An Automata-Theoretic Approach to Modular Model Checking

ORNA KUPFERMAN
Hebrew University
and
MOSHE Y. VARDI
Rice University

In *modular verification* the specification of a module consists of two parts. One part describes the guaranteed behavior of the module. The other part describes the assumed behavior of the system in which the module is interacting. This is called the *assume-guarantee* paradigm. In this paper we consider assume-guarantee specifications in which the guarantee is specified by branching temporal formulas. We distinguish between two approaches. In the first approach, the assumption is specified by branching temporal formulas too. In the second approach, the assumption is specified by linear temporal logic. We consider guarantees in $\forall$CTL and $\forall$CTL*, the universal fragments of CTL and CTL*, and assumptions in LTL, $\forall$CTL, and $\forall$CTL*. We develop two fundamental techniques: building maximal models for $\forall$CTL and $\forall$CTL* formulas and using alternating automata to obtain space-efficient algorithms for fair model checking. Using these techniques we classify the complexity of satisfiability, validity, implication, and modular verification for $\forall$CTL and $\forall$CTL*. We show that modular verification is PSPACE-complete for $\forall$CTL and is EXPSPACE-complete for $\forall$CTL*. We prove that when the assumption is linear, these bounds hold also for guarantees in CTL and CTL*. On the other hand, the problem remains EXPSPACE-hard even when we restrict the assumptions to LTL and take the guarantee as a fixed $\forall$CTL formula.

Categories and Subject Descriptors: []:

General Terms: Algorithms, Verification

Additional Key Words and Phrases: Automata, modular verification, temporal logic

1. INTRODUCTION

Temporal logics, which are modal logics geared toward the description of the temporal ordering of events, have been adopted as a powerful tool for specifying and verifying concurrent programs [Pnueli 1977; 1981]. One of the most significant developments in this area is the discovery of algorithmic methods for verifying temporal-logic properties of *finite-state* programs [Clarke et al. 1986; Lichtenstein

This paper is based on "On the complexity of modular model checking", by M.Y. Vardi, *Proc. 10th IEEE Symposium on Logic in Computer Science*, June 1995, pp. 101–111, and "On the complexity of branching modular model checking", by O. Kupferman and M.Y. Vardi, *Proc. 6th International Conf. on Concurrency Theory* (CONCUR'95), Aug. 1995, Springer-Verlag, Lecture Notes in Computer Science 962, pp. 408–422.
Author's addresses: O. Kupferman, Institute of Computer Science, Hebrew University, Jerusalem 91904, Israel; e-mail: orna@cs.huji.ac.il URL:www.cs.huji.ac.il/~orna. M.Y. Vardi, Department of Computer Science, Rice University, Houston, TX 77251-1892, U.S.A.; e-mail:vardi@cs.rice.edu URL:www.cs.rice.edu/~vardi. Supported in part by the NSF grants CCR-9628400 and CCR-9700061, and by a grant from the Intel Corporation

and Pnueli 1985; Queille and Sifakis 1981]. This derives its significance both from the fact that many synchronization and communication protocols can be modeled as finite-state programs, as well as from the great ease of use of fully algorithmic methods. Finite-state programs can be modeled by transition systems where each state has a bounded description, and hence can be characterized by a fixed number of boolean atomic propositions. This means that a finite-state program can be viewed as a finite *propositional Kripke structure*, and its properties can be specified using *propositional* temporal logic. Thus, to verify the correctness of the program with respect to a desired behavior, one only has to check that the program, modeled as a finite Kripke structure, satisfies (is a model of) the propositional temporal logic formula that specifies that behavior. Hence the name *model checking* for the verification methods derived from this viewpoint [Clarke et al. 1999].

We distinguish between two types of temporal logics: linear and branching [Lamport 1980]. In linear temporal logics, each moment in time has a unique possible future, while in branching temporal logics, each moment in time may split into several possible futures. The complexity of model checking for both linear and branching temporal logics is well understood: suppose we are given a program of size n and a temporal logic formula of size m. For a branching temporal logic such as CTL, model-checking algorithms run in time $O(nm)$ [Clarke et al. 1986], while, for linear temporal logic such as LTL, model-checking algorithms run in time $n2^{O(m)}$ [Lichtenstein and Pnueli 1985]. Since model checking with respect to a linear temporal logic formula is PSPACE-complete [Sistla and Clarke 1985], the latter bound probably cannot be improved. The difference in the complexity of linear and branching model checking has been viewed as an argument in favor of the branching paradigm.

Model checking suffers, however, from the so-called *state-explosion* problem. In a concurrent setting, the program under consideration is typically the parallel composition of many modules. As a result, the size of the state space of the program can be the product of the sizes of the state spaces of the participating modules. This gives rise to state spaces of exceedingly large sizes, which makes even linear-time algorithms impractical. This issue is one of the most important in the area of computer-aided verification and is the subject of active research (cf. Burch et al. [1990]).

Modular verification is one possible way to address the state-explosion problem, cf. Clarke et al. [1989], Aziz et al. [1994], and Kurshan [1987]. In modular verification, one uses proof rules of the following form:

$$\left.\begin{array}{l} M_1 \models \psi_1 \\ M_2 \models \psi_2 \\ C(\psi_1, \psi_2, \psi) \end{array}\right\} M_1 \| M_2 \models \psi$$

Here, $M \models \theta$ means that the module M satisfies the formula θ, the symbol "$\|$" denotes parallel composition, and $C(\psi_1, \psi_2, \psi)$ is some logical condition relating ψ_1, ψ_2, and ψ. Using modular proof rules enables one to apply model checking only to the underlying modules, which have much smaller state spaces.

