Robust Satisfaction

Orna Kupferman[1] and Moshe Y. Vardi[2⋆]

[1] Hebrew University, The institute of Computer Science, Jerusalem 91904, Israel
Email: orna@cs.huji.ac.il, URL: http://www.cs.huji.ac.il/~orna
[2] Rice University, Department of Computer Science, Houston, TX 77251-1892, U.S.A.
Email: vardi@cs.rice.edu, URL: http://www.cs.rice.edu/~vardi

Abstract. In order to check whether an open system satisfies a desired property, we need to check the behavior of the system with respect to an arbitrary environment. In the most general setting, the environment is another open system. Given an open system M and a property ψ, we say that M *robustly satisfies* ψ iff for every open system M', which serves as an environment to M, the composition $M \| M'$ satisfies ψ. The problem of *robust model checking* is then to decide, given M and ψ, whether M robustly satisfies ψ. In this paper we study the robust-model-checking problem. We consider systems modeled by nondeterministic Moore machines, and properties specified by branching temporal logic (for linear temporal logic, robust satisfaction coincides with usual satisfaction). We show that the complexity of the problem is EXPTIME-complete for CTL and the μ-calculus, and is 2EXPTIME-complete for CTL*. We partition branching temporal logic formulas into three classes: universal, existential, and mixed formulas. We show that each class has different sensitivity to the robustness requirement. In particular, unless the formula is mixed, robust model checking can ignore nondeterministic environments. In addition, we show that the problem of classifying a CTL formula into these classes is EXPTIME-complete.

1 Introduction

Today's rapid development of complex and safety-critical systems requires reliable verification methods. In formal verification, we verify that a system meets a desired property by checking that a mathematical model of the system satisfies a formal specification that describes the property. We distinguish between two types of systems: *closed* and *open* [HP85]. (Open systems are called *reactive* systems in [HP85].) A closed system is a system whose behavior is completely determined by the state of the system. An open system is a system that interacts with its environment and whose behavior depends on this interaction. Thus, while in a closed system all the nondeterministic choices are internal, and resolved by the system, in an open system there are also external nondeterministic choices, which are resolved by the environment [Hoa85].

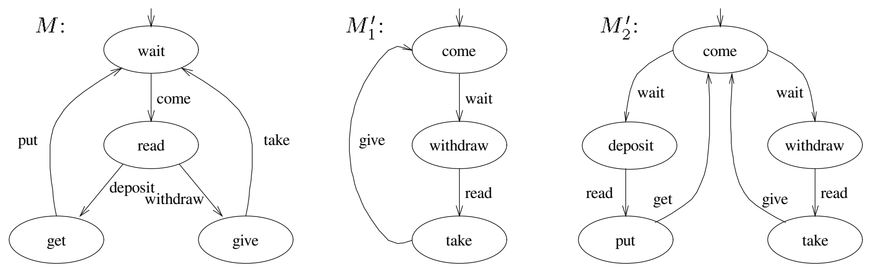
In order to check whether a closed system satisfies a desired property, we translate the system into a formal model, typically a state transition graph, specify the property with a temporal-logic formula, and check formally that the model satisfies the formula.

⋆ Supported in part by the NSF grants CCR-9628400 and CCR-9700061, and by a grant from the Intel Corporation. Part of this work was done when this author was a Varon Visiting Professor at the Weizmann Institute of Science.

Hence the name *model checking* for the verification methods derived from this viewpoint [CE81, QS81]. In order to check whether an open system satisfies a desired property, we need to check the behavior of the system with respect to an arbitrary environment [FZ88]. In the most general setting, the environment is another open system. Thus, given an open system M and a specification ψ, we need to check whether for every (possibly infinite) open system M', which serves as an environment to M, the composition $M||M'$ satisfies ψ. If the answer is yes, we say that M *robustly satisfies* ψ. The problem of *robust model checking*, initially posed in [GL91], is to determine, given M and ψ, whether M robustly satisfies ψ.

Two possible views regarding the nature of time induce two types of temporal logics [Lam80]. In *linear* temporal logics, time is treated as if each moment in time has a unique possible future. Thus, linear temporal logic formulas are interpreted over linear sequences and we regard them as describing a behavior of a single computation of a program. In *branching* temporal logics, each moment in time may split into various possible futures. Accordingly, the structures over which branching temporal logic formulas are interpreted can be viewed as infinite computation trees, each describing the behavior of the possible computations of a nondeterministic program. It turns out that traditional model-checking algorithms and tools are not suitable for the verification of open systems with respect to branching temporal logics [KV96]. In other words, it may be that while M satisfies ψ, it does not robustly satisfy ψ.

To see the difference between robust satisfaction and usual satisfaction, consider the open system M described below. The system M models a cash machine (ATM). At the



state labeled *wait*, M waits for costumers. When a costumer comes, M moves to the state labeled *read*, where it reads whether the costumer wants to *deposit* or *withdraw* money. According to the external choice of the costumer, M moves to either a *get* or *give* state, from which it returns to the *wait* state. An environment for the ATM is an infinite line of costumers, each with his depositing or withdrawing plans. Suppose that we want to check whether the ATM can always get money eventually; thus, whether it satisfies the temporal logic formula $\psi = AGEF get$. Verification algorithms that refer to M as a closed system, perform model checking in order to verify the correctness of the ATM. Since $M \models \psi$, they get a positive answer to this question. Nonetheless, it is easy to see that the ATM does not satisfy the property ψ with respect to all environments. For example, the composition of M with the environment M'_1, in which all the costumers only withdraw money, does not satisfy ψ. Formally, M_1' never supplies to M the input *deposit*, thus M_1' disables the transition of M from the *read* state to the *get* state. Consequently, the composition $M||M_1'$ contains a single computation, in which *get* is not reachable.

A first attempt to solve the robust-model-checking problem was presented in [KV96], which suggested the method of *module checking*. In this algorithmic method we check, given an open system modeled as a finite state-transition graph, and a desired property specified as a temporal-logic formula, whether, no matter how an environment disables some of the system's transitions, the composition of the system with the environment satisfies the property. In particular, in the ATM example, the module-checking paradigm takes into consideration the fact that the environment can consistently disable the transition from the *read* state to the *get* state, and detects the fact that the ATM cannot always get money eventually. The model discussed in [KV96] is somewhat simplistic as it does not allow the system to have internal variables. This assumption is removed in [KV97], which considers module checking with *incomplete information*. In this setting, the system has internal variables, which the environment cannot read. While [KV96] considers arbitrary disabling of transitions, the setting in [KV97] is such that whenever two computations of the system differ only in the values of internal variables along them, the disabling of transitions along them coincide. While the setting in [KV97] is more general, it still does not solve the general robust-model-checking problem.

