P. Burgess. Logic and time. *J. Symbolic Logic*, 44:566–582, 1979.
[10] D. Bustan, A. Flaisher, O. Grumberg, O. Kupferman, and M.Y. Vardi. Regular vacuity. In *Proc. 13th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, volume 3725 of *Lecture Notes in Computer Science*, pages 191–206. Springer-Verlag, 2005.
[11] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, January 1981.
[12] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *Journal of Computer and System Sciences*, 8:117–141, 1974.
[13] E.M. Clarke and I.A. Draghicescu. Expressibility results for linear-time and branching-time logics. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Proc. Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 428–437. Springer-Verlag, 1988.
[14] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.
[15] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.
[16] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
[17] R. Cleaveland. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. *Formal Methods in System Design*, 2:121–147, 1993.
[18] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.
[19] J. E. Doner. Decidability of the weak second-order theory of two successors. *Notices Amer. Math. Soc.*, 12:819, 1965.
[20] C. Elgot. Decision problems of finite-automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, 1961.
[21] E.A. Emerson. Automata, tableaux, and temporal logics. In *Proc. Workshop on Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 79–87. Springer-Verlag, 1985.
[22] 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.
[23] E.A. Emerson and E.M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. 7th InternationalColloq. on Automata, Languages and Programming*, pages 169–181, 1980.
[24] E.A. Emerson and J.Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30:1–24, 1985.
[25] 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.
[26] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 328–337, White Plains, October 1988.
[27] E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 368–377, San Juan, October 1991.
[28] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. In *Proc. 20th ACM Symp. on Principles of Programming Languages*, pages 84–96, New Orleans, January 1985.

[29] E.A. Emerson and C.-L. Lei. Temporal model checking under generalized fairness constraints. In *Proc. 18th Hawaii International Conference on System Sciences*, North Holywood, 1985. Western Periodicals Company.

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

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

[32] C. Fritz. Concepts of automata construction from LTL. In *Proc. 12th Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning*, Lecture Notes in Computer Science 3835, pages 728–742. Springer-Verlag, 2005.

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

[34] 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, August 1995.

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

[36] G. Holzmann. An improved protocol reachability analysis technique. *Software Practice and Experience*, 18(2):137–161, February 1988.

[37] G.J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.

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

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

[40] O. Kupferman, U. Sattler, and M. Y. Vardi. The complexity of the graded μ-calculus. In *Proceedings of the Conference on Automated Deduction*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 423–437. Springer Verlag, 2002.

[41] O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. on Computational Logic*, 2(2):408–429, July 2001.

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

[43] L. Lamport. Sometimes is sometimes "not never" - on the temporal logic of programs. In *Proc. 7th ACM Symp. on Principles of Programming Languages*, pages 174–185, January 1980.

[44] 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, New Orleans, January 1985.

[45] 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, Brooklyn, June 1985. Springer-Verlag.

[46] M.T. Liu. Protocol engineering. *Advances in Computing*, 29:79–195, 1989.

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

[48] Z. Manna and A. Pnueli. Temporal specification and verification of reactive modules. Technical report, Weizmann Institute, 1992.

[49] W. Marrero. Using BDDs to decide ctl. In *Proc. 11th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 3440, pages 222–236. Springer-verlag, 2005.

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

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

[52] A.W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Computation Theory*, volume 208 of *Lecture Notes in Computer Science*, pages 157–168. Springer-Verlag, 1984.

[53] 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 *Proceedings 3rd IEEE Symp. on Logic in Computer Science*, pages 422–427, Edinburgh, July 1988.

[54] D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th International Colloquium on Automata, Languages and Programming*, volume 226 of *Lecture Notes in Computer Science*, pages 275 – 283. Springer-Verlag, 1986.

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

[56] 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.

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

[58] A. Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In *Proc. 12th International Colloquium on Automata, Languages and Programming*, volume 194, pages 15–32. Lecture Notes in Computer Science, Springer-Verlag, 1985.

[59] A. Prior. *Time and Modality.* Oxford University Press, 1957.

[60] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th International Symp. on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, 1981.

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

[62] 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.

[63] H. Rudin. Network protocols and tools to help produce them. *Annual Review of Computer Science*, 2:291–316, 1987.

[64] U. Sattler and M.Y. Vardi. The hybrid μ-calculus. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proc. 1st Int'l Joint Conf. on Automated Reasoning*, Lecture Notes in Computer Science 2083, pages 76–91. Springer-Verlag, 2001.

[65] W.J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *Journal on Computer and System Sciences*, 4:177–192, 1970.

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

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

[68] 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.

[69] R.S. Street and E.A. Emerson. An elementary decision procedure for the μ-calculus. In *Proc. 11th International Colloquium on Automata, Languages and Programming*, volume 172, pages 465–472. Lecture Notes in Computer Science, Springer-Verlag, July 1984.

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

[71] R.E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2):146–160, 1972.

[72] 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.

[73] 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.

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

[75] 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-Verlag, Berlin, 1995.

[76] M.Y. Vardi. Linear vs. branching time: A complexity-theoretic perspective. In *Proc. 13th IEEE Sym.. on Logic in Computer Science*, pages 394–405, 1998.

[77] M.Y. Vardi. Reasoning about the past with two-way automata. In *Proc. 25th International Coll. on Automata, Languages, and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer-Verlag, Berlin, July 1998.

[78] M.Y. Vardi. Sometimes and not never re-revisited: on branching vs. linear time. In D. Sangiorgi and R. de Simone, editors, *Proc. 9th Int'l Conf. on Concurrency Theory*, Lecture Notes in Computer Science 1466, pages 1–17, 1998.

[79] M.Y. Vardi. Branching vs. linear time: Final showdown. In *Proc. 7th International Conference on Tools and algorithms for the construction and analysis of systems*, volume 2031 of *Lecture Notes in Computer Science*, pages 1–22. Springer-Verlag, 2001.

[80] 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-Verlag, 1984.

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

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

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

[84] W. Visser and H. Barringer. Practical CTL* model checking: Should SPIN be extended? *International Journal on Software Tools for Technology Transfer*, 2(4):350–365, 2000.

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

INDEX