Alternating Automata: Checking Truth and Validity for Temporal Logics

Moshe Y. Vardi[⋆]

Rice University
Department of Computer Science
Houston, TX 77005-1892, U.S.A.
Email: vardi@cs.rice.edu
URL: http://www.cs.rice.edu/~vardi

Abstract. We describe an automata-theoretic approach to the automated checking of truth and validity for temporal logics. The basic idea underlying this approach is that for any formula we can construct an alternating automaton that accepts precisely the models of 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 truth and validity checking and yield clean and essentially optimal algorithms for both problems.

1 Introduction

CADE is the major forum for presentation of research in all aspects of automated deduction. Essentially, the focus of CADE is on checking the *validity* of logical formulas. Underlying the notion of logical validity, however, is the notion of logical *truth*. In many computer science applications, the focus is on the checking of logical truth rather than of logical validity. This is certainly the case in database query evaluation (see [Var82]) and in finite-state program verification (see [CES86]). (In fact, we have argued elsewhere that even applications that traditionally focus on logical validity, such as knowledge representation, might be better off focusing on logical truth [HV91].)

In general, the algorithmic techniques in computer-aided validity analysis, i.e., *validity checking*, and in computer-aided truth analysis, i.e., *truth checking*, seem to do very little with each other, in spite of the obvious relationship between truth and validity. Our goal in this paper is to show that for temporal logics it is possible to unify the algorithmic techniques underlying validity and truth checking. We will argue that *alternating automata* provide such a unifying algorithmic tool. (This tool is also applicable to *dynamic logics* [FL79] and *description logics* [GL94], but because of space constraints we cannot cover these logics in this paper.)

Temporal logics, which are logics geared towards the description of the temporal ordering of events, have been adopted as a powerful tool for specifying and verifying concurrent programs [Pnu77, MP92]. One of the most significant developments in this

[⋆] Supported in part by the NSF grant CCR-9628400.

area is the discovery of algorithmic methods for verifying temporal logic properties of *finite-state* programs [CES86, LP85, QS81]. This derives its significance from the fact that many synchronization and communication protocols can be modeled as finite-state programs [Liu89, Rud87]. 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 propositional temporal logic formula that specifies that behavior is true in the program, modeled as a finite Kripke structure; in other words, the program has to be a model of the formula. Hence the name *model checking* for the verification methods derived from this viewpoint (see [CG87, Wol89, CGL93]), though we prefer to use the term *truth checking* in this paper. Note that the formula that specifies the desired behavior clearly should be neither valid nor unsatisfiable, which entails that a computer-aided verification system has to have the capacity for *validity checking* in addition to truth checking.

We distinguish between two types of temporal logics: linear and branching [Lam80]. 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 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 exactly all the computations in which the formula is true. For linear temporal logic the structures are infinite words [WVS83, Sis83, LPZ85, Pei85, SVW87, VW94], while for branching temporal logic the structures are infinite trees [ES84, SE84, Eme85, EJ88, VW86b]. This enables the reduction of temporal logic decision problems, both truth and validity checking, to known automata-theoretic problems.

Initially, the translations in the literature from temporal logic formulas to automata used *nondeterministic* automata (cf. [VW86b, VW94]). These translations have two disadvantages. First, the translation itself is rather nontrivial; indeed, in [VW86b, VW94] 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 truth checking, which in many cases can be done in linear running time [CES86, QS81, Cle93],

Recently it has been shown that if one uses *alternating* automata rather than *nondeterministic* automata, then these problems can be solved [Var94, BVW94]. 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 paper we show 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 truth- and validity-checking 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 [VW86b, VW94], the translation to alternating automata is linear [MSS88, EJ91, Var94, BVW94]. Thus, the advantage of alternating automata is that they enable one to decouple the logic from the algorithmics. The translations from formulas to automata handle the logic, and the algorithms are then applied to the automata.

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 $x \in \tau$. If $\mathcal{D}$ is a singleton set $\{k\}$ then we say that τ is *uniform* and we refer to $\mathcal{D}$-trees as k-trees. A tree is called *leafless* if every node has at least one child. For example, an infinite word is a leafless 1-tree.

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 word automaton* A is a tuple $(\Sigma, S, s^0, \rho, F)$, where Σ is a finite nonempty *alphabet*, S is a finite nonempty set of *states*, $s^0 \in S$ is an *initial* state, $F \subseteq S$ 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 $s_0 = s^0$ 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. [Cho74] *Let A_1 and A_2 be nondeterministic word Büchi automata with n_1 and n_2 states, respectively. Then there is a Büchi word 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 is easy.