To see this, let us go back to the ATM example. Suppose that we want to check whether the ATM can either move from all the successors of the initial state to a state where it gets money, or it can move from all the successors of the initial state to a state where it gives money. When we regard M as a closed system, this property is satisfied. Indeed, M satisfies the temporal-logic formula $\varphi = AXEX\,get \vee AXEX\,give$. Moreover, no matter how we remove transitions from the computation tree of M, the trees we get satisfy either $AXEX\,get$ or $AXEX\,give$[3]. In particular, $M||M_1'$ satisfies $AXEX\,give$. Thus, if we follow the module-checking paradigm, the answer to the question is positive. Consider now the environment M_2'. The initial state of $M||M_2'$ has two successors. One of these successors has a single successor in which the ATM gives money and the second has a single successor in which the ATM gets money. Hence, $M||M_2'$ does not satisfy φ. Intuitively, while the module-checking paradigm considers only disabling of transitions, and thus corresponds to the composition of M with all deterministic environments, robust model checking considers all, possibly nondeterministic, environments. There, the composition of the system with an environment may not just disable some of the system's transitions, but may also, as in the example above, increase the nondeterminism of the system.

In this work we consider the problem of verification of open systems in its full generality and solve the robust-model-checking problem. Thus, given an open system M and a specification ψ, we study the problem of determining whether M robustly satisfies ψ. Both M and its environment are nondeterministic Moore machines. They communicate via input and output variables and they both may have private variables and be nondeterministic. Our setting allows the environment to be infinite, and to have unbounded branching degree. Nevertheless, we show that if there is some environment M' for which

[3] We assume that the composition of the system and the environment is *deadlock free*, thus every state has at least one successor.

$M \| M'$ does not satisfy ψ, then there is also a finite environment M'' with a bounded branching degree (which depends on the number of universal requirements in ψ) such that $M \| M''$ does not satisfy ψ.

We solve the robust-model-checking problem for branching temporal specifications. As with module checking with incomplete information, *alternation* is a suitable and helpful automata-theoretic mechanism for coping with the internal variables of M and M'. In spite of the similarity to the incomplete information setting, the solution the robust model-checking problem is more challenging, as one needs to take into consideration the fact that a module may have different reactions to the same input sequence, yet this is possible only when different nondeterministic choices have been taken along the sequence. Using *alternating tree automata*, we show that the problem of robust satisfaction is EXPTIME-complete for CTL and the μ-calculus, and is 2EXPTIME-complete for CTL*. The internal variables of M make the time complexity of the robust-model-checking problem exponential already in the size of M. The same complexity bounds hold for the problem of module checking with incomplete information [KV97]. Thus, on the one hand, the problem of robust model checking, which generalizes the problem of module checking with incomplete information, is not harder than the latter problem. On the other hand, keeping in mind that the system to be checked is typically a parallel composition of several components, which by itself hides an exponential blow-up [HKV97], our results imply that checking verification of open systems with respect to branching temporal specifications is rather intractable.

Recall that not all specification formalisms are sensitive to the distinction between open and closed systems. The study of verification of open system has motivated the use of *universal temporal logic* [GL94] as a specification formalism. Formulas of universal temporal logics describe requirements that should hold in *all* computations of the system. These requirements may be either linear or branching. In both cases, the more behaviors the system has, the harder it is for the system to satisfy the requirements. Indeed, universal temporal logics induce the *simulation* order between systems [Mil71, CGB86]. That is, a system M simulates a system M' if and only if all universal temporal logic formulas that are satisfied in M are satisfied in M' as well. It follows that traditional model-checking methods are applicable also for the verification of open systems with respect to universal properties. Indeed, since M simulates $M \| M'$ for every M', satisfaction of a universal property in M implies its satisfaction in all the compositions of M with an environment.

One of the main advantages of branching temporal logics with respect to linear temporal logic is, however, the ability to mix universal and *existential* properties; e.g., in order to specify possibility properties like $AGEFp$. Existential properties describe requirements that should hold in *some* computations of the system. We show that non-universal properties can be partitioned into two classes, each with a different sensitivity to the distinction between open and closed systems. We say that a temporal-logic formula φ is *existential* if it imposes only existential requirements on the system, thus $\neg\varphi$ is universal. The formula φ is *mixed* if it imposes both existential and universal requirements, thus φ is neither universal nor existential. While universal formulas are insensitive to the system being open, we show that existential formulas are insensitive to the environment being nondeterministic. Thus, for such formulas, one can use the module-checking method. We study the problems of determining whether a given formula is universal or mixed, and show that they are both EXPTIME-complete. These result are relevant also in the contexts of modular verification [GL94] and backwards reasoning [HKQ98].

In the discussion, we compare robust model checking with previous work about verification of open systems as well as with the closely-related area of supervisory control [RW89, Ant95]. We also argue for the generality of the model studied in this paper and show that it captures settings in which assumptions about the environment are known, as well as settings with global actions and possible deadlocks.

2 Preliminaries

2.1 Trees and Automata

Given a finite set Υ, an Υ*-tree* is a set $T \subseteq \Upsilon^*$ such that if $x \cdot \upsilon \in T$, where $x \in \Upsilon^*$ and $\upsilon \in \Upsilon$, then also $x \in T$. When Υ is not important or clear from the context, we call T a tree. The elements of T are called *nodes*, and the empty word ϵ is the *root* of T. For every $x \in T$, the nodes $x \cdot \upsilon \in T$ where $\upsilon \in \Upsilon$ are the *children* of x. Each node $x \neq \epsilon$ of T has a *direction* in Υ. The direction of a node $x \cdot \upsilon$ is υ. We denote by $dir(x)$ the direction of node x. An Υ-tree T is a *full infinite tree* if $T = \Upsilon^*$. Unless otherwise mentioned, we consider here full infinite trees. A *path* η of a tree T is a set $\eta \subseteq T$ such that $\epsilon \in \eta$ and for every $x \in \eta$ there exists a unique $\upsilon \in \Upsilon$ such that $x \cdot \upsilon \in \eta$. The i'th *level* of T is the set of nodes of length i in T. Given two finite sets Υ and Σ, a Σ*-labeled* Υ*-tree* is a pair $\langle T, V\rangle$ where T is an Υ-tree and $V : T \to \Sigma$ maps each node of T to a letter in Σ. When Υ and Σ are not important or clear from the context, we call $\langle T, V\rangle$ a labeled tree.

Alternating tree automata generalize nondeterministic tree automata and were first introduced in [MS87]. An alternating tree automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha\rangle$ runs on full Σ-labeled Υ-trees (for an agreed set Υ of directions). It consists of a finite set Q of states, an initial state $q_0 \in Q$, a transition function δ, and an acceptance condition α (a condition that defines a subset of Q^ω).

