Verification of Open Systems[*]

Orna Kupferman
Hebrew University[†]

Moshe Y. Vardi
Rice University[‡]

January 15, 2006

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 essence, robust model checking focuses on reasoning algorithmically about interaction.

In this work 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.

[*]The chapter is based on our paper *Robust Satisfaction*, Proc. 10th Conference on Concurrency Theory, volume 1664 of Lecture Notes in Computer Science, pages 383-398, Springer-Verlag, 1999.

[†]Address: School of Computer Science and Engineering, Hebrew University, Jerusalem 91904, Israel. Email: orna@cs.huji.ac.il. Supported in part by BSF grant 9800096, and by a grant from Minerva.

[‡]Address: Department of Computer Science, Houston, TX 77251-1892, U.S.A. Email:vardi@cs.rice.edu. Supported in part by NSF grants CCR-9988322, CCR-0124077, CCR-0311326, IIS-9908435, IIS-9978135, EIA-0086264, and ANI-0216467, by BSF grant 9800096, by Texas ATP grant 003604-0058-2003, and by a grant from the Intel Corporation.

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]. Since an open system has control only about its internal nondeterminism, and should be able to function correctly with respect to all possible ways in which its external nondeterminism is resolved, the term *angelic* nondeterminism is used for nondeterminism that is resolved by the system, while *demonic* nondeterminism is nondeterminism that is resolved by the environment [MM01].

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 as a temporal-logic formula, and check formally that the model satisfies the formula. 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. In essence, robust model checking focuses on reasoning algorithmically about interaction.

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 system. 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 system. We distinguish here between *universal* and *non-universal* temporal logics. Formulas of universal temporal logics, such as LTL, $\forall$CTL, and $\forall$CTL*, describe requirements that should hold in all the branches of the tree [GL94]. These requirements may be either linear (e.g., in all computations, only finitely many requests are sent) as in LTL, or branching (e.g., in all computations we eventually reach a state from which, no matter how we continue, no requests are sent) as in ∀CTL. 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. On the other hand, formulas of non-universal temporal logics, such as CTL and CTL*, may also impose possibility requirements on the system (e.g., there exists a computation in which only finitely many requests are sent) [EH86]. Here, it is no longer true that simulation between systems corresponds to agreement on satisfaction of requirements. Indeed, it might be that adding behaviors to the system helps it to satisfy a possibility requirement or, equivalently, that disabling some of its behaviors causes the requirement not to be satisfied.

It turned out that model-checking methods are applicable also for verification of open systems with respect to universal temporal-logic formulas [MP92, KV96]. To see this, consider an execution of an open system in a maximal environment; i.e., an environment that enables all the external nondeterministic choices. The result is a closed system, and it simulates any other execution of the system in some environment. Therefore, one can check satisfaction of universal requirements in an open system by model checking the system viewed as a closed system (i.e., all nondeterministic choices are internal). This approach, however, cannot be adapted when verifying an open system with respect to non-universal requirements. Here, satisfaction of the requirements with respect to the maximal environment does not imply their satisfaction with respect to all environments. Hence, we should explicitly make sure that all possibility requirements are satisfied, no matter how the environment restricts the system.

To see the difference between robust satisfaction and usual satisfaction, consider the open system M described in Figure 1. The system M models a cash

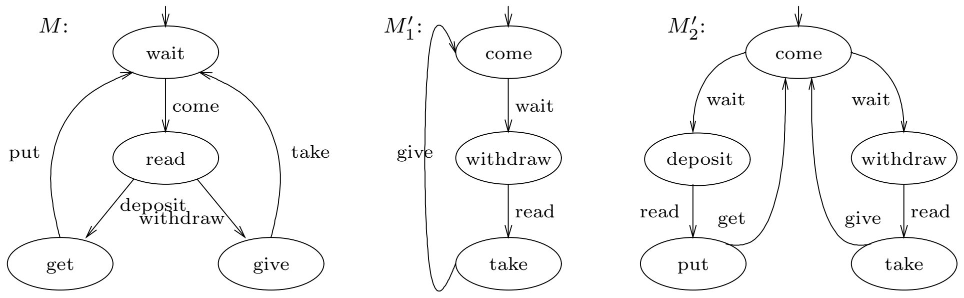

Figure 1: An ATM and two environments for it.

