Module Checking Revisited*

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

[1] EECS Department, UC Berkeley, Berkeley CA 94720-1770, U.S.A.
Email: orna@eecs.berkeley.edu
[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. When we verify the correctness of an open system with respect to a desired requirement, we should take into consideration the different environments with which the system may interact. Each environment induces a different behavior of the system, and we want all these behaviors to satisfy the requirement. *Module checking* is an algorithmic method that checks, given an open system (modeled as a finite structure) and a desired requirement (specified by a temporal-logic formula), whether the open system satisfies the requirement with respect to all environments. In this paper we extend the module-checking method with respect to two orthogonal issues. Both issues concern the fact that often we are not interested in satisfaction of the requirement with respect to all environments, but only with respect to these that meet some restriction. We consider the case where the environment has *incomplete information* about the system; i.e., when the system has internal variables, which are not readable by its environment, and the case where some *assumptions* are known about environment; i.e., when the system is guaranteed to satisfy the requirement only when its environment satisfies certain assumptions. We study the complexities of the extended module-checking problems. In particular, we show that for universal temporal logics (e.g., LTL, $\forall$CTL, and $\forall$CTL*), module checking with incomplete information coincides with module checking, which by itself coincides with model checking. On the other hand, for non-universal temporal logics (e.g., CTL and CTL*), module checking with incomplete information is harder than module checking, which is by itself harder than model checking.

1 Introduction

Temporal logics, which are modal logics geared towards the description of the temporal ordering of events, have been adopted as a powerful tool for specifying and verifying reactive systems [Pnu81]. One of the most significant developments in this area is the discovery of algorithmic methods for verifying temporal-logic properties of *finite-state* systems [CE81, QS81, LP85, CES86]. This derives its significance both from the fact that many synchronization and communication protocols can be modeled as finite-state systems, as well as from the great ease of use of fully algorithmic methods.

We distinguish between two types of temporal logics: *universal* and *non-universal*. Both logics describe the computation tree induced by the system. Formulas of universal temporal logics 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) or branching (e.g., in all computations we eventually reach a state from which, no matter how

* Part of this work was done in Bell Laboratories during the DIMACS special year on Logic and Algorithms.
** Supported in part by the ONR YIP award N00014-95-1-0520, by the NSF CAREER award CCR-9501708, by the NSF grant CCR-9504469, by the AFOSR contract F49620-93-1-0056, by the ARO MURI grant DAAH-04-96-1-0341, by the ARPA grant NAG2-892, and by the SRC contract 95-DC-324.036.
*** Supported in part by the NSF grant CCR-9628400.

we continue, no requests are sent). 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 may also impose possibility requirements on the system (e.g., there exists a computation in which only finitely many requests are sent). 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.

We also distinguish between two types of systems: *closed* and *open* [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 required property, we translate the system into some formal model, specify the property with 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. In order to check whether an open system satisfies a required property, we should check the behavior of the system with respect to any environment, and often there is much uncertainty regarding the environment [FZ88]. In particular, it might be that the environment does not enable all the external nondeterministic choices. To see this, consider a sandwich-dispensing machine that serves, upon request, sandwiches with either ham or cheese. The machine is an open system and an environment for the system is an infinite line of hungry people. Since each person in the line can like either both ham and cheese, or only ham, or only cheese, each person suggests a different disabling of the external nondeterministic choices. Accordingly, there are many different possible environments to consider.

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 a composition of an open system with a maximal environment; i.e., an environment that enables all the external nondeterministic choices. This composition is a closed system, and it is simulated by any other composition of the system with some environment. Therefore, one can check satisfaction of universal requirements in an open system by model checking the composition of the system with this maximal environment. As discussed in [KV96], this approach can not 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. For example, verifying that the sandwich-dispensing machine described above can always eventually serve ham, we want to make sure that this can happen no matter what the eating habits of the people in line are. Note that while this requirement holds with respect to the maximal environment, it does not hold, for instance, in an environment in which all the people in line do not like ham.

In [KV96], we suggested *module checking* as a general method for verification of open systems. Given an open system M and a temporal-logic formula ψ, the module-checking problem asks whether for all possible environments $\mathcal{E}$, the composition of M with $\mathcal{E}$ satisfies ψ. In this paper we extend the module-checking method with respect to two orthogonal issues. Both issues concern the fact that often we are not interested in satisfaction of ψ with respect to all environments, but only with respect to those that meet some restriction. In particular, we consider the

case where $\mathcal{E}$ has *incomplete information* about M; i.e., not all the variables of M are readable by $\mathcal{E}$, and the case where some *assumptions* are known about $\mathcal{E}$. We now describe these extensions in more detail.

An interaction between a system and its environment proceeds through a designated set of input and output variables. In addition, the system often has internal variables, which the environment cannot read. If two states of the system differ only in the values of unreadable variables, then the environment cannot distinguish between them. Similarly, if two computations of the system differ only in the values of unreadable variables along them, then the environment cannot distinguish between them either and thus, its behaviors along these computations are the same. More formally, when we compose a system M with an environment $\mathcal{E}$, and several states in the composition look the same and have the same history according to $\mathcal{E}$'s incomplete information, then the nondeterministic choices done by $\mathcal{E}$ in each of these states coincide. In the sandwich-dispensing machine example, the people in line cannot see whether the ham and the cheese are fresh. Therefore, their choices are independent of this missing information. Given an open system M with a partition of M's variables into readable and unreadable, and a temporal-logic formula ψ, the module-checking problem with incomplete information asks whether the composition of M with $\mathcal{E}$ satisfies ψ, for all environments $\mathcal{E}$ whose nondeterministic choices are independent of the unreadable variables (that is, $\mathcal{E}$ behaves the same in indistinguishable states).