For a set Υ of directions, let $\mathcal{B}^+(\Upsilon \times Q)$ be the set of positive Boolean formulas over $\Upsilon \times Q$; i.e., Boolean formulas built from elements in $\Upsilon \times Q$ using $\wedge$ and $\vee$, where we also allow the formulas $\mathbf{true}$ and $\mathbf{false}$ and, as usual, $\wedge$ has precedence over $\vee$. The transition function $\delta : Q \times \Sigma \to \mathcal{B}^+(\Upsilon \times Q)$ maps a state and an input letter to a formula that suggests a new configuration for the automaton. For example, when $\Upsilon = \{0, 1\}$, having

$$\delta(q, \sigma) = (0, q_1) \wedge (0, q_2) \vee (0, q_2) \wedge (1, q_2) \wedge (1, q_3)$$

means that when the automaton is in state q and reads the letter σ, it can either send two copies, in states q_1 and q_2, to direction 0 of the tree, or send a copy in state q_2 to direction 0 and two copies, in states q_2 and q_3, to direction 1. Thus, unlike nondeterministic tree automata, here the transition function may require the automaton to send several copies to the same direction or allow it not to send copies to all directions.

A *run of an alternating automaton* $\mathcal{A}$ on an input Σ-labeled Υ-tree $\langle T, V\rangle$ is a tree $\langle T_r, r\rangle$ in which the root is labeled by q_0 and every other node is labeled by an element of $\Upsilon^* \times Q$. Unlike T, in which each node has exactly $|\Upsilon|$ children, the tree T_r may have nodes with many children and may also have *leaves* (nodes with no children). Thus, $T_r \subset \mathbb{N}^*$ and a path in T_r may be either finite, in which case it contains a leaf, or infinite. Each node of T_r corresponds to a node of T. A node in T_r, labeled by (x, q), describes a copy of the automaton that reads the node x of T and visits the state q. Note that many nodes of T_r can correspond to the same node of T; in contrast, in a run of a nondeterministic automaton on $\langle T, V \rangle$ there is a one-to-one correspondence between the nodes of the run and the nodes of the tree. The labels of a node and its children have to satisfy the transition function. Formally, $\langle T_r, r \rangle$ is a Σ_r-labeled tree where $\Sigma_r = \Upsilon^* \times Q$ and $\langle T_r, r \rangle$ satisfies the following:

1. $\epsilon \in T_r$ and $r(\epsilon) = (\epsilon, q_0)$.
2. Let $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \theta$. Then there is a (possibly empty) set $S = \{(c_0, q_0), (c_1, q_1), \ldots, (c_{n-1}, q_{n-1})\} \subseteq \Upsilon \times Q$, such that:
   - S satisfies θ, and
   - for all $0 \leq i < n$, we have $y \cdot i \in T_r$ and $r(y \cdot i) = (x \cdot c_i, q_i)$.

For example, if $\langle T, V \rangle$ is a $\{0, 1\}$-tree with $V(\epsilon) = a$ and $\delta(q_0, a) = ((0, q_1) \vee (0, q_2)) \wedge ((0, q_3) \vee (1, q_2))$, then the nodes of $\langle T_r, r \rangle$ at level 1 include the label $(0, q_1)$ or $(0, q_2)$, and include the label $(0, q_3)$ or $(1, q_2)$. Note that if $\theta = \mathbf{true}$, then y need not have children. This is the reason why T_r may have leaves. Also, since there exists no set S as required for $\theta = \mathbf{false}$, we cannot have a run that takes a transition with $\theta = \mathbf{false}$.

Each infinite path ρ in $\langle T_r, r \rangle$ is labeled by a word $r(\rho)$ in Q^ω. Let $\mathit{inf}(\rho)$ denote the set of states in Q that appear in $r(\rho)$ infinitely often. A run $\langle T_r, r \rangle$ is accepting iff all its infinite paths satisfy the acceptance condition. In *Büchi* alternating tree automata, $\alpha \subseteq Q$, and an infinite path ρ satisfies α iff $\mathit{inf}(\rho) \cap \alpha \neq \emptyset$. In *Rabin* alternating tree automata, $\alpha \subseteq 2^Q \times 2^Q$, and an infinite path ρ satisfies an acceptance condition $\alpha = \{\langle G_1, B_1 \rangle, \ldots, \langle G_m, B_m \rangle\}$ iff there exists $1 \leq i \leq m$ for which $\mathit{inf}(\rho) \cap G_i \neq \emptyset$ and $\mathit{inf}(\rho) \cap B_i = \emptyset$. As with nondeterministic automata, an automaton accepts a tree iff there exists an accepting run on it. We denote by $\mathcal{L}(\mathcal{A})$ the language of the automaton $\mathcal{A}$; i.e., the set of all labeled trees that $\mathcal{A}$ accepts. We say that an automaton is *nonempty* iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$.

Formulas of branching temporal logic can be translated to alternating tree automata [EJ91, BVW94]. Since the modalities of conventional temporal logics, such as CTL* and the μ-calculus, do not distinguish between the various successors of a node (that is, they impose requirements either on all the successors of the node or on some successor), the alternating automata that one gets by translating formulas to automata are of a special structure, in which whenever a state q is sent to direction υ, the state q is sent to all the directions $\upsilon \in \Upsilon$, in either a disjunctive or conjunctive manner. Formally, following the notations in [GW99], the formulas in $\mathcal{B}^+(\Upsilon \times Q)$ that appear in the transitions of such alternating tree automata are members of $\mathcal{B}^+(\{\Box, \Diamond\} \times Q)$, where $\Box q$ stands for $\bigwedge_{\upsilon \in \Upsilon} (\upsilon, q)$ and $\Diamond q$ stands for $\bigvee_{\upsilon \in \Upsilon} (\upsilon, q)$. As we shall see in Section 3, this structure of the automata is crucial for solving the robust model-checking problem. We say that an alternating tree automaton is *symmetric* if it has the special structure described above.

2.2 Modules

A *module* is a tuple $M = \langle I, O, W, w^{in}, i^{in}, \rho, \pi \rangle$, where I is a finite set of Boolean input variables, O is a finite set of Boolean output variables (we assume that $I \cap O = \emptyset$),

W is a (possibly infinite) set of states, $w^{in} \in W$ is an initial state, $i^{in} \in 2^I$ is an initial input, $\rho : W \times 2^I \to 2^W$ is a nondeterministic transition function, and $\pi : W \to 2^O$ is a labeling function that assigns to each state its output. We require that for all $w \in W$ and $\sigma \in 2^I$, the set $\rho(w, \sigma)$ is not empty. Intuitively, the module can always respond to external inputs, though the response might be to enter a "bad" state. Intuitively, M starts its execution in w^{in}, where it expect the input i^{in}. Whenever M is in state w and the input is $\sigma \subseteq I$, it moves nondeterministically to one of the states in $\rho(w, \sigma)$. A module is *open* if $I \neq \emptyset$. Otherwise, it is *closed*. The *degree* of M is the minimal integer k such that for all w and σ, the set $\rho(w, \sigma)$ contains at most k states. If for all w and σ the set $\rho(w, \sigma)$ contains exactly k states, we say that M is of *exact degree* k.