machine (ATM). Each state of the machine is labeled by the signal that the machine outputs when it visits the states. Each transition is labeled by the signal that the machine reads when the transition is taken. At the state labeled *wait*, M waits for customers. When a customer comes, M moves to the state labeled *read*, where it reads whether the customer wants to *deposit* or *withdraw* money. According to the external choice of the customer, 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 customers, 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 customers 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, KVW01], which suggested the method of *module checking*. In this algorithmic method we check, given M and ψ, whether, no matter how an environment disables some of M's transitions, it still 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. Technically, allowing the environment to disable some of M's transitions corresponds to restricting the robust-satisfaction problem to environments M' that are both *deterministic* and *have complete information*, in the sense that all the output variables of the system are read by the environment, thus the system has no internal variables.

This latter 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 a deterministic environment with a complete information corresponds to arbitrary disabling of transitions in M, the composition of M with a deterministic system with incomplete information 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. As an example, consider the variant of the ATM machine described in Figure 2. Here, the ATM I has an internal variable indicating whether it has money to give. The fact the variable is internal introduces nondeterminism in the description of I. Thus, I waits for customers, and when a customer comes, I consults the internal variable and moves accordingly to either the state labeled *read, full* or to the state labeled *read, empty.* The customer does not know whether the system is empty or full, and his choice is independent of this information. Only after the

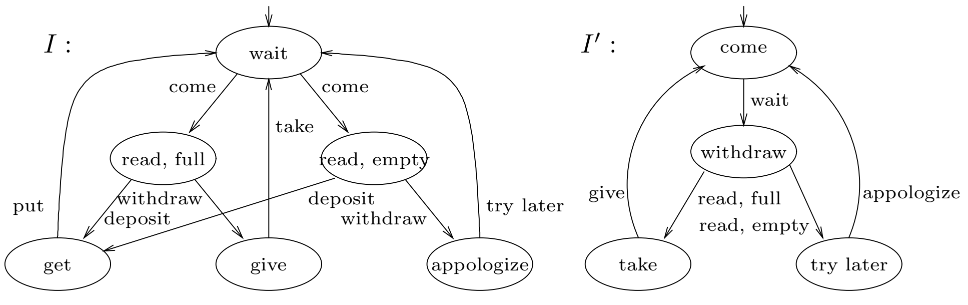

Figure 2: An ATM with internal variables and an environment for it.

choice is made, the system shares this information with the customer (in fact, in the fortunate cases of the system being full or the customer depositing money, the information is kept internal). The environment I' corresponds to the case where all customers withdraw money. Note that only after the choice is made, the customers may discover that the ATM has no money. Thus, technically, when we consider the composition of I with an environment I', we cannot consider, for example, environments in which the transition from the state labeled *read, full* to the state labeled *give* is enabled while the transition from the state labeled *read, empty* to the state labeled *apologize* is disabled, or vice versa.

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 M from Figure 1. 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 $AXEX give$[1]. 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' described in Figure 1. 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.

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

In this paper we study the robust-satisfaction problem and describe a unified approach and solution for it. Thus, given an open system M and a specification ψ, we solve 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 $M \| M'$ 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 non-universal branching temporal specifications is rather intractable.

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 refine the classification of specifications into universal and non-universal ones and show that the existential fragment of non-universal specifications is insensitive to the environment being nondeterministic. Finally, we 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 $V : T \rightarrow \Sigma$ 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 **true** and **false** and, as usual, $\wedge$ has precedence over $\vee$. The transition function $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(\Upsilon \times Q)$ 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 $r(\epsilon) = (\epsilon, q_0)$, for some $q_0 \in Q_0$.
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 the following hold:
   - 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 $\theta =$ **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 $\theta =$ **false**, we cannot have a run that takes a transition with $\theta =$ **false**.