Often, the environment is known to satisfy some assumptions. In the sandwich-dispensing machine example, it may be useful to know that the machine is located in a vegetarian village. In the *assume-guarantee* paradigm [Jon83, Lam83], the specification of an open system consists of two parts. One part describes the guaranteed behavior of the system. The other part describes the assumed behavior of the environment with which the module is interacting. A system M then satisfies a specification with assumption φ and guarantee ψ if and only if in all compositions of M with $\mathcal{E}$, if the composition satisfies φ, then it satisfies ψ as well. Checking assume-guarantee specifications is helpful in modular verification [GL94]. For universal temporal logics, automatic methods for this check are suggested in [Pnu85, Var95, KV95]. These methods depend on the fact that the simulation order captures agreement on universal temporal-logic formulas, and they cannot be extended to handle non-universal formulas. Module checking can be viewed as a special case of the assume-guarantee paradigm, where the guarantee may be any formula, not necessarily a universal one, and the assumption is **true**. We extend here module checking to handle arbitrary assumptions. This suggests a complete and uniform assume-guarantee paradigm, for both the universal and non-universal settings, with both complete and incomplete information.

We solve the problems of module checking with assume-guarantee specifications and with incomplete information and consider their complexities. It turns out that while checking assume-guarantee specifications is not harder than module checking, the presence of incomplete information makes module checking more complex. To see why, consider an environment of a system with unreadable variables. Recall that several states in the composition of the system and the environment may be different and still be indistinguishable by the environment. Accordingly, the environment should behave the same in these states. Such a condition on the behavior of the environment relates remote nodes in the computation tree of the system, and there is no regular condition that relates these nodes (i.e., one cannot define an automaton that accepts all trees in which nodes that are indistinguishable by the environment have the same label). This need to relate indistinguishable nodes makes incomplete information very challenging.

We claim that *alternation* is a suitable and helpful mechanism for coping with incomplete information. Using *alternating tree automata*, we show that the problem of module checking with incomplete information is decidable. In particular, it is EXPTIME-complete and 2EXPTIME-complete for CTL and CTL*, respectively. As the module-checking problem for CTL is hard for

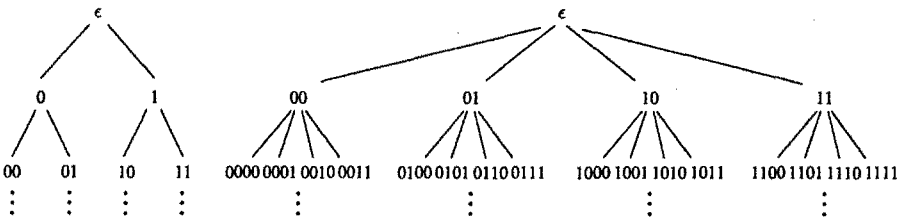
EXPTIME already for environments with complete information, it might seem as if incomplete information can be handled at no cost. This is, however, not true. While both problems can be solved in time that is exponential in the size of the formula, only the one with complete information can be solved in time that is polynomial in the size of the system [KV96]. On the other hand, module checking with incomplete information requires time that is exponential in both the formula and the system. 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, this implies that checking non-universal properties of open systems with internal variables is rather intractable.

2 Preliminaries

2.1 Trees and Labeled Trees

Given a finite set Υ, an *Υ-tree* is a nonempty set $T \subseteq \Upsilon^*$ such that if $s \cdot \upsilon \in T$, where $s \in \Upsilon^*$ and $\upsilon \in \Upsilon$, then also $s \in T$. 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 $s \in T$, the nodes $s \cdot \upsilon \in T$ where $\upsilon \in \Upsilon$ are the *children* of s. An Υ-tree T is a *full infinite tree* if $T = \Upsilon^*$. Each node s of T has a *direction* in Υ. The direction of the root is some designated $\upsilon_0 \in \Upsilon$. The direction of a node $s \cdot \upsilon$ is υ. An *infinite path* π of T is a set $\pi \subseteq T$ such that $\epsilon \in \pi$ and for every $s \in \pi$ there exists a unique $\upsilon \in \Upsilon$ such that $s \cdot \upsilon \in \pi$. 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.

For finite sets X and Y, and a node $s \in (X \times Y)^*$, let $hide_Y(s)$ be the node in X^* obtained from s by replacing each letter $(x \cdot y)$ by the letter x. For example (see figure), when $X = Y = \{0, 1\}$, the node 0010 of the $(X \times Y)$-tree on the right corresponds, by $hide_Y$, to the node 01 of the X-tree on the left. Note that the nodes 0011, 0110, and 0111 of the $(X \times Y)$-tree also correspondto the node 01 of the X-tree.