The state-explosion problem is only one motivation for pursuing modular verification. Modular verification is advocated also for other methodological reasons; a robust verification methodology should provide rules for deducing properties of

programs from the properties of their constituent modules. Indeed, efforts to develop modular verification frameworks were undertaken in the mid 1980s [Pnueli 1985b].

A key observation (see Lamport [1983], Jones [1983], Misra and Chandy [1981], Stark [1984], and Pnueli [1985b]) is that in modular verification the specification should include two parts. One part describes the desired behavior of the module. The other part describes the assumed behavior of the system within which the module is interacting. This is called the *assume-guarantee* paradigm, as the specification describes what behavior the module is *guaranteed* to exhibit, *assuming* that the system behaves in the promised way.

For the linear temporal paradigm, an assume-guarantee specification is a pair $\langle \varphi, \psi \rangle$, where both φ and ψ are linear temporal logic formulas. The meaning of such a pair is that all the computations of the module are guaranteed to satisfy ψ, assuming that all the computations of the environment satisfy φ. As observed in Pnueli [1985b], in this case the assume-guarantee pair $\langle \varphi, \psi \rangle$ can be combined to a single linear temporal logic formula $\varphi \rightarrow \psi$ (see also Jonsson and Tsay [1995]). Thus, model checking a module with respect to assume-guarantee specifications in which both the assumed and the guaranteed behaviors are linear temporal logic formulas is essentially the same as model checking the module with respect to linear temporal logic formulas.

The situation is different for the branching temporal paradigm. Here the guarantee is a branching temporal formula, which describes the computation tree of the module. There are two approaches, however, to the assumptions in assume-guarantee pairs. The first approach, implicit in Clarke et al. [1986] and Emerson and Lei [1985; 1987], and made explicit in Josko [1987a; 1987b; 1989], and Damm et al. [1989], is that the assumption in the assume-guarantee pair concerns the interaction of the module with its environment along each computation, and is therefore more naturally expressed in linear temporal logic. Thus, in this approach, an assume-guarantee pair should consist of a *linear* temporal assumption φ and a *branching* temporal guarantee ψ. The meaning of such a pair is that ψ holds in the computation tree that consists of all computations of the program that satisfy φ. The problem of verifying that a given module M satisfies such a pair $\langle \varphi, \psi \rangle$, which we call the *linear-branching modular model-checking problem*, is more general than either linear or branching model checking.

A second approach was considered in Grumberg and Long [1994], where assumptions are taken to apply to the computation tree of the system within which the module is interacting. Accordingly, assumptions in Grumberg and Long [1994] are also expressed in branching temporal logic. There, a module M satisfies an assume-guarantee pair $\langle \varphi, \psi \rangle$ if and only if whenever M is part of a system satisfying φ, the system satisfies ψ too. We call this *branching modular model checking*. Furthermore, it is argued there, as well as in Damm et al. [1989], Josko [1989], Grumberg and Long [1991], and Dams et al. [1993], that in the context of modular verification it is advantageous to use only *universal* branching temporal logic, i.e., branching temporal logic without existential path quantifiers. That is, in a universal branching temporal logic one can state properties of all computations of a program, but one cannot state that certain computations exist. Consequently, universal branching temporal logic formulas have the helpful property that once they are satisfied

in a module, they are satisfied also in a system that contains this module. The focus in Grumberg and Long [1994] is on using ∀CTL, the universal fragment of CTL, for both the assumption and the guarantee.

In this paper, we focus on the branching modular model-checking problem, which we show to be a proper extension of the linear-branching modular model-checking problem. We consider assumptions and guarantees in both ∀CTL and in the more expressive ∀CTL*. We start by examining the most fundamental questions about these logics: *satisfiability*, *validity*, and *implication* (since ∀CTL and ∀CTL* are not closed under negation, these problems are not interreducible as they are for CTL and CTL*). Solving the satisfiability and validity problems enables us to check that a specification is not trivially true or false. Implication generalizes these questions and is strongly related to modular model checking. Recall that in the linear framework, checking an assume-guarantee pair $\langle \varphi, \psi \rangle$ in a module M is equivalent to checking whether M satisfies the implication $\varphi \rightarrow \psi$. In the branching framework, we show here that an assume-guarantee pair $\langle \varphi, \psi \rangle$ holds in a universal module that exhibits all possible behaviors if and only if the implication $\varphi \rightarrow \psi$ is valid.

We use two fundamental techniques to solve these questions. The first technique is the *maximal-model* technique introduced in Grumberg and Long [1994]. It is shown there that with every ∀CTL formula φ one can associate a *maximal model* M_φ (called the *tableau* of φ in Grumberg and Long [1994]) such that a module M satisfies φ precisely when M *simulates* M_φ (we define simulation later on). We use here automata-theoretic techniques for CTL* [Vardi and Stockmeyer 1985; Emerson and Jutla 1988] to construct maximal models for ∀CTL* formulas. While maximal models for ∀CTL involve an exponential blow-up, maximal models for ∀CTL* involve a doubly exponential blow-up.

The second technique is the automata-theoretic framework to branching-time model checking introduced in Kupferman et al. [2000]. It is shown there how to use *alternating* tree automata to obtain space-efficient model checking methods. Since the maximal models that we construct include fairness conditions, we extend the automata-theoretic method of Kupferman et al. [2000] to yield space-efficient *fair* model-checking algorithms. We then show how performing fair model checking over maximal models can solve the satisfiability, validity, and implication problems for ∀CTL and ∀CTL*. Our results show that these problems are computationally easier than the analogous problems for CTL and CTL*. For example, while all three problems are EXPTIME-complete for CTL, we have that satisfiability and implication are PSPACE-complete and validity is NP-complete for ∀CTL.