Each infinite path ρ in $\langle T_r, r\rangle$ is labeled by a word $r(\rho)$ in Q^ω. Let *inf*(ρ) 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 *inf*$(\rho) \cap \alpha \neq \emptyset$. In *parity* alternating tree automata, $\alpha = \langle F_1, F_2, \ldots, F_{2k}\rangle$, with $F_1 \subset F_2 \subset \cdots \subset F_{2k} = Q$, and and infinite path ρ satisfies α iff the minimal index i for which *inf*$(\rho) \cap F_i \neq \emptyset$ is even. 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, KVW00]. 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. Theorem 2.1 below reviews the known constructions.

Theorem 2.1 [EJ91, KVW00]

(1) *A CTL or an alternation-free μ-calculus formula ψ can be translated to a symmetric alternating Büchi automaton with $O(|\psi|)$ states.*

(2) *A μ-calculus formula ψ can be translated to a symmetric alternating parity automaton with $O(|\psi|)$ states and index $O(|\psi|)$.*

(3) *A CTL* formula ψ can be translated to a symmetric alternating parity automaton with $2^{O(|\psi|)}$ states and index 3.*

2.2 Modules

A *module* is a tuple $M = \langle I, O, W, w_{in}, \rho, \pi\rangle$, 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, $\rho : W \times 2^I \to 2^W \setminus \{\emptyset\}$ is a nondeterministic transition function, and $\pi : W \to 2^O$ is a labeling function that assigns to each state its output. Note that we require that for all $w \in W$ and $\sigma \in 2^I$, the set $\rho(w, \sigma)$ is not empty. Thus, the module can always respond to external inputs, though the response might be to enter a "bad" state. The module M starts its execution in w_{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*.

Consider two modules $M = \langle I, O, W, w_{in}, \rho, \pi\rangle$ and $M' = \langle I', O', W', w'_{in}, \rho', \pi'\rangle$, such that $I \subseteq O'$ and $I' \subseteq O$. Note that the all the inputs of M are the outputs of M' and vice versa. The *composition* of M and M' is the closed module $M \| M' = \langle \emptyset, O \cup O', W'', w''_{in}, \rho'', \pi''\rangle$, where

- $W'' = W \times W'$.
- $w''_{in} = \langle w_{in}, w'_{in}\rangle$.
- For every state $\langle w, w'\rangle \in W''$, we have $\rho''(\langle w, w'\rangle, \emptyset) = \rho(w, \pi'(w') \cap I) \times \rho'(w', \pi(w) \cap I')$.
- For every state $\langle w, w'\rangle \in W''$, we have $\pi''(\langle w, w'\rangle) = \pi(w) \cup \pi'(w')$.

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.

We now define when a closed module M satisfies a formula. A closed module $M = \langle \emptyset, O, W, w_{in}, \rho, \pi \rangle$ can be induces an *enabling tree* $\langle T, V \rangle$. The enabling tree of M is a full infinite $\{\top, \bot\}$-labeled W-tree, thus $T = W^*$. Intuitively, $\langle T, V \rangle$ indicates which behaviors of M are enabled by labeling with $\top$ nodes that correspond to computations that M can traverse, and labeling other

Every closed module $M = \langle \emptyset, O, W, w_{in}, \rho, \pi \rangle$ induces an *enabling tree* $\langle T, V \rangle$. The enabling tree of M is a full infinite $\{\top, \bot\}$-labeled W-tree, thus $T = W^*$. Intuitively, $\langle T, V \rangle$ indicates which behaviors of M are enabled by labeling with $\top$ nodes that correspond to computations that M can traverse, and labeling other computations with $\bot$. Formally, we define $dir(\epsilon)$ to be w_{in}, and we label ϵ by $\top$. Consider a node $x \in T$ such that $dir(x) = w$ and $V(x) = \top$. For every state $w' \in W$, we define

$$V(x.w') = \left[\begin{array}{ll} \top & \text{if } w' \in \rho(w, \emptyset). \\ \bot & \text{otherwise.} \end{array} \right.$$

Consider a node $x = w_1, w_2, \ldots, w_m \in T$. By the definition of V, the module M can traverse the computation $w_{in}, w_1, w_2, \ldots, w_m$ iff all the prefixes y of x have $V(y) = \top$. Indeed, then and only then we have $w_1 \in \rho(w_{in}, \emptyset)$, and $w_{i+1} \in \rho(w_i, \emptyset)$ 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). Note that while every state in M_1 may read $2^{|I_1|}$ different inputs and move to $|W_1|$ successors, every state in $M_1 \| M_2$ may have $|W_1| \cdot |W_2|$ successors. Note also that M_2 may be nondeterministic. Accordingly, M_2 cannot only prune transitions of M_1 (by not providing the input with which this transition is taken, causing the transition not to contribute to a transition in the product); it can also split transitions of M_1 (by reacting nondeterministically to some output, causing a transition of M_1 to contribute several transitions in the product).