Let Z be a finite set. For a Z-labeled X-tree $\langle T, V \rangle$, we define the *Y-widening* of $\langle T, V \rangle$, denoted $wide_Y(\langle T, V \rangle)$, as the Z-labeled $(X \times Y)$-tree $\langle T', V' \rangle$ where for every $s \in T$, we have $hide_Y^{-1}(s) \subseteq T'$ and for every $t \in T'$, we have $V'(t) = V(hide_Y(t))$. Note that for every node $t \in T'$, and $x \in X$, the children $t \cdot (x \cdot y)$ of t, for all y, agree on their label in $\langle T', V' \rangle$. Indeed, they are all labeled with $V(hide_Y(t) \cdot x)$.

2.2 Modules and Composition of Modules

We describe a system by a *module* $M = \langle I, O, H, W, w_0, R, L \rangle$, where

- I, O, and H are sets of input, readable output, and hidden (internal) variables, respectively. We assume that I, O, and H are pairwise disjoint, we use K to denote the variables known to the environment; thus $K = I \cup O$, and we use P to denote all variables; thus $P = K \cup H$.
- W is a set of states, and $w_0 \in W$ is an initial state.

- $R \subseteq W \times W$ is a total transition relation. For $\langle w, w' \rangle \in R$, we say that w' is a successor of w. Requiring R to be total means that every state w has at least one successor.
- $L : W \rightarrow 2^P$ maps each state to the set of variables that hold in this state. The intuition is that in every state w, the module reads $L(w) \cap I$ and writes $L(w) \cap (O \cup H)$.

Note that M has no fairness condition. The difficulties caused by adding such a condition are orthogonal to the problems considered in this work. As we shall further discuss in Section 3.2, our framework can be easily adjusted to handle modules with fairness conditions. A *computation* of M is a sequence $w_0, w_1, \ldots$ of states, such that for all $i \geq 0$ we have $\langle w_i, w_{i+1} \rangle \in R$. We define the *size* $|M|$ of M as $(|W| * |P|) + |R|$. We assume, without loss of generality, that all the states of M are labeled differently; i.e., there exist no w_1 and w_2 in W for which $L(w_1) = L(w_2)$ (otherwise, we can add variables in H that differentiate states with identical labeling). With each module M we can associate a computation tree $\langle T_M, V_M \rangle$ obtained by pruning M from the initial state. More formally, $\langle T_M, V_M \rangle$ is a 2^P-labeled 2^P-tree (not necessarily with a fixed branching degree). Each node of $\langle T_M, V_M \rangle$ corresponds to a state of M, with the root corresponding to the initial state. A node corresponding to a state w is labeled by $L(w)$ and its children correspond to the successors of w in M. The assumption that the nodes are labeled differently enable us to embody $\langle T_M, V_M \rangle$ in a $(2^P)^*$-tree, with a node with direction v labeled v.

A module M is *closed* iff $I = \emptyset$. Otherwise, it is *open*. Consider an open module M. The module interacts with some environment $\mathcal{E}$ that supplies its inputs. When M is in state w, its ability to move to a certain successor w' of w is conditioned by the behavior of its environment. If, for example, $L(w') \cap I = \sigma$ and the environment does not supply σ to M, then M cannot move to w'. Thus, the environment may disable some of M's transitions. We can think of an environment to M as a *strategy* $\mathcal{E} : (2^K)^* \rightarrow \{\top, \bot\}$ that maps a finite history s of a computation (as seen by the environment) to either $\top$, meaning that the environment enables M to trace s, or $\bot$, meaning that the environment does not enable M to trace s. In other words, if M reaches a state w by tracing some $s \in (2^K)^*$, and a successor w' of w has $L(w) \cap K = \sigma$, then an interaction of M with $\mathcal{E}$ can proceed from w to w' iff $\mathcal{E}(s \cdot \sigma) = \top$. We say that the tree $\langle (2^K)^*, \mathcal{E} \rangle$ *maintains* the strategy applied by $\mathcal{E}$. We denote by $M \lhd \mathcal{E}$ the composition of M with $\mathcal{E}$; that is, the tree obtained by pruning from the computation tree $\langle T_M, V_M \rangle$ subtrees according to $\mathcal{E}$. Note that $\mathcal{E}$ may disable all the successors of w. We say that a composition $M \lhd \mathcal{E}$ is *deadlock free* iff for every state w, at least one successor of w is enabled. Given M, we can define the *maximal environment* $\mathcal{E}_{max}$ for M. The maximal environment has $\mathcal{E}_{max}(x) = \top$ for all $x \in (2^K)^*$; thus it enables all the transitions of M.