Let $M_1 = \langle I, O, W_1, w_1^{in}, i_1^{in}, \rho_1, \pi_1 \rangle$ and $M_2 = \langle O, I, W_2, w_2^{in}, i_2^{in}, \rho_2, \pi_2 \rangle$ be two modules such that $\pi_1(w_1^{in}) = i_2^{in}$ and $\pi_2(w_2^{in}) = i_1^{in}$. Note that the inputs of M_1 are the outputs of M_2 and vice versa. The *composition* of M_1 and M_2 is the closed module $M_1 || M_2 = \langle \emptyset, I \cup O, W, w^{in}, \emptyset, \rho, \pi \rangle$, where

- $W = W_1 \times W_2$.
- $w^{in} = \langle w_1^{in}, w_2^{in} \rangle$.
- For every $\langle w_1, w_2 \rangle \in W$, we have $\rho(\langle w_1, w_2 \rangle, \emptyset) = \rho_1(w_1, \pi_2(w_2)) \times \rho_2(w_2, \pi_1(w_1))$.
- For every $\langle w_1, w_2 \rangle \in W$, we have $\pi(\langle w_1, w_2 \rangle) = \pi_1(w_1) \cup \pi_2(w_2)$.

Note that since we assume that for all $w \in W$ and $\sigma \in 2^I$, the set $\rho(w, \sigma)$ is not empty, the composition of M with M' is *deadlock free*, thus every reachable state has at least one successor. Note also that the restriction to M' that closes M does not effect the answer to the robust-model-checking problem. Indeed, if there is some M' such that $M || M'$ is open and does not satisfy ψ, we can easily extend M' so that its composition with M would be closed and would still not satisfy ψ.

Every module $M = \langle I, O, W, w^{in}, i^{in}, \rho, \pi \rangle$ induces an *enabling tree* $\langle T, V \rangle$. The enabling tree of M is a full infinite $\{\top, \bot\}$-labeled $(W \times 2^I)$-tree, thus $T = (W \times 2^I)^*$. We define $dir(\epsilon)$ to be $\langle w^{in}, i^{in} \rangle$, and we label ϵ by $\top$. Intuitively, $\langle T, V \rangle$ indicates which behaviors of M are enabled. Consider a node $x \in T$ such that $dir(x) = \langle w, \sigma \rangle$. For every state $w' \in W$ and input $\sigma' \in 2^I$, we define $V(x.\langle w', \sigma' \rangle)$ as $\top$ if $w' \in \rho(w, \sigma)$, and as $\bot$ otherwise. Consider a node $x = \langle w_1, \sigma_1 \rangle, \langle w_2, \sigma_2 \rangle, \ldots, \langle w_m, \sigma_m \rangle \in T$. By the definition of V, the module M can traverse the computation $w^{in}, w_1, w_2, \ldots, w_m$ when it reads the input sequence $i^{in}, \sigma_1, \sigma_2, \ldots, \sigma_{m-1}$ iff all the prefixes y of x have $V(y) = \top$. Indeed, then and only then we have $w_1 \in \rho(w^{in}, i^{in})$, and $w_{i+1} \in \rho(w_i, \sigma_i)$ for all $1 \leq j \leq m-1$.

Following the definition of a product between two modules, the enabling tree of $M_1 || M_2$ is a $\{\top, \bot\}$-labeled $(W_1 \times W_2)$-tree. Intuitively, M_2 supplies to M_1 its input (and vice versa). Therefore, while the trees of M_1 are $(W_1 \times 2^I)$-trees, reflecting the fact that every state in M_1 may read $2^{|I|}$ different inputs and move to $|W_1|$ successors, the tree of $M_1 || M_2$ is a $(W_1 \times W_2)$-tree, reflecting the fact that every state in $M_1 || M_2$ may have $|W_1| \cdot |W_2|$ successors. Note that M_2 may be nondeterministic. Accordingly, a node associated with a state w of M_1 may have k successors that are labeled $\top$ in the enabling tree of M_1 and have $k' > k$ successors that are labeled $\top$ in the enabling tree of $M_1 || M_2$. That is, M_2 can not only prune transitions of M_1; it can also split transitions of M_1.

Recall that the enabling tree of a module M is a full infinite $\{\top, \bot\}$-labeled $(W \times 2^I)$-tree. As we shall see in Section 3, the fact that the tree is full circumvents some technical difficulties. We now define when M satisfies a formula. For that, we prune from the full tree nodes that correspond to unreachable states of M. Since each state of M has at least one successor, every node in the pruned tree also has at least one successor. Consequently, we are able, in Section 3, to duplicate subtrees and go back to convenient full trees. For an enabling tree $\langle T, V \rangle$, the $\top$-*restriction* of $\langle T, V \rangle$ is the $\{\top\}$-labeled tree with directions in $(W \times 2^I)$ that is obtained from $\langle T, V \rangle$ by pruning subtrees with a root labeled $\bot$. For a module M, the *computation tree* of M is a $2^{I \cup O}$-labeled $(W \times 2^I)$-tree obtained from the $\top$-restriction of M's enabling tree by replacing the $\top$ label of a node with direction $\langle w, \sigma \rangle$ by the label $\pi(w) \cup \sigma$. Note that when M is closed, its computation tree is a W-tree. We say that M satisfies a branching temporal logic formula ψ over $I \cup O$ iff M's computation tree satisfies ψ. The problem of *robust model checking* is to determine, given M and ψ, whether for every M', the composition $M \| M'$ satisfies ψ (we assume that the reader is familiar with branching temporal logic. We refer here to the logics CTL, CTL* , and the μ-calculus [Eme90, Koz83]).

3 Robust Model Checking

In this section we solve the robust-model-checking problem and study its complexity. Thus, given a module M and a branching temporal logic formula ψ, we check whether for every M', the composition $M \| M'$ satisfies ψ. We assume that M has finitely many states and allow M' to have infinitely many states. Nevertheless, we show that if some environment that violates ψ exists, then there exists also a violating environment with finitely many states and a bounded branching degree. For a branching temporal logic formula ψ, we denote by $\mathcal{E}(\psi)$ the number of existential subformulas (subformulas of the form $E\xi$) in ψ. It is known that $\mathcal{E}(\psi)$ bounds the branching degree required in order to satisfy ψ [Eme90]. We now extend this result and show that, also in robust model checking, it suffices to consider environments of degree $\mathcal{E}(\psi)$. For an integer $k \geq 1$, let $[k] = \{1, \ldots, k\}$.