We now define when a closed module M satisfies a formula. Recall that the enabling tree of M is a full infinite $\{\top, \bot\}$-labeled W-tree. As we shall see in Section 3, the fact that the tree is full circumvents some technical difficulties. In order to define when M satisfies a formula, 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 that is obtained from $\langle T, V \rangle$ by pruning subtrees with a root labeled $\bot$. For a closed module M with output signals in O, the *computation tree* of M is a 2^O-labeled W-tree obtained from the $\top$-restriction of M's enabling tree by replacing the $\top$ label of a node with direction w by the label $\pi(w)$. We say that M satisfies a branching temporal logic formula ψ over 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, but we 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. The "sufficient branching-degree" property for branching temporal logics states that if a CTL* or a μ-calculus formula ψ is satisfiable, then ψ is also satisfiable in a computation tree of branching degree $\mathcal{E}(\psi)+1$ [CE81, SE89]. We now extend this result and show that in robust model checking of a module M with state space W it suffices to consider environments of degree $|W|(\mathcal{E}(\psi)+1)$. We not that while this bound is good enough for obtaining tight complexity bounds for the robust satisfaction problem (other factors of the problem dominate the complexity), we do not know whether the bound is tight.

Note that, unlike the classical sufficient branching-degree property for branching temporal logic, here we want to bound the branching degree of the environment, rather than that of the composition $M\|M'$. Consider, for example, a module M with an initial state s_0 that has two successors: a state s_1 with $p \in \pi(s_1)$ and a state s_2 with $p \notin \pi(s_2)$ In order for M to satisfy the formula $\psi = EX(p \wedge q) \wedge EX(p \wedge \neg q)$, for an input variable q, we have to split the state s_1. Though $\mathcal{E}(\psi) = 2$, such a split may result in a composition of branching degree 4. It can, however, be achieved by composing M with an environment M' of branching degree 2, say $\rho'(s'_0, p) = \{s'_1, s'_2\}$, with $q \in \pi'(s_1)$ and $q \notin \pi(s_2)$. Theorem 3.1 below shows that it is sufficient to compose M with an environment of branching degree $|W|(\mathcal{E}(\psi)+1)$. Intuitively, it follows from the fact that we never have to split a state into more than $|W|(\mathcal{E}(\psi)+1)$ states.

Theorem 3.1 *Consider a module M and a branching temporal logic formula ψ over $I \cup O$. If there exists M' such that $M\|M' \models \psi$, then there also exists M'' of exact degree $|W|(\mathcal{E}(\psi)+1)$ such that $M\|M'' \models \psi$.*

Proof (sketch): A temporal logic formula ψ is satisfiable iff there is a module M of branching degree $\mathcal{E}(\psi)+1$ satisfying it. The proof is based on the definition of a choice function, which maps each state and subformula that is satisfied in the state and involves a disjunction (either an explicit disjunction like $\varphi_1 \vee \varphi_2$ or an explicit disjunction like existential formulas or least fixed-point) to the way it is satisfied (for example, to φ_1 or φ_2 in case of an explicit disjunction, and to a particular successor in the case of an existential formula). For CTL and the μ-calculus, it is shown in [EH86] and [SE89], respectively, that the choice function may require a state s to have at most $\mathcal{E}(\psi)$ successors in order to satisfy all the formulas that are satisfied in s. For CTL*, the need is for $\mathcal{E}(\psi)+1$ successors [ES84], where the additional branch guarantees we do not block the path along which path formulas are satisfied.