The hiding and widening operators (see Section 2.1) enable us to refer to the interaction of M with $\mathcal{E}$ as seen by both M and $\mathcal{E}$. As we shall see below, this interaction looks different from the two points of views. First, clearly, the labels of the computation tree of M, as seen by $\mathcal{E}$, do not contain variables in H. Consequently, $\mathcal{E}$ thinks that $\langle T_M, V_M \rangle$ is a 2^K-tree, rather than a 2^P-tree. Indeed, $\mathcal{E}$ cannot distinguish between two nodes that differ only in the values of variables in H in their labels. Accordingly, a branch of $\langle T_M, V_M \rangle$ into two such nodes is viewed by $\mathcal{E}$ as a single transition. This incomplete information of $\mathcal{E}$ is reflected in its strategy, which is independent of H. Thus, successors of a state that agree on the labeling of the readable variables are either all enabled or all disabled. Formally, if $\langle (2^K)^*, \mathcal{E} \rangle$ is the $\{\top, \bot\}$-labeled 2^K-tree that maintains the strategy applied by $\mathcal{E}$, then the $\{\top, \bot\}$-labeled 2^P-tree $wide_{(2^H)}(\langle (2^K)^*, \mathcal{E} \rangle)$ maintains the "full" strategy for $\mathcal{E}$, as seen by someone that sees both K and H.

Another way to see the effect of incomplete information is to associate with each environment $\mathcal{E}$ a tree obtained from $\langle T_M, V_M \rangle$ by pruning some of its subtrees. A subtree with root $s \in T_M$ is pruned iff $K'(hide_{(2^H)}(s)) = \bot$. Every two nodes s_1 and s_2 that are indistinguishable according to $\mathcal{E}$'s incomplete information have $hide_{(2^H)}(s_1) = hide_{(2^H)}(s_2)$. Hence, either both subtrees

with roots s_1 and s_2 are pruned or both are not pruned. Note that once $\mathcal{E}(x) = \bot$ for some $s \in (2^K)^*$, we can assume that $\mathcal{E}(s \cdot t)$ for all $t \in (2^K)^*$ is also $\bot$. Indeed, once the environment disables the transition to a certain node s, it actually disables the transitions to all the nodes in the subtree with root s. Note also that $M \lhd \mathcal{E}$ is deadlock free iff for every $s \in T_M$ with $\mathcal{E}(hide_{(2^H)}(s)) = \top$, at least one direction $v \in 2^P$ has $s \cdot v \in T_M$ and $\mathcal{E}(hide_{(2^H)}(s \cdot v)) = \top$.

3 Module Checking

The *module-checking* problem (with complete information) is defined as follows. Let M be a module with $H = \emptyset$, and let ψ be a temporal-logic formula over the set P of M's variables. Does $M \lhd \mathcal{E}$ satisfy ψ for every environment $\mathcal{E}$ for which $M \lhd \mathcal{E}$ is deadlock free? When the answer to the module-checking question is positive, we say that M *reactively satisfies* ψ, denoted $M \models_r \psi$. The module-checking problem is introduced and solved in [KV96][3]. We define two orthogonal extensions of the module-checking problem:

- *Module Checking with Incomplete Information*: Let M be a module and let ψ be a temporal-logic formula over P. Does $M \lhd \mathcal{E}$ satisfy ψ for every environment $\mathcal{E}$ for which $M \lhd \mathcal{E}$ is deadlock free?
- *Assume-Guarantee Module Checking*: Let M be a module with $H = \emptyset$ and let φ and ψ be temporal-logic formulas over P. Does $M \lhd \mathcal{E}$ satisfy ψ for every environment $\mathcal{E}$ for which $M \lhd \mathcal{E}$ is deadlock free and satisfies φ. When the answer to the assume-guarantee module-checking question is positive, we say that M reactively satisfies ψ with assumption φ, denoted $\langle \varphi \rangle M \langle \psi \rangle$.

In this section we solve the two extended problems, as well as the problem of assume-guarantee module checking with incomplete information, which subsumes them. We consider temporal-logic formulas in LTL, CTL, and CTL*. We first handle the case where ψ and φ are universal temporal-logic formulas. As shown in [KV96], checking whether M reactively satisfies a universal formula ψ can be reduced to checking whether $M \lhd \mathcal{E}_{max}$ satisfies ψ. Since $M \lhd \mathcal{E}_{max}$ is simulated by any composition $M \lhd \mathcal{E}$ irrespective of the variables readable by $\mathcal{E}$, this remains valid in the presence of incomplete information. In addition, the assume-guarantee problem for LTL, $\forall$CTL, and $\forall$CTL* (the universal fragments of CTL and CTL*, in which only universal path quantification is allowed) has been studied in the literature. Hence the following theorem.

Theorem 1.

(1) [KV96] *The module-checking problem with incomplete information is PTIME-complete (and solvable in linear time) for $\forall$CTL and is PSPACE-complete for LTL and $\forall$CTL*.*
(2) [Pnu85, KV95] *The assume-guarantee module-checking problem is PSPACE-complete for LTL and $\forall$CTL and is EXPSPACE-complete for $\forall$CTL*.*

As with module checking, things become more challenging when we turn to solve the problems for the case ψ and φ are not necessarily universal temporal-logic formulas. We first show that assume-guarantee module checking can be easily reduced to module checking.

Lemma 2. *For every module M and formulas φ and ψ, we have $\langle \varphi \rangle M \langle \psi \rangle$ iff $M \models_r \varphi \rightarrow \psi$.*

[3] In [KV96], we define a module using system and environment states, and only transitions from environment states may be disabled. Here, the interaction of the system with its environment is more explicit, and transitions are disabled by the environment assigning values to the system's input variables.