Proposition 2.

1. [EL85b, EL85a] *The nonemptiness problem for nondeterministic Büchi word automata is decidable in linear time.*
2. [VW94] *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 [CKS81] in the context of Turing machines and in [BL80, CKS81] for finite automata. The alternation formalisms in [BL80] and [CKS81] are different, though equivalent. We follow here the formalism of [BL80], which was extended in [MS87] 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 word 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 partial transition function.

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 $\rho(s, a_i) = \mathbf{true}$, then x does not need to have any children. On the other hand, we cannot have $\rho(s, a_i) =$ **false** since **false** is not satisfiable, and we cannot have $\rho(s, a_i)$ be undefined. The run r is *accepting* if every infinite branch in r includes infinitely many labels in F. Thus, a branch in an accepting run has to hit the **true** transition or hit accepting states infinitely often.

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. [MH84] *Let A be an alternating Büchi word automaton with n states. Then there is a nondeterministic Büchi word automaton A_n with $2^{O(n)}$ states such that $L_\omega(A_n) = L_\omega(A)$.*

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

Proposition 4. [Var96] *The nonemptiness problem for alternating Büchi word automata is decidable in exponential time or 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, $s^0 \in S$ is an initial state, $F \subseteq S$ is a set of accepting states, and $\rho : S \times \Sigma \times \mathcal{D} \to 2^{S^*}$ 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 the initial state and the transitions obey the transition function ρ; that is, $r(\varepsilon) = s^0$, 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 $T_\omega(A)$.

Proposition 5. [Rab70, VW86b] *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 $N^* \times S$-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 $\Sigma_r = N^* \times S$ 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 $(1, q_1)$.

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

Proposition 6. [MS95] *Let A be an alternating Büchi automaton with n states. Then there is a nondeterministic Büchi automaton A_n with $2^{O(n)}$ 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. [Var96] *The nonemptiness problem for alternating Büchi tree automata is decidable in exponential time.*

The nonemptiness problem for nondeterministic tree automata is reducible to the 1-letter nonemptiness problem for them, i.e., the nonemptiness problem for trees labeled by a 1-letter alphabet, say $\{a\}$. 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 $\rho'(s, a, k) = \bigcup_{b \in \Sigma} \rho(s, b, k)$. 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 $\rho'(s, a, k) = \bigvee_{b \in \Sigma} \rho(s, b, k)$. 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 explains the difference in the complexities in Propositions 5 and 7.

The problem of coordinating between different copies of the automaton does not occur when we consider uniform, leafless trees labeled by a singleton alphabet. There, it is guaranteed that all copies proceed according to the same (single) labeling. In fact, there is no difference between a run of the automaton on a uniform, leafless tree labeled by a singleton alphabet and a run on an infinite word over a singleton alphabet. It turns out that nonemptiness for alternating word automata over a 1-letter alphabet is easier than the general nonemptiness problem. Actually, it is as easy as the nonemptiness problem for nondeterministic Büchi tree automata (Proposition 5).

Proposition 8. [BVW94] *The nonemptiness problem for alternating Büchi word automata over a 1-letter alphabet is decidable in quadratic time.*

As we shall see later, the alternating automata in our applications have a special structure, studied first in [MSS86]. 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 words over a 1-letter alphabet is easier than the nonemptiness problem for alternating Büchi word automata over a 1-letter alphabet.

Proposition 9. [BVW94] *The nonemptiness problem for weak alternating word automata over a 1-letter alphabet is decidable in linear time.*

As we will 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 *hesitant* alternating automata (or HAA, for short).

Proposition 10. [BVW94] *The nonemptiness problem for hesitant alternating word automata of size n and depth m over a 1-letter alphabet can be solved in time $O(n)$ or in space $O(m \log^2 n)$.*

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 connective U (until) [Eme90a]. LTL is interpreted over *computations*. A computation is a function $\pi : \omega \rightarrow 2^{Prop}$, which assigns truth values to the elements of $Prop$ at each time instant (natural number). An LTL formula φ is *true* in a computation π and a point $i \in \omega$, 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 $\pi, k \models \xi$.

We say that a formula φ is true in a computation π, denoted $\pi \models \varphi$, iff $\pi, 0 \models \varphi$.