Our case is more complicated, as we need to bound the branching degree of the model with which we compose M. By increasing the bound by a $|W|$ factor, we can use the techniques of [EH86, SE89, ES84]: the $|W|$ factor guarantees that if ψ is satisfied in $M\|M'$ for some M', and the choice function with respect to $M\|M'$ maps different existential formulas that are associated with state $\langle s,s'\rangle$ to successors $\langle t_1,t'_1\rangle,\ldots,\langle t_k,t'_k\rangle$ of $\langle s,s'\rangle$ with the same W-component (that is, there is $t\in W$ such that $t_i=t$ for several i's), then the choice function for $M\|M''$ can use the same W-component as well.

□

It is an open question whether the factor $|W|$ in the bound in theorem is needed.

We now use Theorem 3.1 to show that the robust-satisfaction problem for branching temporal logics can be reduced to the emptiness problem for alternating tree automata. For an integer $k\geq 1$, let $[k]=\{1,\ldots,k\}$.

Theorem 3.2 *Consider a module M with state space W 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 $k=|W|(\mathcal{E}(\psi)+1)$. 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: 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' interacts 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 that 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 $(2^O \times [k])$-trees, instead of $(W \times W')$-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 $M = \langle I, O, W, w_{in}, \rho, \pi \rangle$. 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 [KVW00]. 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 each environment induces a different computation tree, so there are many computation trees to check, and a 1-letter automaton does not suffice. Let $\mathcal{A}_\psi = \langle 2^{I \cup O}, Q, q_0, \delta, \alpha \rangle$. We define $\mathcal{A}_{M,\psi} = \langle 2^I, Q', q_0', \delta', \alpha' \rangle$, where

- $Q' = W \times Q$. 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.
- $q_0' = \langle w_{in}, q_0 \rangle$.
- 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
  - a disjunction $\Diamond q'$ by the disjunction $\bigvee_{\upsilon \in 2^O} \bigvee_{j \in [k]} \bigvee_{w' \in s(w,\sigma,\upsilon)} (\langle \upsilon, j \rangle, \langle w', q' \rangle)$.

  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 size of $\rho(w, \sigma)$. 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, thus for all $w \in W$ and $\sigma \in 2^I$, the set $s(w, \sigma, \upsilon)$ is not empty for at least one $\upsilon \in 2^O$, 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 prove that $\mathcal{L}(\mathcal{A}_{M,\psi})$ is empty iff M robustly satisfies $\neg\psi$. Assume first that $\mathcal{L}(\mathcal{A}_{M,\psi})$ is not empty. We prove that there is an environment M' such that $M \| M' \models \psi$, thus M does not robustly satisfy $\neg\psi$. Let $\langle T, V \rangle$ be a 2^I-labeled $(2^O \times [k])$-tree accepted by $\mathcal{A}_{M,\psi}$. We define $M' = \langle O, I, (2^O \times [k])^*, \epsilon, \rho, \pi' \rangle$, where for all states $y \in (2^O \times [k])^*$, we have $\pi'(y) = V(y)$, and for all $\upsilon \in 2^O$, we have $\rho'(y, \upsilon) = \{y \cdot \langle \upsilon', j \rangle : \upsilon' \in 2^O \text{ and } j \in [k]\}$. Thus, the output of the environment M' is induced by $\langle T, V \rangle$, and regardless of its input, M' branches to $2^{|O|}k$ successor, each extending the current state by a different pair in $2^O \times [k]$.