Theorem 1. *Consider a module M and a branching temporal logic formula ψ over $I \cup O$. Let $k = \max\{1, \mathcal{E}(\psi)\}$. If there exists M' such that $M \| M' \models \psi$, then there also exists M' of exact degree k such that $M \| M' \models \psi$.*

Proof (sketch): Assume that $M \| M' \models \psi$ for some M'. Thus, the computation tree $\langle T, V \rangle$ of $M \| M'$ satisfies ψ. In order for that to be true, each node in $\langle T, V \rangle$ has to satisfy a set of subformulas of ψ. Formally, there is a mapping V' of T to sets of subformulas of ψ such that $\psi \in V'(\epsilon)$, and for every $x \in T$, the set $V'(x)$ contains formulas that hold in x, such that the labeling along paths that start at x is enough to "justify" $V'(x)$. For example, if a node x is labeled by EXp, then at least one successor of x is labeled by p. Consider a node x. Some formulas in $V'(x)$ impose on the paths starting at x universal requirements. To satisfy these requirements, x need not have children (yet all the children that x does have, belong to paths that satisfy these universal requirements). In addition, some formulas in $V'(x)$ impose on the paths starting at x at most k existential requirements. Each such requirement needs to be satisfied by some path starting at x, yet it does not have to be satisfied by more than one such path. Also, it may be that the existential requirements impose particular values on the input variables in the successors of x, and different existential requirements may impose the same value. Accordingly, we can prune some of the paths that start at x and satisfy the formulas in $V'(x)$ with not more than k successors of x for each $\sigma \in 2^I$, or by a single successor, in the case $V'(x)$ contains no existential requirements. The pruned tree can therefore be obtained by taking the product of M with a module M'' of degree k where M'' is a suitable pruning of the infinite module obtained by unwinding M'. In order to get a module of exact degree k, we can then duplicate some of the subtrees of M''. (For the μ-calculus, the proof is considerably more complicated and uses techniques from [SE89].) $\square$

In order to understand the difference between Theorem 1 and the classical "bounded-degree property" for branching temporal logic, recall that the theorem refers to the branching degree of the environment, rather than to that of the composition $M||M'$. Consider, for example, a module M with an initial state that has two successors, one labeled p and one labeled $\neg p$. In order for M to satisfy the formula $\psi = EX(p \wedge q) \wedge EX(p \wedge \neg q)$, for an input variable q, a split of the state labeled p is required. Though $\mathcal{E}(\psi) = 2$, such a split results in a composition of branching degree 4. It can, however, be achieved by composing M with an environment M' of branching degree 2. Theorem 1 shows that, though we may sometimes need the branching degree of $M||M'$ to be bigger than $\mathcal{E}(\psi)$, it is sufficient to compose M with an environment of branching degree $\mathcal{E}(\psi)$. We now use Theorem 1 to show that the robust-satisfaction problem for branching temporal logics can be reduced to the emptiness problem for alternating tree automata.

Theorem 2. *Consider a module M and branching temporal logic formula ψ over $I \cup O$. Let $\mathcal{A}_\psi$ be the symmetric alternating tree automaton that corresponds to ψ and let $k = \max\{1, \mathcal{E}(\psi)\}$. There is an alternating tree automaton $\mathcal{A}_{M,\psi}$ over 2^I-labeled $(2^O \times [k])$-trees such that*

1. *$\mathcal{L}(\mathcal{A}_{M,\psi})$ is empty iff M robustly satisfies $\neg\psi$.*
2. *$\mathcal{A}_{M,\psi}$ and $\mathcal{A}_\psi$ have the same acceptance condition.*
3. *The size of $\mathcal{A}_{M,\psi}$ is $O(|M| \cdot |\mathcal{A}_\psi| \cdot k)$.*

Proof (sketch): Before we describe $\mathcal{A}_{M,\psi}$, let us explain the difficulties in the construction and why alternation is so helpful solving them. The automaton $\mathcal{A}_{M,\psi}$ searches for a module M' of exact degree k for which $M||M' \in \mathcal{L}(\mathcal{A}_\psi)$. The modules M and M' interact via the sets I and O of variables. Thus, M' does not know the state in which M is, and it only knows M's output. Accordingly, not all $\{\top, \bot\}$-labeled $(W \times W')$-trees are possible enabling trees of a product $M||M'$. Indeed, $\mathcal{A}_{M,\psi}$ needs to consider only trees in which the behavior of M' is consistent with its incomplete information: if two nodes have the same output history (history according to M''s incomplete information), then either they agree on their label (which can be either $\bot$ or a set of input variables), or the two nodes are outcomes of two different nondeterministic choices that M' has taken along this input history. This consistency condition is non-regular and cannot be checked by an automaton [Tha73]. It is this need, to restrict the set of candidate enabling trees to trees that meet some non-regular condition, that makes robust model checking in the branching paradigm so challenging. The solution is to consider, instead $(W \times W')$-trees, $(2^O \times [k])$-trees. Each node in such a tree may correspond to several nodes in a $(W \times W')$-tree, all with the same output history. Then, alternation is used in order to make sure that while all these nodes agree on their labeling, each of them satisfy requirements that together guarantee the membership in $\mathcal{A}_\psi$.

Let $M = \langle I, O, W, w^{in}, i^{in}, \rho, \pi \rangle$. For $w \in W$, $\sigma \in 2^I$, and $\upsilon \in 2^O$, we define

$$s(w, \sigma, \upsilon) = \{w' \mid w' \in \rho(w, \sigma) \text{ and } \pi(w') = \upsilon\}.$$

That is, $s(w, \sigma, \upsilon)$ contains all the states with output υ that w moves to when it reads σ. The definition of the automaton $\mathcal{A}_{M,\psi}$ can be viewed as an extension of the product alternating tree automaton obtained in the alternating-automata theoretic framework for branching time model checking [BVW94]. There, as we are concerned with model checking, there is a single computation tree with respect to which the formula is checked, and the automaton obtained is a 1-letter automaton. The difficulty here, as we are concerned with robust model checking, is that there are many computation trees to check, so a 1-letter automaton does not suffice. Let $\mathcal{A} = \langle 2^{I \cup O}, Q, q_0, \delta, \alpha \rangle$. We define $\mathcal{A}_{M,\psi} = \langle 2^I, Q', q_0, \delta', \alpha' \rangle$, where