By relating the implication problem with the branching modular model-checking problem, we show that the same two fundamental techniques of maximal models and space-efficient fair model checking also yield a solution to the latter problem. We prove that the problem is PSPACE-complete for ∀CTL and is EXPSPACE-complete for ∀CTL*. We show that the increase in complexity is solely due to the assumption part of the specification. This suggests that modular model checking in the branching temporal framework can be practical only for very small assumptions.

We turn on to investigate the linear-branching modular model-checking problem. Recall that the problem is a special case of the branching modular model-checking problem. While this suggests an EXPSPACE upper bound for the problem, which

we prove to be tight, we show that the linear-branching setting is essentially simpler. Using the automata-theoretic framework for linear temporal logics [Vardi and Wolper 1986], we describe an algorithm for the linear-branching modular model-checking problem that allows the branching guarantee to be any CTL or CTL* formula (that is, it is not restricted to their universal fragments) and avoids the construction of maximal models. An even simpler construction, which avoids the use of determinization, is described in Vardi [1995].

2. PRELIMINARIES

2.1 The Temporal Logics LTL, CTL*, and CTL

The logic LTL is a linear temporal logic. Formulas of LTL are built from a set AP of atomic proposition using the usual Boolean operators and the temporal operators X ("next time"), U ("until"), and $\tilde{U}$ ("duality of until"). We present here a positive normal form in which negation may be applied only to atomic propositions. Given a set AP, an LTL formula is defined as follows:

—**true**, **false**, p, or $\neg p$, for $p \in AP$.
—$\psi \vee \varphi$, $\psi \wedge \varphi$, $X\psi$, $\psi U \varphi$, or $\psi \tilde{U} \varphi$, where ψ and φ are LTL formulas.

We define the semantics of LTL with respect to a *computation* $\pi = \sigma_0, \sigma_1, \sigma_2, \ldots$, where for every $j \geq 0$, we have that σ_j is a subset of AP, denoting the set of atomic propositions that hold in the jth position of π. We denote the suffix $\sigma_j, \sigma_{j+1}, \ldots$ of π by π^j. We use $\pi \models \psi$ to indicate that an LTL formula ψ holds in the path π. The relation $\models$ is inductively defined as follows:

—For all π, we have that $\pi \models$ **true** and $\pi \not\models$ **false**.
—For an atomic proposition $p \in AP$, we have $\pi \models p$ if and only if $p \in \sigma_0$ and $\pi \models \neg p$ if and only if $p \notin \sigma_0$.
—$\pi \models \psi \vee \varphi$ if and only if $\pi \models \psi$ or $\pi \models \varphi$.
—$\pi \models \psi \wedge \varphi$ if and only if $\pi \models \psi$ and $\pi \models \varphi$.
—$\pi \models X\psi$ if and only if $\pi^1 \models \psi$.
—$\pi \models \psi U \varphi$ if and only if there exists $k \geq 0$ such that $\pi^k \models \varphi$ and $\pi^i \models \psi$ for all $0 \leq i < k$.
—$\pi \models \psi \tilde{U} \varphi$ if and only if for every $k \geq 0$ for which $\pi^k \not\models \varphi$, there exists $0 \leq i < k$ such that $\pi^i \models \psi$.

We denote the length of a formula φ by $|\varphi|$, and we use the following abbreviations in writing formulas:

—$\rightarrow$ and $\leftrightarrow$, interpreted in the usual way (that is $\psi \rightarrow \varphi = \varphi \vee \neg\psi$, and $\psi \leftrightarrow \varphi = \psi \rightarrow \varphi \wedge \varphi \rightarrow \psi$).
—$F\psi =$ **true**$U\psi$ ("eventually").
—$G\psi =$ **false**$\tilde{U}\psi$ ("always").

The logic $CTL^\star$ is a branching temporal logic. A path quantifier, E ("for some path") or A ("for all paths"), can prefix an assertion composed of an arbitrary combination of linear time operators. There are two types of formulas in CTL*: *state formulas*, whose satisfaction is related to a specific state, and *path formulas*,

whose satisfaction is related to a specific path. Formally, let AP be a set of atomic proposition names. A CTL* state formula (again, in a positive normal form) is either

—**true**, **false**, p or $\neg p$, for $p \in AP$,

—$\psi \vee \varphi$ or $\psi \wedge \varphi$ where ψ and φ are CTL* state formulas, or

—$E\psi$ or $A\psi$, where ψ is a CTL* path formula.

A CTL* path formula is either

—A CTL* state formula, or

—$\psi \vee \varphi$, $\psi \wedge \varphi$, $X\psi$, $\psi U \varphi$, or $\psi \tilde{U} \varphi$, where ψ and φ are CTL* path formulas.

The logic CTL* consists of the set of state formulas generated by the above rules. The logic CTL is a restricted subset of CTL*. In CTL, the temporal operators X, U, and $\tilde{U}$ must be immediately preceded by a path quantifier. Formally, it is the subset of CTL* obtained by restricting the path formulas to be $X\psi$, $\psi U \varphi$, or $\psi \tilde{U} \varphi$, where ψ and φ are CTL state formulas.