Lemma 2 follows immediately from the definition of assume-guarantee module checking. As the reduction to module checking is so simple, one may wonder why the original assume-guarantee problem, with φ and ψ in universal logics could not be simply reduced to model checking. The reason lies in the fact that universal temporal logics are not closed under negation. Thus, the formula $\varphi \rightarrow \psi$ is no longer a universal temporal-logic formula, and checking it with respect to any environment cannot be done easily. The reduction above implies that assume-guarantee module checking is not harder than module checking. As assume-guarantee module checking is also at least as hard as module checking, the theorem below follows from the known complexity bounds for the module-checking problem [KV96].

Theorem 3. *The assume-guarantee module-checking problem is EXPTIME-complete for CTL and is 2EXPTIME-complete for CTL*.*

While handling of assumptions about the environment is easy, handling incomplete information is complicated. The solution we suggest is based on alternating tree automata and is outlined below. In Sections 3.1 and 3.2, we define alternating tree automata and describe the solutions in detail. We start by recalling the solution to the module-checking problem. Given M and ψ, we proceed as follows.

A1. Define a nondeterministic tree automaton $\mathcal{A}_M$ that accepts all the 2^P-labeled trees that correspond to compositions of M with some $\mathcal{E}$ for which $M \lhd \mathcal{E}$ is deadlock free. Thus, each tree accepted by $\mathcal{A}_M$ is obtained from $\langle T_M, V_M \rangle$ by pruning some of its subtrees.

A2. Define a nondeterministic tree automaton $\mathcal{A}_{\neg\psi}$ that accepts all the 2^P-labeled trees that do not satisfy ψ.

A3. $M \models_r \psi$ iff no composition $M \lhd \mathcal{E}$ satisfies $\neg\psi$, thus iff the intersection of $\mathcal{A}_M$ and $\mathcal{A}_{\neg\psi}$ is empty.

The reduction of the module-checking problem to the emptiness problem for tree automata implies, by the finite-model property of tree automata [Eme85], that defining reactive satisfaction with respect to only *finite-state* environments is equivalent to the current definition.

In the presence of incomplete information, not all possible pruning of $\langle T_M, V_M \rangle$ correspond to compositions of M with some $\mathcal{E}$. In order to correspond to such a composition, a tree should be *consistent in its pruning*. A tree is consistent in its pruning iff for every two nodes that the paths leading to them differ only in values of variables in H (i.e., every two nodes that have the same history according to $\mathcal{E}$'s incomplete information), either both nodes are pruned or both nodes are not pruned. Intuitively, hiding variables from the environment makes it easier for M to reactively satisfy a requirement: out of all the pruning of $\langle T_M, V_M \rangle$ that should satisfy the requirement in the case of complete information, only these that are consistent should satisfy the requirement in the presence of incomplete information. Unfortunately, the consistency condition is non-regular, and cannot be checked by an automaton. In order to circumvent this difficulty, we employ alternating tree automata. We solve the module-checking problem with incomplete information as follows.

B1. Define an alternating tree automaton $\mathcal{A}_{M,\neg\psi}$ that accepts a $\{\top, \bot\}$-labeled 2^K-tree iff it corresponds to a strategy $\langle (2^K)^*, \mathcal{E} \rangle$ such that $M \lhd \mathcal{E}$ is deadlock free and does not satisfy ψ.

B2. $M \models_r \psi$ iff all deadlock free compositions of M with $\mathcal{E}$ that is independent of H satisfy ψ, thus iff no strategy induces a computation tree that does not satisfy ψ, thus iff $\mathcal{A}_{M,\neg\psi}$ is empty.

We now turn to a detailed description of the solution of the module-checking problem with incomplete information, and the complexity results it entails. For that, we first define formally alternating tree automata.

3.1 Alternating Tree Automata

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 $\delta : Q \times \Sigma \to \mathcal{B}^+(\Upsilon \times Q)$ 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 $\Upsilon^* \times Q$. 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. 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 $(1, q_2)$. Each infinite path ρ in $\langle T_r, r \rangle$ is labeled by a word $r(\rho)$ in Q^ω. Let $inf(\rho)$ denote the set of states in Q that appear in $r(\rho)$ infinitely often. A run $\langle T_r, r \rangle$ is accepting iff all its infinite paths satisfy the acceptance condition. In *Büchi* alternating tree automata, $\alpha \subseteq Q$, and an infinite path ρ satisfies α iff $inf(\rho) \cap \alpha \neq \emptyset$. 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 $\mathcal{L}(\mathcal{A}) \neq \emptyset$.

We define the *size* $|\mathcal{A}|$ of an alternating automaton $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, \alpha \rangle$ as $|Q| + |\alpha| + |\delta|$, where $|Q|$ and $|\alpha|$ are the respective cardinalities of the sets Q and α, and where $|\delta|$ is the sum of the lengths of the satisfiable (i.e., not **false**) formulas that appear as $\delta(q, \sigma)$ for some q and σ.