- $Q' = \{q_0\} \cup (W \times Q)$. Intuitively, when the automaton is in state $\langle w, q \rangle$, it accepts all trees that are induced by an environment M' for which the composition with M with initial state w is accepted by $\mathcal{A}$ with initial state q. The initial state q_0 corresponds to the state $\langle w^{in}, q_0 \rangle$, yet it also checks that the first input is i^{in}.
- The transition function $\delta' : Q' \times 2^I \to \mathcal{B}^+((2^O \times [k]) \times Q')$ is defined as follows.
  - For all w, q, and σ, the transition $\delta'(\langle w, q \rangle, \sigma)$ is obtained from $\delta(q, \sigma \cup \pi(w))$ by replacing a conjunction $\Box q'$ by the conjunction

$$\bigwedge_{\upsilon \in 2^O} \bigwedge_{j \in [k]} \bigwedge_{w' \in s(w,\sigma,\upsilon)} (\langle \upsilon, j \rangle, \langle w', q' \rangle),$$

    and replacing a disjunction $\Diamond q'$ by the disjunction

$$\bigvee_{\upsilon \in 2^O} \bigvee_{j \in [k]} \bigvee_{w' \in s(w,\sigma,\upsilon)} (\langle \upsilon, j \rangle, \langle w', q' \rangle).$$

  - For the initial state q_0, we define $\delta'(q_0, i^{in}) = \delta'(\langle w^{in}, \psi \rangle, i^{in})$. For all $\sigma \neq i^{in}$, we define $\delta'(q_0, \sigma) = \mathbf{false}$.

  Consider, for example, a transition from the state $\langle w, q \rangle$. Let $\sigma \in 2^I$ be such that $\delta(q, \sigma \cup \pi(w)) = \Box s \wedge \Diamond t$. The successors of w that are enabled with input σ should satisfy $\Box s \wedge \Diamond t$. Thus, all these successors should satisfy s and at least one successor should satisfy t. The state w may have several successors in $\rho(w, \sigma)$ with the same output $\upsilon \in 2^O$. These successors are indistinguishable by M'. Therefore, if M' behaves differently in such two successors, it is only because M' is in a different state when it interacts with these successors. The number k bounds the number of states in $\rho(w, \sigma)$. Accordingly, M' can exhibit k different behaviors when it interacts with indistinguishable successors of w. For each $j \in [k]$, the automaton sends all the successors of w in $s(w, \sigma, \upsilon)$ to the same direction $\langle \upsilon, j \rangle$, where they are going to

face the same future. Since $\delta(q, \sigma \cup \pi(w)) = \Box s \wedge \Diamond t$, a copy in state s is sent to all the successors, and a copy in state t is sent to some successor. Note that as M is deadlock free, the conjunctions and disjunctions in δ cannot be empty.
- α' is obtained from α by replacing every set participating in α by the set $W \times \alpha$.

□

We now consider the complexity bounds for various branching temporal logics that follow from our algorithm.

Theorem 3. *Robust model checking is*

(1) *EXPTIME-complete for CTL, μ-calculus, and the alternation-free μ-calculus.*
(2) *2EXPTIME-complete for CTL*.*

Proof (sketch): Consider a branching temporal logic formula ψ of length n. Let $\mathcal{A}_\psi$ be the symmetric alternating tree automaton that corresponds to ψ. By [EJ91, BVW94], $\mathcal{A}_\psi$ is a Büchi automaton with $O(n)$ states for ψ in CTL or in the alternation-free μ-calculus, $\mathcal{A}_\psi$ is a parity automaton with $O(n)$ states and d sets in the acceptance condition for ψ in μ-calculus with alternation depth d, and $\mathcal{A}_\psi$ is a Rabin automaton with $2^{O(n)}$ states and 2 pairs in the acceptance condition for ψ in CTL*. In Theorem 2, we reduced the robust-model-checking problem of M with respect to $\neg\psi$ to the problem of checking the nonemptiness of the automaton $\mathcal{A}_{M,\psi}$, which is of size $|M| \cdot |\mathcal{A}_\psi| \cdot \max\{1, \mathcal{E}(\psi)\}$, and which has the same type and size of acceptance condition as $\mathcal{A}_\psi$. The upper bounds then follow from the complexity of the nonemptiness problem for the various automata [MS95, VW86, KV98].

For the lower bounds, one can reduce the satisfiability problem for a branching temporal logic to the robust-model-checking problem for that logic. To see this, note that, by the "bounded-degree property" of branching temporal logic, a search for a satisfying model for ψ can be reduced to a search for a satisfying $2^{I \cup O}$-labeling of a tree with branching degree $\max\{1, \mathcal{E}(\psi)\}$. Then, one can relate the choice of the labels to choices made by the environment. □

The *implementation complexity* of robust model checking is the complexity of the problem in terms of the module, assuming that the specification is fixed. As we discuss in Section 4, there are formulas for which robust model checking coincides with module checking with incomplete information. Since module checking with incomplete information is EXPTIME-hard already for CTL formulas of that type, it follows that the implementation complexity of robust model checking for CTL (and the other, more expressive, logics) is EXPTIME-complete.

In our definition of robust satisfaction, we allow the environment to have infinitely many states. We now claim that finite environments are strong enough. The proof is based on a "finite-model property" of tree automata, proven in [Rab70] for nondeterministic tree automata and extended in [MS95, KV97] to alternating tree automata. As we discuss in Section 5, this result is of great importance in the dual paradigm of supervisory control, where instead of hostile environments we consider collaborative controllers.

Theorem 4. *Given a module M and a branching temporal logic formula ψ, if there is an infinite module M' of degree k such that $M||M'$ satisfies ψ, then there also exists a finite module M'' of degree k such that $M||M''$ satisfies ψ.*

The alternating-automata-theoretic approach to CTL and CTL* model checking is extended in [KV95] to handle Fair-CTL and Fair-CTL* [EL85]. Using the same extension, we can solve the problem of robust model checking also for handle modules augmented with fairness conditions.

4 Universal and Mixed Formulas

The study of verification of open system has motivated the use of universal temporal logic [GL94]. Formally, a formula ψ is *universal* iff for every module M, if M satisfies ψ, then for every M', the composition $M||M'$ also satisfies ψ. By the above definition, M satisfies a universal property ψ iff M robustly satisfies ψ. In this section we show that the set of non-universal properties can be further partitioned into two classes, each with a different sensitivity to the robustness of the satisfaction. In addition, we study the complexity of classifying a CTL formula to its sensitivity class. We say that a CTL formula ψ is *mixed* iff ψ imposes both universal and existential properties in a nontrivial way. Thus, ψ is mixed iff neither ψ nor $\neg\psi$ is universal. We first show that formulas that are not mixed are insensitive to the environment being nondeterministic.