The logic $\forall CTL^\star$ is a restricted subset of CTL* that allows only the universal path quantifier A. Note that since negation in CTL* can be applied only to atomic propositions, assertions of the form $\neg A\psi$, which is equivalent to $E\neg\psi$, are not possible. Thus, the logic $\forall$CTL* is not closed under negation. The logic $\forall$CTL is defined similarly, as the restricted subset of CTL that allows the universal path quantifier only. The logics $\exists$CTL* and $\exists$CTL are defined analogously, as the existential fragments of CTL* and CTL, respectively. Note that negating a $\forall$CTL* formula results in an $\exists$CTL* formula. For example, $\neg ApU(AXq)$ is equivalent to $E(\neg p)\tilde{U}(EX\neg q)$. Conversely, negating a $\exists$CTL* formula results in an $\forall$CTL* formula.

The *closure* $cl(\psi)$ of a CTL* formula ψ is the set of all state subformulas of ψ (including ψ but excluding **true** and **false**). For example, $cl(E(pU(AXq))) = \{E(pU(AXq)), p, AXq, q\}$. It is easy to see that the size of $cl(\psi)$ is linear in the size of ψ. We say that a CTL* formula φ is a *U-formula* if it is of the form $A\varphi_1 U \varphi_2$ or $E\varphi_1 U \varphi_2$. The subformula φ_2 is then called the *eventuality* of φ. Similarly, φ is a *$\tilde{U}$-formula* if it is of the form $A\varphi_1 \tilde{U} \varphi_2$ or $E\varphi_1 \tilde{U} \varphi_2$. We denote by $AU(\psi)$ the set of formulas of the from $A\varphi_1 U \varphi_2$ in $cl(\psi)$. The sets $EU(\psi)$, $A\tilde{U}(\psi)$, and $E\tilde{U}(\psi)$ are defined similarly.

We define the semantics of CTL* (and its sublanguages) with respect to *fair Rabin modules* (*fair modules*, for short). A fair module $M = \langle AP, W, R, W_0, L, \alpha \rangle$ consists of a set AP of atomic propositions, a set W of states, a transition relation $R \subseteq W \times W$, a set $W_0 \subseteq W$ of initial states, a labeling function $L : W \rightarrow 2^{AP}$, and a Rabin fairness condition $\alpha \subseteq 2^Q \times 2^Q$ that defines a subset of W^ω (the exact semantics of α is given shortly). Our choice of this type of fairness condition is technically motivated, as will be clarified in the sequel. For a state $w \in W$, we use $bd(w)$ to denote the branching degree of w, i.e., the number of different R-successors that w has.

A *computation* of a fair module is an infinite sequence of states, $\pi = w_0, w_1, \ldots$ such that for every $i \geq 0$, we have that $\langle w_i, w_{i+1} \rangle \in R$. We extend the labeling function L to computations and denote by $L(\pi)$ the word $L(w_0) \cdot L(w_1) \cdots$. For a computation π, let $inf(\pi)$ denote the set of states that repeat infinitely often in π,

i.e.,

$$inf(\pi) = \{w : \text{ for infinitely many } i \geq 0, \text{ we have } w_i = w\}.$$

A computation of M is *fair* if and only if it satisfies the fairness condition α. Thus, if $\alpha = \{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$, then π is fair if and only if there exists $1 \leq i \leq k$ such that $inf(\pi) \cap G_i \neq \emptyset$ and $inf(\pi) \cap B_i = \emptyset$, in other words, if and only if π visits G_i infinitely often and visits B_i only finitely often. We say that a fair module is *nonempty* if and only if there exists a fair computation that starts at an initial state. A *module* is a fair module with no fairness condition. That is, all the computations of a module are considered fair. We denote a module by $M = \langle AP, W, R, W_0, L\rangle$. We define the *size* $|M|$, of a fair module or a module M as above as $|W|+|R|+|W_0|$. Thus, the definition of $|M|$ ignores the labeling function and the fairness condition. When the module is fair, we refer to the number of pairs in α as the *degree* of M.

We use $w \models \varphi$ to indicate that a state formula φ holds at state w (assuming an agreed fair module M). The relation $\models$ is inductively defined as follows (the relation $\pi \models \psi$ for a path formula ψ is the same as for ψ in LTL).

—For all w, we have that $w \models \mathbf{true}$ and $w \not\models \mathbf{false}$.

—For an atomic proposition $p \in AP$, we have $w \models p$ if and only if $p \in L(w)$ and $w \models \neg p$ if and only if $p \notin L(w)$.

—$w \models \psi \vee \varphi$ if and only if $w \models \psi$ or $w \models \varphi$.

—$w \models \psi \wedge \varphi$ if and only if $w \models \psi$ and $w \models \varphi$.

—$w \models E\psi$ if and only if there exists a fair computation $\pi = w_0, w_1, \ldots$ such that $w_0 = w$ and $\pi \models \psi$.

—$w \models A\psi$ if and only if for all fair computations $\pi = w_0, w_1, \ldots$ such that $w_0 = w$, we have $\pi \models \psi$.

—$\pi \models \varphi$ for a computation $\pi = w_0, w_1, \ldots$ and a state formula φ if and only if $w_0 \models \varphi$.

A fair module M satisfies a formula φ if and only if φ holds in *all* initial states of M. The problem of determining whether M satisfies φ is the *fair model-checking* problem.

For technical convenience, we assume that the relation R is total; thus every state has at least one successor. We can ensure that R is total by adding to W a sink state w_{sink} labeled by some atomic proposition *sink* not in AP, and adding transitions from all states to w_{sink}. The atomic proposition *sink* then enables us to exclude computations that are not "real" computations of the original module. For example, by replacing an existential formula $E\psi$, for a path formula ψ, by the formula $E(\psi \wedge G\neg sink)$, we can make sure that the property ψ is satisfied along a computation of the original module. For a detailed description of this transformation see the treatment of the atomic proposition $\top$ in Section 3.2, where we restrict path quantification to computations in which $\top$ always holds. In particular, as noted there, the transformation involves only a linear blow up, and it can be enhanced (still with only a linear blow up) so that if the formula we start with is a CTL formula, so is the formula we end up with.