We now define the semantics of LTL 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, and $V : W \rightarrow 2^{Prop}$ 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, location counter, etc.), R describes all the possible transitions between states (allowing for nondeterminism), and V relates the states to the propositions. 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.

Let $\mathbf{u} = u_0, u_1 \ldots$ be a path of a finite-state program $P = (W, w^0, R, V)$ such that $u_0 = w^0$. The sequence $V(u_0), V(u_1) \ldots$ is a *computation of* P. We say that the LTL formula φ is *true* in P if φ is true in if *all* computations of P. We say that φ is *valid* if it is true in all programs. It is easy to see that φ is valid iff it is true in all computatins.

Computations can also be viewed as infinite words over the alphabet 2^{Prop}. It turns out that the computations in which a given formula is true 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. [MSS88, Var94] *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 in which the formula φ is true.*

Proof: The set S of states consists of all subformulas of φ and their negation (we identify the formula $\neg\neg\psi$ with ψ). The initial state s^0 is φ itself. The set F of accepting states consists of all formulas in S of the form $\neg(\xi U \psi)$. It remains to define the transition function ρ.

The *dual* $\overline{\theta}$ of a formula is obtained from θ by switching $\vee$ and $\wedge$, by switching **true** and **false**, and, in addition, by negating subformulas in S, e.g., $\overline{\neg p \vee (q \wedge Xq)}$ is $p \wedge (\neg q \vee \neg Xq)$. More formally,

- $\overline{\xi} = \neg\xi$, for $\xi \in S$,
- $\overline{\textbf{true}} = \textbf{false}$,
- $\overline{\textbf{false}} = \textbf{true}$,
- $\overline{(\alpha \wedge \beta)} = (\overline{\alpha} \vee \overline{\beta})$, and
- $\overline{(\alpha \vee \beta)} = (\overline{\alpha} \wedge \overline{\beta})$.

We can now define ρ:

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

∎

By applying Proposition 3, we now get:

Corollary 12. [VW94] *Given an LTL formula φ, one can build a 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 in which the formula φ is true.*

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 [Eme90a]. The path quantifiers are A ("for all paths") and E ("for some path"). The linear-time connectives are X ("next time") and U ("until"). Thus, given a set $Prop$ of atomic propositions, a CTL formula is one of the following:

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

The semantics of CTL is defined with respect to programs. A CTL formula φ is *true* in a program $P = (W, w^0, R, V)$ and a state $u \in W$, denoted $P, u \models \varphi$, if following conditions hold:

- $P, u \models p$ for $p \in Prop$ if $p \in V(u)$.
- $P, u \models \neg\psi$ if $P, u \not\models \psi$.
- $P, u \models \xi \wedge \psi$ iff $P, u \models \xi$ and $P, u \models \psi$.
- $P, u \models EX\psi$ if $P, v \models \psi$ for some v such that uRv holds.
- $P, u \models AX\psi$ if $P, v \models \psi$ for all v such that uRv holds.
- $P, u \models E(\xi U \psi)$ if there exist a path $\mathbf{u} = u_0, u_1, \ldots$, with $u_0 = u$, and some $i \geq 0$, such that $P, u_i \models \psi$ and for all j, $0 \leq j < i$, we have $P, u_j \models \xi$.
- $P, u \models A(\xi U \psi)$ if for all paths $\mathbf{u} = u_0, u_1, \ldots$, with $u_0 = u$, there is some $i \geq 0$ such that $P, u_i \models \psi$ and for all j, $0 \leq j < i$, we have $P, u_j \models \xi$.

We say that φ is *true* in P, denoted $P \models \varphi$, if $P, w^0 \models \varphi$. We say that φ is *valid* if φ is true in all programs P.

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 in which a given formula is true 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. [MSS88, BVW94] *Given a CTL formula φ and a finite set $\mathcal{D} \subset N$, one can build an HAA $A_\varphi^{\mathcal{D}} = (\Sigma, \mathcal{D}, S, s^0, \rho, F)$, where $\Sigma = 2^{Prop}$ and $|S|$ is in $O(|\varphi|)$, such that $T_\omega(A_\varphi^{\mathcal{D}})$ is exactly the set of $\mathcal{D}$-tree programs in which φ is true.*

Proof: The set S of states consists of all subformulas of φ and their negation (we identify the formula $\neg\neg\psi$ with ψ). The initial state s^0 is φ itself. The set F of accepting states consists of all formulas in S of the form $\neg E(\xi U \psi)$ and $\neg A(\xi U \psi)$. 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.