In order to prove that $M \| M'$ satisfies ψ, we show how the accepting run of $\mathcal{A}_{M,\psi}$ on $\langle T, V \rangle$ induces an accepting run of $\mathcal{A}_\psi$ on the computation tree of $M \| M'$. Let $\langle T_r, r \rangle$ be the accepting run of $\mathcal{A}_{M,\psi}$ on $\langle T, V \rangle$. Consider the $((W \times (2^O \times [k])^*)^* \times Q)$-labeled tree $\langle T_r, r' \rangle$ in which for all $x \in T_r$, if $r(x) = \langle y, \langle w, q \rangle \rangle$, then $r'(x) = \langle \langle w, y \rangle, q \rangle$. We claim that $\langle T_r, r' \rangle$ is an accepting run of $\mathcal{A}_\psi$ on the computation tree of $M \| M'$. In order to see that, note that the state space of $M \| M'$ is $W \times (2^O \times [k])^*$ and its transition function ρ'' is such that $\rho''(\langle w, y \rangle, \emptyset) = \rho(w, \pi'(y)) \times \rho'(y, \pi(w)) = \bigcup_{\upsilon \in 2^O} (s(w, \pi'(y), \upsilon) \times \{y \cdot \langle \upsilon', j \rangle : \upsilon' \in 2^O \text{ and } j \in [k]\})$. Consider a node $x \in T_r$ with $r(x) = \langle y, \langle w, q \rangle \rangle$. Let $\upsilon \in 2^O$ be such that $dir(y) = \langle \upsilon, j \rangle$ for some $j \in [k]$. Each conjunction $\Box q'$ in $\delta(q, \upsilon \cup \pi(w))$ induces $|\rho(w, \upsilon) \times 2^O \times [k]|$ successors to x, labeled by exactly all the elements in $\{\langle y \cdot \langle \upsilon', j \rangle, \langle w', q' \rangle \rangle : \upsilon' \in 2^O, j \in [k], \text{ and } w' \in \rho(w, \upsilon)\}$. Similarly, each disjunction $\Diamond q'$ in $\delta(q, \upsilon \cup \pi(w))$ induces a single successor to x, labeled by $\langle y \cdot \langle \upsilon', j \rangle, \langle w', q' \rangle \rangle$, for some $\upsilon' \in 2^O, j \in [k]$, and $w' \in \rho(w, \upsilon)$. Now, in r', we have $r'(x) = \langle \langle w, y \rangle, q \rangle$, and each conjunction $\Box q'$ in $\delta(q, \upsilon \cup \pi(w))$ induces $|\rho(w, \upsilon) \times 2^O \times [k]|$ successors to x, labeled by exactly all the elements in $\{\langle \langle w', y \cdot \langle \upsilon', j \rangle \rangle, q' \rangle : \upsilon' \in 2^O, j \in [k], \text{ and } w' \in \rho(w, \upsilon)\}$. Similarly, each disjunction $\Diamond q'$ in $\delta(q, \upsilon \cup \pi(w))$ induces a single successor to x, labeled by $\langle \langle w', y \cdot \langle \upsilon', j \rangle \rangle, q' \rangle$, for some $\upsilon' \in 2^O, j \in [k]$, and $w' \in \rho(w, \upsilon)$. Thus, r' is a legal run of $\mathcal{A}_\psi$ on the computation tree of $M \| M'$. Finally, by the definition of α', the fact that $\langle T_r, r \rangle$ is accepting implies that so is $\langle T_r, r' \rangle$.

For the other direction, assume that M does not robustly satisfy $\neg\psi$. Then, by Theorem 3.1, there is an environment M' of branching degree k such that $M \| M' \models \psi$. Let $M' = \langle O, I, W', w'_{in}, \rho', \pi' \rangle$. We define a 2^I-labeled $(2^O \times [k])$-