2.2 Simulation Relation and Composition of Modules

In the context of modular verification, it is helpful to define an order relation between fair modules [Grumberg and Long 1994]. Intuitively, the order captures what it means for a fair module M' to have "more behaviors" than a fair module M. Let $M = \langle AP, W, R, W_0, L, \alpha \rangle$ and $M' = \langle AP', W', R', W'_0, L', \alpha' \rangle$ be two fair modules for which $AP' \subseteq AP$, and let w and w' be states in W and W', respectively. A relation $H \subseteq W \times W'$ is a *simulation relation* from $\langle M, w \rangle$ to $\langle M', w' \rangle$ if and only if the following conditions hold:

(1) $H(w, w')$.

(2) For all s and s', we have that $H(s, s')$ implies the following:
   (a) $L(s) \cap AP' = L(s')$.
   (b) For every fair computation $\pi = s_0, s_1, \ldots$ in M, with $s_0 = s$, there exists a fair computation $\pi' = s'_0, s'_1, \ldots$ in M', with $s'_0 = s'$, such that for all $i \geq 0$, we have $H(s_i, s'_i)$.

A simulation relation H is a *simulation from M to M'* if and only if for every $w \in W_0$ there exists $w' \in W'_0$ such that H is a simulation relation from $\langle M, w \rangle$ to $\langle M', w' \rangle$. If there exists a simulation from M to M', we say that M *simulates* M' and we write $M \leq M'$. Intuitively, it means that the fair module M' has more behaviors than the fair module M. In fact, every possible behavior of M is also a possible behavior of M'. Note that our simulation is an extension of the classical simulation used by Milner [1971], where there are no fairness conditions. We sometimes relate modules (with no fairness condition) with $\leq$. Then, we assume that all computations are fair, and that the relation that follows coincides with the one in Milner [1971].

THEOREM 2.1. [Grumberg and Long 1994]

(1) The simulation relation $\leq$ is a preorder (i.e., a reflexive and transitive order).

(2) For every M and M' such that $M \leq M'$, and for every universal branching temporal logic formula φ, $M' \models \varphi$ implies $M \models \varphi$.

Let M and M' be two modules. The *composition* of M and M', denoted $M \| M'$, is a module that has exactly these behaviors that are joint to M and M'. Formally, let $M = \langle AP, W, R, W_0, L \rangle$ and $M' = \langle AP', W', R', W'_0, L' \rangle$. We assume that $W \cap W' = \emptyset$ (otherwise, we rename one of the state spaces). Then, $M \| M' = \langle AP'', W'', R'', W''_0, L'' \rangle$, where,

—$AP'' = AP \cup AP'$.
—$W'' = \{\langle w, w' \rangle : L(w) \cap AP' = L(w') \cap AP\}$.
—$R'' = \{\langle \langle w, w' \rangle, \langle s, s' \rangle \rangle : \langle w, s \rangle \in R \text{ and } \langle w', s' \rangle \in R'\} \cap (W'' \times W'')$.
—$W''_0 = (W_0 \times W'_0) \cap W''$.
—For every $\langle w, w' \rangle \in W''$, we have $L''(\langle w, w' \rangle) = L(w) \cup L'(w')$.

Note that the transition relation R'' may not be total even when both R and R' are total. Then, recall, we can make R'' total by adding a sink state.

We also define the composition of a fair module M with a module M'. Here, $M \| M'$ is a fair module that has exactly these behaviors that are joint to M

and M' and are fair in M. Formally, if $M = \langle AP, W, R, W_0, L, \alpha\rangle$ and $M' = \langle AP', W', R', W_0', L'\rangle$, then $M\|M' = \langle AP'', W'', R'', W_0'', L'', \alpha''\rangle$, where AP'', W'', R'', W_0'', and L'' are as in the composition of two modules, and

—$\alpha'' = \{\langle (G \times W') \cap W'', (B \times W') \cap W''\rangle : \langle G, B\rangle \in \alpha\}$.

It is easy to see that if M and M' have n and n' states, and M has m pairs in its fairness condition, then $M\|M'$ has nn' states and m pairs.

Note that the composition of two fair modules is not defined, as such a composition may require a fairness condition that is more elaborated than Rabin.

The following properties of compositions are proven in Grumberg and Long [1994] for fair Streett modules (modules where the fairness condition is Streett), and we prove them here for modules and fair Rabin modules.

THEOREM 2.2. *For every module M and fair Rabin modules M' and M'', the following hold.*

(1) If $M' \leq M''$ then $M\|M' \leq M\|M''$.
(2) $M \leq M\|M$.
(3) $M\|M' \leq M'$.