- $\rho(p, a, k) =$ **true** if $p \in a$.
- $\rho(p, a, k) =$ **false** if $p \notin a$.
- $\rho(\neg\psi, a) = \overline{\rho(\psi, a)}$,
- $\rho(\xi \wedge \psi, a, k) = \rho(\xi, a, k) \wedge \rho(\psi, a, k)$.
- $\rho(EX\psi, a, k) = \bigvee_{c=0}^{k-1}(c, \psi)$.
- $\rho(AX\psi, a, k) = \bigwedge_{c=0}^{k-1}(c, \psi)$.
- $\rho(E(\xi U \psi), a, k) = \rho(\psi, a, k) \vee (\rho(\xi, a, k) \wedge \bigvee_{c=0}^{k-1}(c, E(\xi U \psi)))$.
- $\rho(A(\xi U \psi), a, k) = \rho(\psi, a, k) \vee (\rho(\xi, a, k) \wedge \bigwedge_{c=0}^{k-1}(c, A(\xi U \psi)))$.

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 or the negation of 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. ∎

4 Truth and Validity Checking

4.1 Linear Temporal Logic

We are given an LTL formula φ. Recall 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 14. [SC85] *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 14 is essentially optimal, since the validity problem for LTL is PSPACE-hard [SC85].

We now assume that we are given a finite-state program $P = (W, w^0, R, V)$ and an LTL formula φ, and we want to check that φ is true in P. P 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 truth-checking problem is to verify that φ is true in all infinite words accepted by the automaton A_P. The truth-checking 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 [SVW87, Mic88, Saf88, KV97]). 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 15. [LP85, SC85, VW86a] *Checking whether an LTL formula φ is true in a finite-state program P can be done in time $O(|P| \cdot 2^{O(|\varphi|)})$ or in space $O((|\varphi| + \log |P|)^2)$.*

We note that the upper space bound of Theorem 14 is essentially optimal, since the truth-checking problem for LTL is PSPACE-hard [SC85] for fixed programs and NLOGSPACE-hard for fixed formulas [VW86a]. We also 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 [LP85]. For a practical verification algorithm that is based on the automata-theoretic approach see [CVWY92].

4.2 Branching Temporal Logic

We are given a CTL formula φ. Recall 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 16. [Eme90b] *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 16 that a CTL formula is valid iff $T_\omega(A_{\neg\varphi}) = \emptyset$. Combining this with Proposition 7, we get:

Theorem 17. [EH85] *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 17 is essentially optimal, since the validity problem for CTL is EXPTIME-hard [FL79].

We now consider truth checking for CTL. For LTL, each program may correspond to infinitely many computations. Truth 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 CTL, each program corresponds to a single "computation tree". On that account, truth 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.

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 in which φ is true (per Theorem 13). 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 hesitant alternating word 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 an HAA that accepts exactly all $\mathcal{D}$-tree programs in which φ is true, and let $S_1, \ldots, S_n$ be the partition of S. The *product automaton* of P and $A_{\mathcal{D},\varphi}$ is the HAA

$$A_{P,\varphi} = (\{a\}, W \times S, \delta, \langle w^0, \varphi \rangle, G),$$

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 $\delta(\langle w, s \rangle, a) = \theta'$, 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 18. [BVW94] *$A_{P,\varphi}$ is nonempty if and only if $P \models \varphi$.*

We can now put together Propositions 9, 10, and 18 to get a truth-checking algorithm for CTL.

Theorem 19. [CES86, BVW94] *Checking whether a CTL formula φ is true in a finite-state program P can be done in time $O(|P| \cdot |\varphi|)$ or in space $O(|\varphi| \log^2 |P|)$.*

We note that the upper space bound of Theorem 14 is probably optimal, since the truth-checking problem for CTL is PTIME-hard for fixed programs and NLOGSPACE-hard for fixed formulas [BVW94].

References

[BL80] J.A. Brzozowski and E. Leiss. Finite automata, and sequential networks. *Theoretical Computer Science*, 10:19–35, 1980.

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

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

[CG87] E.M. Clarke and O. Grumberg. Avoiding the state explosion problem in temporal logic model-checking algorithms. In *Proc. 6th ACM Symposium on Principles of Distributed Computing*, pages 294–303, Vancouver, British Columbia, August 1987.