Theorem 5. *Consider a module M and a specification ψ. If ψ is not mixed, then M robustly satisfies ψ iff $M||M' \models \psi$ for every deterministic M'.*

Proof (sketch): Clearly, if M robustly satisfies ψ, then $M||M' \models \psi$ for every deterministic M'. For the other direction, assume that ψ is not mixed and that $M||M' \models \psi$ for every deterministic M'. We prove that then, M robustly satisfies ψ. Thus, that $M||M' \models \psi$ for every possibly nondeterministic M'. We distinguish between two cases. If ψ is universal, then, as M simulates $M||M'$ for every (possibly nondeterministic) M', robust satisfaction coincides with usual satisfaction and we are done. If ψ is existential, assume that there is a nondeterministic M' such that $M||M'$ does not satisfy ψ. Let M'' be any deterministic module obtained from M' by removing transitions. Since M' simulates M'', the composition $M||M'$ simulates the composition $M||M''$ [GL94]. Therefore, as ψ is existential, it must be that $M||M''$ does not satisfy ψ as well. $\square$

Thus, to robustly model check formulas that are not mixed, one can use the method of module checking with incomplete information [KV97]. We now study the problems of determining whether a given CTL formula is universal (or existential) or mixed, and show that they are all EXPTIME-complete.

Theorem 6. *Given a CTL formula ψ, checking whether ψ is universal is EXPTIME-complete.*

Proof (sketch): For a set $\mathcal{T}$ of trees and an integer k, we define $reshape(\mathcal{T}, k)$ as the set of trees obtained from trees in $\mathcal{T}$ by prunings or duplications of subtrees, so that each node has at most k successors. Given a CTL formula ψ, let $k = \max\{1, \mathcal{E}(\psi)\}$, and let $\mathcal{T}$ be the set of trees of branching degree k that satisfy ψ. It can be shown that the formula ψ is universal iff $reshape(\mathcal{T}) \subseteq \mathcal{T}$. Given ψ, let $\mathcal{A}_\psi$ be a nondeterministic Buchi automation for ψ; that is, $\mathcal{L}(\mathcal{A}_\psi) = \mathcal{T}$. By "reshaping" the transition function of $\mathcal{A}_\psi$, we can define a nondeterministic Büchi automaton $\mathcal{A}'_\psi$ such that $\mathcal{L}(\mathcal{A}_\psi) = reshape(\mathcal{T}, k)$. Then, ψ is universal iff $\mathcal{L}(\mathcal{A}'_\psi) \subseteq \mathcal{L}(\mathcal{A}_\psi)$. In order to check the latter, we check the nonemptiness of $\mathcal{L}(\mathcal{A}'_\psi) \cap \mathcal{L}(\mathcal{A}_{\neg\psi})$. Since both $\mathcal{A}'_\psi$ and $\mathcal{A}_{\neg\psi}$ are exponential in $|\psi|$, and the nonemptiness check is polynomial, the EXPTIME upper bound follows.

For the lower bound, we do a reduction from alternating linear-space Turing machines. Given a machine T, we construct a CTL formula ψ such that ψ is universal iff the machine T does not accept the empty tape. Typically, ψ is satisfied in a tree iff the tree does not represent an accepting computation tree of T on the empty tape. We can define ψ that is polynomial in T. One can then prove that the machine T rejects the empty tape iff $\psi = \mathbf{true}$, and that $\psi = \mathbf{true}$ iff ψ is universal. $\square$

Theorem 7. *Given a CTL formula ψ, checking whether ψ is mixed is EXPTIME-complete.*

Proof (sketch): Since ψ is mixed iff both ψ and $\neg\psi$ are non-universal, the upper bound follows from Theorem 6. The lower bound is similar to the one in Theorem 6, only that now we prove that ψ is mixed iff the machine T accepts the empty tape. To prove this, we replace the second claim in the proof of Theorem 6 with the claim that $\psi = \mathbf{true}$ iff ψ is not mixed. $\square$

5 Related Work and Discussion

Different researchers have considered the problem of reasoning about open systems. The distinction, in [HP85], between closed and open systems first led to the realization that *synthesis* of open systems corresponds to a search for a winning strategy in a *game* between the system and the environment [PR89], in which the winning condition is expressed in terms of a linear temporal logic formula. Transformation of the game-theoretic approach to model checking and adjustment of verification methods to the open-system setting started, for linear temporal logic, with the problem of *receptiveness* [Dil89, AL93, GSSL94]. Essentially, the receptiveness problem is to determine whether every finite prefix of a computation of a given open system can be extended to an infinite computation that satisfies a linear temporal property irrespective of the behavior of the environment. In *module checking* [KV96], the setting is again game-theoretic: an open system is required to satisfy a branching temporal property no matter how the environment disables its transitions. Verification of open systems was formulated in terms of a game between agents in a multi-agent system in [AHK97]. *Alternating-time temporal logic*, introduced there, enables path quantifiers to range over computations that a team of agents can force the system into, and thus enables the specification of multi-agent systems. In particular, ATL and ATL* are the alternating-time versions of CTL and CTL*, respectively.

Unlike [AHK97], in which all the agents of the system are specified, our setting here assumes that only one agent, namely the system, is given. We ask whether there exists another agent, namely the environment, which is not yet known, such that the composition of the system and the environment violates a required property. Thus, while the outcome of the games that correspond to alternating temporal logic are computations, here the outcomes are trees[4]. The unknown environment may be nondeterministic, thus the branching structure of the trees is not necessarily a restriction of the branching structure of the system. Since the properties we check are branching, the latter point is crucial. As follows from the 2EXPTIME lower bounds for both ATL* model checking and CTL* robust model checking, verification of general properties of open systems is "robustly hard". Exceptions are universal properties, for which robust satisfaction coincides with usual satisfaction, as well as properties that can be specified in the logic ATL. Indeed, the logic ATL identifies a class of properties for open systems for which it suffices to solve iterated finite games, which can be done in linear time.

Robust satisfaction is closely related to *supervisory control* [RW89, Ant95]. Given a finite-state machine whose transitions are partitioned into controllable and uncontrollable, and a specification for the machine, the control problem requires the construction of a controller that chooses the controllable transitions so that the machine always satisfies the specification. Clearly, checking whether all the compositions $M||M'$ of a system M with an environment M' satisfies a property ψ is dual to checking whether there is a controller M' such that $M||M'$ satisfy the property $\neg\psi$. Thus, from a control-theory point of view, the results of this paper generalize known supervisory-control methods to the case where both the system and the controller are nondeterministic Moore machines. In particular, our results imply that nondeterministic controllers are more powerful than deterministic ones, and describe how to synthesize finite-state controllers.