PROOF. The proof is very similar to the proof for Streett modules given in Grumberg and Long [1994]. We start with (1). Assume that $M' \leq M''$. Let H be a simulation from M' to M''. Let W, W_1, and W_2 be the states spaces of M, $M\|M'$, and $M\|M''$, respectively. It is easy to see that the relation $H' = \{\langle\langle w, w'\rangle, \langle w, w''\rangle\rangle : \langle w, w'\rangle \in W_1, \langle w, w''\rangle \in W_2$, and $H(w', w'')\}$ is a simulation from $M\|M'$ to $M\|M''$. In order to prove (2), recall that the state space of $M\|M$ is $W \times W$, where W is the state space of M. Therefore, it is easy to see that the relation $H = \{\langle w, \langle w, w\rangle\rangle\}$ is a simulation from M to $M\|M$. Similarly, in order to prove (3), recall that the state space of $M\|M'$ is $W \times W'$, where W and W' are the state spaces of M and M', respectively. Therefore, it is easy to see that the relation $H = \{\langle\langle w, w'\rangle, w'\rangle\}$ is a simulation from $M\|M'$ to M'. □

A fair module M is a *maximal model* for an $\forall$CTL* formula φ if it allows all behaviors consistent with φ. Formally, M is a maximal model of φ if $M \models \varphi$ and for every fair module M' we have that $M' \leq M$ if $M' \models \varphi$. Note that by the preceding theorem, if $M' \leq M$, then $M' \models \varphi$. Thus, M_φ is a maximal model for φ if for every fair module M, we have that $M \leq M_\varphi$ if and only if $M \models \varphi$.

THEOREM 2.3. [Grumberg and Long 1994] *For every $\forall$CTL formula φ, there exists a maximal model M_φ of size $2^{O(|\varphi|)}$.*

We will describe later a construction of maximal models for $\forall$CTL* formulas.

2.3 Büchi Word Automata

Given an alphabet Σ, an *infinite word over* Σ is an infinite sequence $w = w_1 \cdot w_2 \cdot w_3 \cdots$ of letters in Σ. A *Büchi automaton over infinite words* is $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, F\rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is an acceptance condition (a condition that defines a subset of Q^ω). Intuitively, $\delta(q, \sigma)$ is the set of states that $\mathcal{A}$ can move into when it is in state q and it reads the letter σ. Since

$\mathcal{A}$ may have several initial states and since the transition function may specify many possible transitions for each state and letter, $\mathcal{A}$ may be *nondeterministic*. If $|Q_0| = 1$ and δ is such that for every $q \in Q$ and $\sigma \in \Sigma$, we have that $|\delta(q, \sigma)| = 1$, then $\mathcal{A}$ is a *deterministic* automaton.

Given an input infinite word $w = c_0 \cdot c_1 \cdots \in \Sigma^\omega$, a *run* of $\mathcal{A}$ on w can be viewed as a function $r : \mathbb{N} \to Q$ where $r(0) \in Q_0$ (i.e., the run starts in one of the initial states) and for every $i \geq 0$, we have $r(i+1) \in \delta(r(i), c_i)$ (i.e., the run obeys the transition function). Each run r induces a set $inf(r)$ of states that r visits *infinitely often*. Formally,

$$inf(r) = \{q \in Q \ : \text{ for infinitely many } i \in \mathbb{N}, \text{ we have } r(i) = q\}.$$

As Q is finite, it is guaranteed that $inf(r) \neq \emptyset$. The run r *accepts* w if and only if $inf(r) \cap F \neq \emptyset$. Note that a nondeterministic automaton can have many runs on w. In contrast, a deterministic automaton has a single run on w. An automaton $\mathcal{A}$ accepts an input word w if and only if there exists a run r of $\mathcal{A}$ on w such that r accepts w. The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of infinite words that $\mathcal{A}$ accepts. Thus, each word automaton defines a subset of Σ^ω.

Computations of a fair module can be viewed as infinite words over the alphabet 2^{AP}. According to this view, each fair module corresponds to a language over the alphabet 2^{AP} and can be associated with an automaton. A similar connection has been established between LTL formulas and Büchi automata:

THEOREM 2.4. [Vardi and Wolper 1994] *Given an LTL formula ψ, there is a Büchi automaton $\mathcal{A}_\psi = \langle 2^{AP}, Q, \delta, Q_0, F \rangle$, with $2^{O(|\psi|)}$ states, such that $\mathcal{L}(\mathcal{A}_\psi)$ is exactly the set of computations satisfying ψ.*

2.4 Alternating Tree Automata

An *infinite tree* is an infinite set $T \subseteq \mathbb{N}^*$ such that if $x \cdot c \in T$ where $x \in \mathbb{N}^*$ and $c \in \mathbb{N}$, then also $x \in T$, and for all $0 \leq c' < c$, we have that $x \cdot c' \in T$. The elements of T are called *nodes*, and the empty word ϵ is the *root* of T. For every $x \in T$, the nodes $x \cdot c$ where $c \in \mathbb{N}$ are the *successors* of x. The number of successors of a node x is called the *branching degree* of x and is denoted by $d(x)$. We consider here trees in which each node has at least one successor and only finitely many successors. A *path* ρ of a tree T is a prefix-closed set $\rho \subseteq T$ such that $\epsilon \in \rho$ and for every $x \in \rho$ there exists a unique $c \in \mathbb{N}$ such that $x \cdot c \in \rho$. For a path ρ and $j \geq 0$, let ρ_j denote the node of length j in ρ. Intuitively, each path $\rho \subseteq T$ induces a unique sequence, $\rho_0, \rho_1, \ldots$, in $\mathbb{N}^\omega$. Given an alphabet Σ, a Σ-*labeled tree* is a pair $\langle T, V \rangle$ where T is a tree and $V : T \to \Sigma$ maps each node of T to a letter in Σ. For a Σ-labeled tree $\langle T, V \rangle$ and a path $\rho \subseteq T$, we denote by $V(\rho)$ the sequence $V(\rho_0), V(\rho_1), \ldots$, in Σ^ω. Of special interest to us are Σ-labeled trees in which $\Sigma = 2^{AP}$ for some set AP of atomic propositions. We call such Σ-labeled trees *computation trees*. By regarding each of the nodes in a computation tree as a state in a fair module, we can view a computation tree as a fair module with infinitely many states. We sometimes refer to satisfaction of temporal logic formulas in a computation tree, meaning their satisfaction in this fair module. Given a set $\mathcal{D} \subset \mathbb{N}$, a $\mathcal{D}$-tree is a computation tree in which all the nodes have branching degrees in $\mathcal{D}$. Note that an infinite word can be viewed as a $\{1\}$-tree.