[CGL93] E.M. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In *A Decade of Concurrency – Reflections and Perspectives (Proc. REX School/Symposium)*, volume 803 of *Lecture Notes in Computer Science*, pages 124–175. Springer-Verlag, Berlin, 1993.

[Cho74] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *J. Computer and System Sciences*, 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.

[Cle93] R. Cleaveland. A linear-time model-checking algorithm for the alternation-free modal μ-calculus. *Formal Methods in System Design*, 2:121–147, 1993.

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

[EH85] 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.

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

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

[EL85a] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. In *Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages*, pages 84–96, New Orleans, January 1985.

[EL85b] E.A. Emerson and C.-L. Lei. Temporal model checking under generalized fairness constraints. In *Proc. 18th Hawaii International Conference on System Sciences*, pages 277–288, Hawaii, 1985.

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

[Eme90a] E.A. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, B:997–1072, 1990.

[Eme90b] E.A. Emerson. Temporal and modal logic. *Handbook of theoretical computer science*, pages 997–1072, 1990.

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

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

[GL94] O. Grumberg and D.E. Long. Model checking and modular verification. *ACM Trans. on Programming Languages and Systems*, 16(3):843–871, 1994.

[HV91] J. Y. Halpern and M. Y. Vardi. Model checking vs. theorem proving: a manifesto. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation (Papers in Honor of John McCarthy)*, pages 151–176. Academic Press, San Diego, Calif., 1991.

[KV97] Orna Kupferman and M. Y. Vardi. Weak alternating automata are not so weak. In *Proc. 5th Israeli Symp. on Theory of Computing and Systems*, 1997.

[Lam80] L. Lamport. Sometimes is sometimes “not never” - on the temporal logic of programs. In *Proceedings of the 7th ACM Symposium on Principles of Programming Languages*, pages 174–185, January 1980.

[Liu89] M.T. Liu. Protocol engineering. *Advances in Computing*, 29:79–195, 1989.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages*, pages 97–107, New Orleans, January 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, Brooklyn, 1985. Springer-Verlag, Berlin.

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

[Mic88] M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, Berlin, 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 the theorems by Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1-2):69–108, April 1995.

[MSS86] D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In L. Kott, editor, *Automata, Languages and Programming, Proc. 13th Int. Colloquium (ICALP ’86)*, volume 226 of *Lecture Notes in Computer Science*, pages 275–283. Springer-Verlag, Berlin, 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 *Proceedings 3rd IEEE Symposium on Logic in Computer Science*, pages 422–427, Edinburgh, July 1988.

[Pei85] R. Peikert. ω-regular languages and propositional temporal logic. Technical Report 85-01, ETH, 1985.

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

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Int. Symp. on programming, Proc. 5th Int. Symposium*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, Berlin, 1981.

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

[Rud87] H. Rudin. Network protocols and tools to help produce them. *Annual Review of Computer Science*, 2:291–316, 1987.

[Saf88] S. Safra. On the complexity of omega-automata. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, pages 319–327, White Plains, October 1988.

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

[SE84] R. S. Streett and E. A. Emerson. The propositional mu-calculus is elementary. In J. Paredaens, editor, *Automata, Languages and Programming, Proc. 11th Int. Colloquium (ICALP '84)*, volume 172 of *Lecture Notes in Computer Science*, pages 465–472. Springer-Verlag, Berlin, 1984.

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

[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.

[Var82] M.Y. Vardi. The complexity of relational query languages. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 137–146, San Francisco, 1982.

[Var94] M.Y. Vardi. Nontraditional applications of automata theory. In *Theoretical Aspects of Computer Software, Proc. Int. Symposium (TACS'94)*, volume 789 of *Lecture Notes in Computer Science*, pages 575–597. Springer-Verlag, Berlin, 1994.

[Var96] M.Y. Vardi. Alternating automata and program verification. In *Computer Science Today –Recent Trends and Developments*, Lecture Notes in Computer Science 1000, pages 471–485. Springer-Verlag, 1996.

[VW86a] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, June 1986.

[VW86b] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Science*, 32(2):182–21, April 1986.

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

[Wol89] P. Wolper. On the relation of programs and computations to models of temporal logic. In *Temporal Logic in Specification, Proc.*, volume 398 of *Lecture Notes in Computer Science*, pages 75–123. Springer-Verlag, Berlin, 1989.

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