3.2 Solving the Problem of Module-Checking with Incomplete Information

Theorem 4. *Given a module M and a CTL formula ψ over the sets I, O, and H, of M's variables, there exists an alternating Büchi tree automaton $\mathcal{A}_{M,\psi}$ over $\{\top, \bot\}$-labeled $2^{I \cup O}$-trees, of size $O(|M| * |\psi|)$, such that $\mathcal{L}(\mathcal{A}_{M,\psi})$ is exactly the set of strategies $\mathcal{E}$ such that $M \lhd \mathcal{E}$ is deadlock free and satisfies ψ.*

Proof (sketch): Let $M = \langle I, O, H, W, w_0, R, L \rangle$, and let $K = I \cup O$. For $w \in W$ and $v \in 2^K$, we define $s(w, v) = \{w' \mid \langle w, w' \rangle \in R \text{ and } L(w') \cap K = v\}$ and $d(w) = \{v \mid s(w, v) \neq \emptyset\}$. That is, $s(w, v)$ contains all the successors of w that agree in their readable variables with v. Each such successor corresponds to a node in $\langle T_M, V_M \rangle$ with a direction in $hide^{-1}_{(2^H)}(v)$. Accordingly,

$d(w)$ contains all directions v for which nodes corresponding to w in $\langle T_M, V_M \rangle$ have at least one successor with a direction in $hide^{-1}_{(2^H)}(v)$.

Essentially, the automaton $\mathcal{A}_{M,\psi}$ is similar to the product alternating tree automaton obtained in the alternating-automata theoretic framework for CTL model checking [BVW94]. There, as there is a single computation tree with respect to which the formula is checked, the automaton obtained is a 1-letter automaton. Here, as there are many computation trees to check, we get a 2-letter automaton: each $\{\top, \bot\}$-labeled tree induces a different computation tree, and $\mathcal{A}_{M,\psi}$ considers them all. In addition, it checks that the composition of the strategy in the input with M is deadlock free. We assume that ψ is given in a positive normal form, thus negations are applied only to atomic propositions. We define $\mathcal{A}_{M,\psi} = \langle \{\top, \bot\}, Q, q_0, \delta, \alpha \rangle$, where

- $Q = (W \times (cl(\psi) \cup \{p_\top\}) \times \{\forall, \exists\}) \cup \{q_0\}$, where $cl(\psi)$ denotes the set of ψ's subformulas. Intuitively, when the automaton is in state $\langle w, \varphi, \forall \rangle$, it accepts all strategies for which w is either pruned or satisfies φ, where $\varphi = p_\top$ is satisfied iff the root of the strategy is labeled $\top$. When the automaton is in state $\langle w, \varphi, \exists \rangle$, it accepts all strategies for which w is not pruned and it satisfies φ. We call $\forall$ and $\exists$ the *mode* of the state. While the states in $W \times \{p_\top\} \times \{\forall, \exists\}$ check that the composition of M with the strategy in the input is deadlock free, the states in $W \times cl(\psi) \times \{\forall, \exists\}$ check that this composition satisfies ψ. The initial state q_0 sends copies to check both the deadlock freeness of the composition and the satisfaction of ψ.
- The transition function $\delta : Q \times \Sigma \to \mathcal{B}^+(2^K \times Q)$ is defined as follows (with $m \in \{\exists, \forall\}$).
  - $\delta(q_0, \bot) =$ **false**, and $\delta(q_0, \top) = \delta(\langle w_0, p_\top, \exists \rangle, \top) \wedge \delta(\langle w_0, \psi, \exists \rangle, \top)$.
  - For all w and φ, we have $\delta(\langle w, \varphi, \forall \rangle, \bot) =$ **true** and $\delta(\langle w, \varphi, \exists \rangle, \bot) =$ **false**.
  - $\delta(\langle w, p_\top, m \rangle, \top) =$
    $$(\bigvee_{v \in 2^K} \bigvee_{w' \in s(w,v)} (v, \langle w', p_\top, \exists \rangle)) \wedge (\bigwedge_{v \in 2^K} \bigwedge_{w' \in s(w,v)} (v, \langle w', p_\top, \forall \rangle)).$$
  - $\delta(\langle w, p, m \rangle, \top) =$ **true** if $p \in L(w)$, and $\delta(\langle w, p, m \rangle, \top) =$ **false** if $p \notin L(w)$.
  - $\delta(\langle w, \neg p, m \rangle, \top) =$ **true** if $p \notin L(w)$, and $\delta(\langle w, \neg p, m \rangle, \top) =$ **false** if $p \in L(w)$.
  - $\delta(\langle w, \varphi_1 \wedge \varphi_2, m \rangle, \top) = \delta(\langle w, \varphi_1, m \rangle, \top) \wedge \delta(\langle w, \varphi_2, m \rangle, \top)$.
  - $\delta(\langle w, \varphi_1 \vee \varphi_2, m \rangle, \top) = \delta(\langle w, \varphi_1, m \rangle, \top) \vee \delta(\langle w, \varphi_2, m \rangle, \top)$.
  - $\delta(\langle w, AX\varphi, m \rangle, \top) = \bigwedge_{v \in 2^K} \bigwedge_{w' \in s(w,v)} (v, \langle w', \varphi, \forall \rangle)$.
  - $\delta(\langle w, EX\varphi, m \rangle, \top) = \bigvee_{v \in 2^K} \bigvee_{w' \in s(w,v)} (v, \langle w', \varphi, \exists \rangle)$.
  - $\delta(\langle w, A\varphi_1 U \varphi_2, m \rangle, \top) =$
    $$\delta(\langle w, \varphi_2, m \rangle, \top) \vee (\delta(\langle w, \varphi_1, m \rangle, \top) \wedge \bigwedge_{v \in 2^K} \bigwedge_{w' \in s(w,v)} (v, \langle w', A\varphi_1 U \varphi_2, \forall \rangle)).$$
  - $\delta(\langle w, E\varphi_1 U \varphi_2, m \rangle, \top) =$
    $$\delta(\langle w, \varphi_2, m \rangle, \top) \vee (\delta(\langle w, \varphi_1, m \rangle, \top) \wedge \bigvee_{v \in 2^K} \bigvee_{w' \in s(w,v)} (v, \langle w', E\varphi_1 U \varphi_2, \exists \rangle)).$$
  - $\delta(\langle w, AG\varphi, m \rangle, \top) = \delta(\langle w, \varphi, m \rangle, \top) \wedge \bigwedge_{v \in 2^K} \bigwedge_{w' \in s(w,v)} (v, \langle w', AG\varphi, \forall \rangle)$.
  - $\delta(\langle w, EG\varphi, m \rangle, \top) = \delta(\langle w, \varphi, m \rangle, \top) \wedge \bigvee_{v \in 2^K} \bigvee_{w' \in s(w,v)} (v, \langle w', EG\varphi, \exists \rangle)$.