Finite automata on infinite trees (tree automata, for short) run on such Σ-labeled trees. *Alternating automata* on infinite trees generalize nondeterministic tree automata and were first introduced in Muller and Schupp [1987]. For simplicity, we refer first to automata over infinite binary trees. Consider a nondeterministic tree automaton $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, \alpha \rangle$. The transition relation δ maps an automaton state $q \in Q$ and an input letter $\sigma \in \Sigma$ to a set of pairs of states. Each such pair suggests a nondeterministic choice for the automaton's next configuration. When the automaton is in a state q and is reading a node x labeled by a letter σ, it proceeds by first choosing a pair $\langle q_1, q_2 \rangle \in \delta(q, \sigma)$ and then splitting into two copies. One copy enters the state q_1 and proceeds to the node $x \cdot 0$ (the left successor of x), and the other copy enters the state q_2 and proceeds to the node $x \cdot 1$ (the right successor of x).

For a finite set X, let $\mathcal{B}^+(X)$ be the set of positive Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$ and $\vee$), where we also allow the formulas **true** and **false** and, as usual, $\wedge$ has precedence over $\vee$. For a set $Y \subseteq X$ and a formula $\theta \in \mathcal{B}^+(X)$, we say that Y *satisfies* θ if and only if assigning **true** to elements in Y and assigning **false** to elements in $X \setminus Y$ makes θ true. We can represent δ using $\mathcal{B}^+(\{0,1\} \times Q)$. For example, $\delta(q, \sigma) = \{\langle q_1, q_2 \rangle, \langle q_3, q_1 \rangle\}$ can be written as $\delta(q, \sigma) = (0, q_1) \wedge (1, q_2) \vee (0, q_3) \wedge (1, q_1)$.

In nondeterministic tree automata, each disjunct in δ is a conjunction with exactly one element associated with each direction. In alternating automata on binary trees, $\delta(q, \sigma)$ can be an arbitrary formula from $\mathcal{B}^+(\{0,1\} \times Q)$. We can have, for instance, a transition

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

The above transition illustrates that several copies may go to the same direction and that the automaton is not required to send copies to all the directions.

Formally, a finite alternating automaton on infinite binary trees is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, \alpha \rangle$ where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{B}^+(\{0,1\} \times Q)$ is a transition function, $Q_0 \subseteq Q$ is a set of initial states, and α specifies the acceptance condition.

Generalizing alternating automata to trees where nodes can have different branching degrees, all in a finite set $\mathcal{D} \subset \mathbb{N}$, we have that the transition function is $\delta : Q \times \Sigma \times \mathcal{D} \to \mathcal{B}^+(\mathbb{N} \times Q)$ with the requirement, that for every $k \in \mathcal{D}$, we have $\delta(q, \sigma, k) \in \mathcal{B}^+(\{0, \ldots, k-1\} \times Q)$. When the automaton is in a state q as it reads a node that is labeled by a letter σ and has k successors, it applies the transition $\delta(q, \sigma, k)$. For each $q \in Q$ and $\sigma \in \Sigma$, we denote $\bigvee_{k \in \mathcal{D}} \delta(q, \sigma, k)$ by $\delta(q, \sigma)$. We define the *size* $|\mathcal{A}|$ of an automaton $\mathcal{A} = \langle \Sigma, \mathcal{D}, Q, \delta, q_0, \alpha \rangle$ as $|Q| + |\alpha| + |\delta|$, where $|Q|$ is the cardinality of the set Q, $|\alpha|$ is the sum of the cardinalities of the sets appearing in the pairs in α, and $|\delta|$ is the sum of the lengths of the nonidentically false formulas that appear as $\delta(q, \sigma, k)$ for some $q \in Q$, $\sigma \in \Sigma$, and $k \in \mathcal{D}$ (note that the restriction to nonidentically false formulas is to avoid an unnecessary $|Q| \cdot |\Sigma| \cdot |\mathcal{D}|$ minimal size for δ). Note that $\mathcal{A}$ can be encoded in space $O(|\mathcal{A}|)$.

A *run of an alternating automaton* $\mathcal{A}$ on a tree $\langle T, V \rangle$ is a tree $\langle T_r, r \rangle$ in which the root is labeled by some $q_0 \in Q_0$ and every other node is labeled by an element of $T \times Q$. Unlike T, in which each node has at least one successor, the tree T_r may have *leaves* (nodes with no successors). Thus, a path in T_r may be either finite, in

q	$\delta(q,a)$	$\delta(q,b)$	$\delta(q,c)$
q_0	$(0,q_1) \vee (1,q_1)$	$(0,q_3) \wedge (1,q_3) \wedge ((0,q_2) \vee (1,q_2))$	$(0,q_3) \wedge (1,q_3) \wedge ((0,q_1) \vee (1,q_1))$
q_1	$(0,q_1) \vee (1,q_1)$	$(0,q_2) \vee (1,q_2)$	$(0,q_1) \vee (1,q_1)$
q_2	$(0,q_1) \vee (1,q_1)$	**true**	$(0,q_1) \vee (1,q_1)$
q_3	**true**	$(0,q_3) \wedge (1,q_3)$	$(0,q_3) \wedge (1,q_3)$

Fig. 1. The transition relation of the automaton $\mathcal{A}$.