Often, the requirement that M satisfies ψ in all environments is too restrictive, and we are really concerned in the satisfaction of ψ in compositions of M with environments about which some *assumptions* are known. In the *assume-guarantee* paradigm to verification, each specification is a pair $\langle\varphi, \psi\rangle$, and M satisfies $\langle\varphi, \psi\rangle$ iff for every M', if $M||M'$ satisfies φ, then $M||M'$ also satisfies ψ. When φ and ψ are given in linear temporal logic, M satisfies $\langle\varphi, \psi\rangle$ iff M satisfies the implication $\varphi \to \psi$ [Pnu85] (see also [JT95]). The situation is different in the branching paradigm. For universal temporal logic, M satisfies $\langle\varphi, \psi\rangle$ iff ψ is satisfied in the composition $M||M_\varphi$, of M with a module M_φ that embodies all the behaviors that satisfy φ [GL94, KV95]. For general branching temporal logic, the above is no longer valid. Robust model checking can be viewed as a special case of the assume-guarantee setting, where φ is $\mathbf{true}$. Robust model checking, however, can be used to solve the general assume-guarantee setting. Indeed, M satisfies $\langle\varphi, \psi\rangle$ iff M robustly satisfies the implication $\varphi \to \psi$. Thus, while in the linear framework the assume-guarantee paradigm corresponds to usual model checking, robustness is required in the branching framework.

Since assumptions about the environment and its interaction with the systems are natural part of the specification in robust model checking, the model studied in this paper subsumes extensions that can be expressed in terms properties of the environment and its interaction with the system. For example, recall that our compositions here are deadlock free, thus deadlock is modeled by entering some "bad" state. In order to check that M satisfies a property ψ in all the compositions $M||M'$ in which this bad state is not

[4] Game logic [AHK97] considers games in which the output are trees, yet both players are known.

reachable, we have to perform robust model checking of M with respect to the property $(AG\theta) \rightarrow \psi$, with $\theta = \neg bad$, assuming that the bad state is labeled by bad. In a similar way, we can specify in θ other global assumptions about the composition, and thus model settings that support handshaking or other forms of coordinations between processes, as well as more general global actions, as in [HF89].

References

[AHK97] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proc. 38th IEEE Symposium on Foundations of Computer Science*, pages 100–109, 1997.

[AL93] M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, 1993.

[Ant95] M. Antoniotti. *Synthesis and verification of discrete controllers for robotics and manufacturing devices with temporal logic and the Control-D system.* PhD thesis, New York University, New York, 1995.

[BVW94] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In D. L. Dill, editor, *Proc. 6th CAV*, LNCS 818, pages 142–155, 1994.

[CE81] 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*, LNCS 131, pages 52–71, 1981.

[CGB86] E.M. Clarke, O. Grumberg, and M.C. Browne. Reasoning about networks with many identical finite-state processes. In *Proc. 5th PODC*, pages 240–248, 1986.

[Dil89] D.L. Dill. *Trace theory for automatic hierarchical verification of speed independent circuits*. MIT Press, 1989.

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

[EL85] 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.

[Eme90] E.A. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, pages 997–1072, 1990.

[FZ88] M.J. Fischer and L.D. Zuck. Reasoning about uncertainty in fault-tolerant distributed systems. In M. Joseph, editor, *Proc. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 331, pages 142–158, 1988.

[GL91] O. Grumberg and D.E. Long. Model checking and modular verification. In *Proc. 2nd CONCUR*, LNCS 527, pages 250–265, 1991.

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

[GSSL94] R. Gawlick, R. Segala, J. Sogaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. In *Proc. 21st ICALP*, LNCS 820, pages 166–177, 1994.

[GW99] E. Graedel and I. Walukiewicz. Guarded fixed point logic. In *Proc. 14th LICS*, 1999.

[HF89] J.Y. Halpern and R. Fagin. Modelling knowladge and action in distributed systems. *Distributed Computing*, 3(4):159–179, 1989.

[HKQ98] T.A. Henzinger, O. Kupferman, and S. Qadeer. From pre-historic to post-modern symbolic model checking. In *Proc. 10th CAV*, LNCS 1427, 1998.

[HKV97] D. Harel, O. Kupferman, and M.Y. Vardi. On the complexity of verifying concurrent transition systems. In *Proc. 8th CONCUR*, LNCS 1243, pages 258–272, 1997. Springer-Verlag.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HP85] D. Harel and A. Pnueli. On the development of reactive systems. In K. Apt, editor, *Logics and Models of Concurrent Systems*, volume F-13 of *NATO Advanced Summer Institutes*, pages 477–498. Springer-Verlag, 1985.

[JT95] B. Jonsson and Y.-K. Tsay. Assumption/guarantee specifications in linear-time temporal logic. In *Proc. TAPSOFT '95*, LNCS 915, pages 262–276, 1995.

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

[KV95] O. Kupferman and M.Y. Vardi. On the complexity of branching modular model checking. In *Proc. 6th CONCUR*, LNCS 962, pages 408–422, 1995.

[KV96] O. Kupferman and M.Y. Vardi. Module checking. In *Proc. 8th CAV*, LNCS 1102, pages 75–86, 1996.

[KV97] O. Kupferman and M.Y. Vardi. Module checking revisited. In *Proc. 9th CAV*, LNCS 1254, pages 36–47, 1997.

[KV98] O. Kupferman and M.Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th STOC*, pages 224–233, 1998.

[Lam80] L. Lamport. Sometimes is sometimes "not never" - on the temporal logic of programs. In *Proc. 7th POPL*, pages 174–185, 1980.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Proc. 2nd International Joint Conference on Artificial Intelligence*, pages 481–489. British Computer Society, September 1971.

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

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

[Pnu85] A. Pnueli. In transition from global to modular temporal reasoning about programs. In K. Apt, editor, *Logics and Models of Concurrent Systems*, volume F-13 of *NATO Advanced Summer Institutes*, pages 123–144. Springer-Verlag, 1985.

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, 1989.

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th International Symp. on Programming*, LNCS 137, pages 337–351, 1981.

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

[RW89] P.J.G. Ramadge and W.M. Wonham. The control of descrete event systems. *IEEE Transactions on Control Theory*, 77:81–98, 1989.

[SE89] R.S. Streett and E.A. Emerson. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation*, 81(3):249–264, 1989.

[Tha73] J.W. Thatcher. Tree automata: an informal survey. In A.V. Aho, editor, *Currents in the theory of computing*, pages 143–172. Prentice-Hall, Englewood Cliffs, 1973.

[VW86] 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.