tree $\langle T, V\rangle$ accepted by $\mathcal{A}_{M,\psi}$. Intuitively, $\langle T, V\rangle$ is an unwinding of M', and we first define it as a W'-labeled tree $\langle T, f\rangle$ as follows. For the root, we have $f(\epsilon) = w'_{in}$. For a node $y \cdot \langle \upsilon, j\rangle$, with $f(y) = w'$, let $\langle w'_1, \ldots, w'_k\rangle$ be an ordering on the k successors of w' in $\rho'(w', \upsilon)$. We define $f(y \cdot \langle \upsilon, j\rangle) = w'_j$. In order to get from $\langle T, f\rangle$ the 2^I-labeled tree $\langle T, V\rangle$, we define $V(y) = \pi'(f(y))$ for all $y \in T$. In the product of M with M', a node y in $\langle T, V\rangle$ may be paired with several states of M. In order to prove that $\langle T, V\rangle$ is accepted by $\mathcal{A}_{M,\psi}$, we show how an accepting run $\langle T_r, r\rangle$ of $\mathcal{A}_\psi$ on the computation tree of $M\|M'$ induces an accepting run $\langle T_r, r'\rangle$ of $\mathcal{A}_{M,\psi}$ on $\langle T, V\rangle$. Intuitively, a copy of $\mathcal{A}_\psi$ in state q that reads a node $x \in (W \times W')^*$ with direction $\langle w, w'\rangle$ induces a copy of $\mathcal{A}_{M,\psi}$ in state $\langle w, q\rangle$ that reads a node $y \in T$ for which $f(y) = w'$ and y is paired with w (and possibly with other states of M, which induce additional copies that read y) in the product of M with M'. Let $\langle T, g\rangle$ be a 2^W-labeling of T in which each node y is labeled by the set of states that y is paired with in the product of M with M'. Thus, $g(\epsilon) = \{w'_{in}\}$ and $g(y \cdot \langle \upsilon, j\rangle) = \bigcup_{w \in g(y)} s(w, \pi'(f(y)), \upsilon)$. For a node $y = \langle \upsilon_1, j_1\rangle \cdot \langle \upsilon_2, j_2\rangle \cdot \langle \upsilon_l, j_l\rangle$ and $0 \leq i \leq l$, let $y[i]$ denote the prefix of length i of y; thus $y[i] = \langle \upsilon_1, j_1\rangle \cdot \langle \upsilon_2, j_2\rangle \cdot \langle \upsilon_i, j_i\rangle$. We say that a node $y \in T$ corresponds to a node $z = \langle w_0, w'_0\rangle, \langle w_1, w'_1\rangle, \ldots, \langle w_l, w'_l\rangle$ in the computation tree of $M\|M'$ if $|y| = l$ and for all $0 \leq i \geq l$, we have that $f(y[i]) = w'_i$ and $w_i \in g(y[i])$. In addition, for all $0 \leq i \leq l - 1$, we have that $w_{i+1} \in \rho(w_i, \pi'(w_i))$. Note that y corresponds to $|g(y)|$ nodes. On the other hand, only a single node $y \in T$ corresponds to z; indeed, $w'_0, w'_1, \ldots, w'_l$ fix a single sequence of output signals (the 2^O elements) and nondeterministic choices (the $[k]$ elements). We can now define the accepting run $\langle T_r, r'\rangle$ of $\mathcal{A}_{M,\psi}$ on $\langle T, V\rangle$. Consider a node $x \in T_r$ with $r(x) = \langle z, q\rangle$. Let $dir(z) = \langle w, w'\rangle$ and let y be the single node in T that corresponds to z. Then, $r'(x) = \langle y, \langle w, q\rangle\rangle$. The fact that $\langle T_r, r'\rangle$ is a legal run follows from the way we define the transition function of $\mathcal{A}_{M,\psi}$ and the labeling f and g. Finally, by the definition of α', the fact that $\langle T_r, r\rangle$ is accepting implies that so is $\langle T_r, r'\rangle$. $\square$

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

Theorem 3.3 *Robust model checking is*

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

(2) *2EXPTIME-complete for CTL*.*

Proof: 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 Theorem 2.1, $\mathcal{A}_\psi$ 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 index $O(n)$ for ψ in μ-calculus, and $\mathcal{A}_\psi$ is a parity automaton with $2^{O(n)}$ states and index 3 for ψ in CTL*. In Theorem 3.2, 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 $|M|\cdot|\mathcal{A}_\psi|\cdot|W|\cdot(\mathcal{E}(\psi)+1)$, and which has the same type as $\mathcal{A}_\psi$. The upper bounds then follow from the fact the nonemptiness problem for alternating Büchi tree automata can be solved in exponential time, whereas the one for an alternating parity automaton with m states and index h can be solved in time $m^{O(h)}$ [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. The details are similar to the reduction from satisfiability described for the related problem of module checking in [KVW01]. Essentially, 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 $(\mathcal{E}(\psi)+1)$. Then, one can relate the choice of the labels to choices made by the environment. $\square$

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 coincide 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 4, this result is of great importance in the dual paradigm of supervisory control, where instead of hostile environments we consider collaborative controllers.

Theorem 3.4 *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 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, KVW01], 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 [AHK02]. *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.

Unlike [AHK02], 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[2]. 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.

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. An extension to our setting here, described from the control-theory point of view, is the case where the controlled system may work in a non-maximal environment. Thus, we would like to know whether M has a controller M' such that for all environments M'', the composition $M \| M' \| M''$ satisfies the specification. This

[2]Game logic [AHK02] considers games in which the output are trees, yet both players are known.

setting is studied, for specifications in CTL and CTL*, in [KMTV00], where it is shown that the additional requirement makes the problem exponentially harder. Intuitively, the exponential increase in the complexity follows from the extra nesting of alternating "exists" and "for alls" in the description of the problem.