  Consider, for example, a transition from the state $\langle w, AX\varphi, \exists \rangle$. First, if the transition to w is disabled (that is, the automaton reads $\bot$), then, as the current mode is existential, the run is rejecting. If the transition to w is enabled, then w's successors that are enabled should satisfy φ. The state w may have several successors that agree on some labeling $v \in 2^K$ and differ only on the labeling of variables in H. These successors are indistinguishable by the environment, and the automaton sends them all to the same direction v. This guarantees that either all these successors are enabled by the strategy (in case the letter to be read in direction v is $\top$) or all are disabled (in case the letter in direction v is $\bot$). In addition, since the requirement to satisfy φ concerns only successors of w that are enabled, the mode of the new states is universal. The copies of $\mathcal{A}_{M,\psi}$ that check the composition with the strategy to be deadlock free guarantee that at least one successor of w is enabled. Note that as the transition relation R is total, the conjunctions and disjunctions in δ cannot be empty.

- $\alpha = W \times G(\psi) \times \{\exists, \forall\}$, where $G(\psi)$ is the set of all formulas of the form $AG\varphi$ or $EG\varphi$ in $cl(\psi)$. Thus, while the automaton cannot get trapped in states associated with "Until-formulas" (then, the eventuality of the until is not satisfied), it may get trapped in states associated with "Always-formulas" (then, the safety requirement is never violated).

We now consider the size of $\mathcal{A}_{M,\neg\psi}$. Clearly, $|Q| = O(|W| * |\psi|)$. Also, as the transition associated with a state $\langle w, \varphi, m\rangle$ depends on the successors of w, we have that $|\delta| = O(|R| * |\psi|)$. Finally, $|\alpha| \leq |Q|$, and we are done. □

Extending the alternating automata described in [BVW94] to handle incomplete information is possible thanks to the special structure of the automata, which alternate between universal and existential modes. This structure (the "hesitation condition", as called in [BVW94]) exists also in automata associated with CTL* formulas, and imply the following analogous theorem.

Theorem 5. *Given a module M and a CTL* formula ψ over the sets I, O, and H, of M's variables, there exists an alternating Rabin tree automaton $\mathcal{A}_{M,\psi}$ over $\{\top, \bot\}$-labeled $2^{I \cup O}$-trees, with $|W| * 2^{O(|\psi|)}$ states and two pairs, such that $\mathcal{L}(\mathcal{A}_{M,\psi})$ is exactly the set of strategies $\mathcal{E}$ such that $M \lhd \mathcal{E}$ is deadlock free and 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 handle here modules augmented with fairness conditions.

We now consider the complexity bounds that follow from our algorithm.

Theorem 6. *The module-checking problem with incomplete information is EXPTIME-complete for CTL and is 2EXPTIME-complete for CTL*.*

Proof (sketch): The lower bounds follows from the known bounds for module checking with complete information [KV96]. For the upper bounds, in Theorems 4 and 5 we reduced the problem $M \models_r \psi$ to the problem of checking the nonemptiness of the automaton $\mathcal{A}_{M,\neg\psi}$. When ψ is a CTL formula, $\mathcal{A}_{M,\neg\psi}$ is an alternating Büchi automaton of size $O(|M| * |\psi|)$. By [VW86, MS95], checking the nonemptiness of $\mathcal{A}_{M,\neg\psi}$ is then exponential in the sizes of M and ψ. When ψ is a CTL* formula, the automaton $\mathcal{A}_{M,\neg\psi}$ is an alternating Rabin automaton, with $|W| * 2^{O(|\psi|)}$ states and two pairs. Accordingly, by [EJ88, MS95], checking the nonemptiness of $\mathcal{A}_{M,\neg\psi}$ is exponential in $|W|$ and double exponential in $|\psi|$. □