Recall that only non-universal specification formalisms are sensitive to the distinction between open and closed systems. In particular, specifications in linear temporal logic are not sensitive. 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 possibility properties like $AGEFp$. Existential properties describe requirements that should hold in *some* computations of the system. In [KV99], 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].

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 $\varphi \rightarrow \psi$ [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 **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 $\varphi \rightarrow \psi$. 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 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

[AHK02] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, September 2002.

[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.

[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*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

[CGB86] E.M. Clarke, O. Grumberg, and M.C. Browne. Reasoning about networks with many identical finite-state processes. In *Proc. 5th ACM Symp. on Principles of Distributed Computing*, pages 240–248, Calgary, Alberta, August 1986.

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

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

[EJ91] E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 368–377, 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. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 997–1072. Elsevier, MIT Press, 1990.

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

[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*, volume 331 of *Lecture Notes in Computer Science*, pages 142–158. Springer-Verlag, 1988.

[GL91] O. Grumberg and D.E. Long. Model checking and modular verification. In *Proc. 2nd Conference on Concurrency Theory*, volume 527 of *Lecture Notes in Computer Science*, pages 250–265. Springer-Verlag, 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 *Automata, Languages, and Programming, Proc. 21st ICALP*, volume 820 of *Lecture Notes in Computer Science*, pages 166–177. Springer-Verlag, 1994.

[GW99] E. Graedel and I. Walukiewicz. Guarded fixed point logic. In *Proc. 14th Symp. on Logic in Computer Science*, July 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 *Computer Aided Verification, Proc. 10th International Conference*, volume 1427 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[HKV97] D. Harel, O. Kupferman, and M.Y. Vardi. On the complexity of verifying concurrent transition systems. In *Proc. 8th Conference on Concurrency Theory*, volume 1243 of *Lecture Notes in Computer Science*, pages 258–272, Warsaw, July 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 P.D. Mosses, M. Nielsen, and M.I. Schwartzbach, editors, *TAPSOFT '95: Theory and Practice of Software Development*, volume 915 of *Lecture Notes in Computer Science*, pages 262–276, Aarhus, Denmark, May 1995. Springer-Verlag.

[KMTV00] O. Kupferman, P. Madhusudan, P.S. Thiagarajan, and M.Y. Vardi. Open systems in reactive environments: Control and synthesis. In *Proc. 11th International Conference on Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 92–107. Springer-Verlag, 2000.

[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 Conference on Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 408–422, Philadelphia, August 1995. Springer-Verlag.

[KV96] O. Kupferman and M.Y. Vardi. Module checking. In *Computer Aided Verification, Proc. 8th International Conference*, volume 1102 of *Lecture Notes in Computer Science*, pages 75–86. Springer-Verlag, 1996.

[KV97] O. Kupferman and M.Y. Vardi. Module checking revisited. In *Computer Aided Verification, Proc. 9th International Conference*, volume 1254 of *Lecture Notes in Computer Science*, pages 36–47. Springer-Verlag, 1997.

[KV98] O. Kupferman and M.Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th ACM Symp. on Theory of Computing*, pages 224–233, Dallas, 1998.

[KV99] O. Kupferman and M.Y. Vardi. Robust satisfaction. In *Proc. 10th Conference on Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 383–398. Springer-Verlag, August 1999.

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

[KVW01] O. Kupferman, M.Y. Vardi, and P. Wolper. Module checking. *Information and Computation*, 164:322–344, 2001.

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

[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.

[MM01] A.K. McIver and C. Morgan. Demonic, angelic and unbounded probabilistic choices in sequential programs. *Acta Informatica*, 37(4-5):329–354, 2001.

[MP92] Z. Manna and A. Pnueli. Temporal specification and verification of reactive modules. Technical report, Weizmann Institute, 1992.

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

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

[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 ACM Symp. on Principles of Programming Languages*, pages 179–190, Austin, January 1989.

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

[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 discrete 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 μ-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.