By Lemma 2, the bounds above hold also for the problem of assume-guarantee module checking with incomplete information. As the module-checking problem for CTL is already EXPTIME-hard for environments with complete information, it might seem as if incomplete information can be handled at no cost. This is, however, not true. Let us define the *program complexity* of module checking as the complexity of the problem in terms of the size of the system, assuming that the specification is fixed [VW86]. Since the system is typically much bigger than the specification, this complexity is of particular interest [LP85]. By [KV96], the program complexity of CTL module checking with complete information is PTIME-complete. On the other hand, the time complexity of the algorithm we present here is exponential in the size of the both the formula and the system. Can we do better? In Theorem 7 below, we answer this question negatively. To see why, consider a module M with hidden variables. When M interacts with an environment $\mathcal{E}$, the module seen by $\mathcal{E}$ is different from M. Indeed, every state of the module seen by $\mathcal{E}$ corresponds to a set of states of M. Therefore, coping with incomplete information involves some subset construction, which blows-up the state space exponentially. In our algorithm, the subset construction hides in the emptiness test of $\mathcal{A}_{M,\neg\psi}$.

Theorem 7. *The program complexity of CTL module checking with incomplete information is EXPTIME-complete.*

Proof (sketch): The upper bound follows from Theorem 6. For the lower bound, we do a reduction from the outcome problem for two-players games with incomplete information, proved to be EXPTIME-hard in [Rei84]. A two-player game with incomplete information consists of an AND-OR graph with an initial state and a set of designated states. Each of the states in the graph is labeled by readable and unreadable observations. The game is played between two players, called the OR-player and the AND-player. The two players generate together a path in the graph. The path starts at the initial state. Whenever the game is at an OR-state, the OR-player determines the next state. Whenever the game is at an AND-state, the AND-player determines the next state. The outcome problem is to determine whether the OR-player has a strategy that depends only on the readable observations (that is, a strategy that maps finite sequences of sets of readable observations to a set of known observations) such that following this strategy guarantees that, no matter how the AND-player plays, the path eventually visits one of the designated states.

Given an AND-OR graph G as above, we define a module M_G such that M_G reactively satisfies a fixed CTL formula φ iff the OR-player has no strategy as above. The environments of M_G correspond to strategies for the OR-player. Each environment suggests a pruning of $\langle T_{M_G}, V_{M_G} \rangle$ such that the set of paths in the pruned tree corresponds to a set of paths that the OR-player can force the game into, no matter how the AND-player plays. The module M_G is very similar to G, and the formula φ requires the existence of a computation that never visits a designated state. The formal definition of M_G and φ involves some technical complications required in order to make sure that the environment disables only transitions from OR-states. □

4 Discussion

Module checking considers the verification of open systems. In [KV96], we claim that the complexity of the module-checking problem, which is EXPTIME for specifications in CTL and only PSPACE for specifications in LTL, questions the traditional belief of the computational superiority of the branching-time paradigm. In this paper we considered open systems that have internal variables. In this common case, the environment has incomplete information about the system, and the module-checking problem should be revised accordingly. We showed that incomplete information makes CTL module checking even harder, while it comes at no cost for linear (and universal) logics. Hence, it provides an additional evidence that checking CTL properties is actually harder than checking LTL properties.

The setting we consider here is more general than the one in [KV96], but can still be generalized further. In both [KV96] and here, we assume that an environment may disable some of the system's transition. More general settings allow more dominant environments. For example, if we consider environments that are modules, then a composition of a system with an environment may not only disable some of the system's transitions, but also add new transitions (e.g., the environment may cause a certain transition of the system to branch into two transitions, each leading to a state with different assignments to the environment's variables). As in module checking, while verification of universal properties in these settings can be done using closed-system verification methods, there is a need to revise verification methods in order to handle non-universal properties.

Acknowledgment We thank Rajeev Alur for referring us to [Rei84] and pointing its relevance to the lower bound in Theorem 7.

References

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

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. LP*, LNCS 131, pp. 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*, pp. 240–248, August 1986.

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

[EJ88] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th FOCS*, pp. 368–377, October 1988.

[EL85] E.A. Emerson and C.-L. Lei. Temporal model checking under generalized fairness constraints. In *Proc. 18th Hawaii International Conference on System Sciences*, Hawaii, 1985.

[Eme85] E.A. Emerson. Automata, tableaux, and temporal logics. In *Proc. LP*, LNCS 193, pp. 79–87, 1985.

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

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

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

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

[Jon83] C.B. Jones. Specification and design of (parallel) programs. In *Proc. 9th IFIP*, pp. 321–332, North-Holland, 1983.

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

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

[Lam83] L. Lamport. Specifying concurrent program modules. *ACM Trans. on Programming Languages and Systenms*, 5:190–222, 1983.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th POPL*, pp. 97–107, January 1985.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Proc. 2nd IJCAI*, British Computer Society, pp. 481–489, September 1971.

[MP92] Z. Manna and A. Pnueli. Temporal specification and verification of reactive modules. 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.

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

[Pnu85] A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In *Proc. Advanced School on Current Trends in Concurrency*, LNCS 224, pp. 510–584, 1985.

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

[Rei84] J.H. Reif. The complexity of two-player games of incomplete information. *J. on Computer and System Sciences*, 29:274–301, 1984.

[Var95] M.Y. Vardi. On the complexity of modular model checking. In *Proc. 10th LICS*, June 1995.

[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.