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 input tree. The labels of a node and its successors have to satisfy the transition function. Formally, $\langle T_r, r \rangle$ is a Σ_r-labeled tree where $\Sigma_r = T \times Q$ 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 $\delta(q, V(x), d(x)) = \theta$. Then there is a (possibly empty) set $S = \{(c_0,q_0),(c_1,q_1),\ldots,(c_n,q_n)\} \subseteq \{0,\ldots,d(x)-1\} \times Q$, such that the following holds:
—S satisfies θ, and
—for all $0 \leq i \leq n$, 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 binary tree with $V(\epsilon) = a$, $Q_0 = \{q_0\}$, 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 successors. 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**. A run $\langle T_r, r \rangle$ is accepting if and only if all its infinite paths satisfy the acceptance condition. As with nondeterministic automata, an automaton accepts a tree if and only if there exists a run that accepts it. We denote by $\mathcal{L}(\mathcal{A})$ the language of the automaton $\mathcal{A}$, i.e., the set of all Σ-labeled trees that $\mathcal{A}$ accepts.

Example. We define an alternating tree automaton $\mathcal{A}$ that accepts exactly all $\{a,b,c\}$-labeled binary trees in which all paths have a node labeled a and there exists a path with two successive b labels. The automaton is $\mathcal{A} = \langle \{a,b,c\}, \{q_0,q_1,q_2,q_3\}, \delta, q_0, \emptyset \rangle$, where δ is defined in the Table 1.

In the state q_0, the automaton checks both requirements. If a is true, only the second requirement is left to be checked. This is done by sending a copy in state q_1, which searches for two successive b's in some branch, to either the left or the right child. If b is true, $\mathcal{A}$ needs to send more copies. First, it needs to check that all paths in the left and right subtrees have a node labeled a. This is done by sending copies in state q_3 to both the left and the right children. Second, it needs to check that one of these subtrees contains two successive b's. This is done (keeping in mind that the just read b might be the first b in a sequence of two b's) by sending a copy in state q_2 to one of the children. Similarly, if c is true, $\mathcal{A}$ sends copies that check both requirements. As before, a requirement about a is sent universally and

a requirement about the b's is sent existentially.

2.5 An Automata-Theoretic Framework to Branching-Time Model Checking

In Kupferman et al. [2000], we introduced *hesitant alternating automata* (HAA). A HAA over binary trees is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, \alpha \rangle$, where Σ, Q, δ, and Q_0 are as in usual alternating tree automata and $\alpha = \langle G, B \rangle$, with $G \subseteq Q$ and $B \subseteq Q$, is the acceptance condition. As in *weak alternating automata* [Muller et al. 1986], there exists a partition of Q into disjoint sets Q_i, $1 \leq i \leq m$, and a partial order $\leq$ on the collection of the Q_i's such that for every $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, \sigma)$ for some $\sigma \in \Sigma$, we have $Q_j \leq Q_i$. Thus, transitions from a state in Q_i lead to states in either the same Q_i or a lower one. We call m the *depth* of $\mathcal{A}$. In addition, each set Q_i is classified as either *transient*, *existential*, or *universal*, such that for each set Q_i and for all $q \in Q_i$, $\sigma \in \Sigma$, the following holds:

(1) If Q_i is a transient set, then $\delta(q, \sigma)$ contains no elements of Q_i.

(2) If Q_i is an existential set, then $\delta(q, \sigma)$ only contains *disjunctively related* elements of Q_i (i.e., if the transition is rewritten in disjunctive normal form, there is at most one element of Q_i in each disjunct).

(3) If Q_i is a universal set, then $\delta(q, \sigma)$ only contains *conjunctively related* elements of Q_i (i.e., if the transition is rewritten in conjunctive normal form, there is at most one element of Q_i in each conjunct).

It follows that every infinite path of a run ultimately gets trapped within some existential or universal set Q_i. For a path π, we denote by $inf(\pi)$ the set of states that π visits infinitely often. The path then satisfies an acceptance condition $\alpha = \langle G, B \rangle$ if and only if either Q_i is an existential set and $inf(\pi) \cap G \neq \emptyset$, or Q_i is an universal set and $inf(\pi) \cap B = \emptyset$. Note that the acceptance condition of HAA combines the Rabin and the Streett acceptance conditions: existential sets refer to a Rabin condition $\{\langle G, \emptyset \rangle\}$ and universal sets refer to a Streett condition $\{\langle B, \emptyset \rangle\}$. The name *hesitant* comes from the fact that a run of the automaton hesitates between the three types of sets, yet eventually it chooses one set and stays there forever. In Appendix 2, we describe the construction of HAA and give further explanation and intuition to the notion of hesitation. For a state $q \in Q$, we say that q is an *existential state* if and only if $q \in Q_i$ for an existential set Q_i. Universal states are defined analogously. Given a set $\mathcal{D} \subset \mathbb{N}$ of branching degrees, a HAA over $\mathcal{D}$-trees is then $\mathcal{A} = \langle \Sigma, \mathcal{D}, Q, \delta, q_0, \alpha \rangle$, where, as with alternating tree automata, $\delta : Q \times \Sigma \times \mathcal{D} \to \mathcal{B}^+(\mathbb{N} \times Q)$ takes the branching degree of the current node in the input as an additional argument.

THEOREM 2.5. [Kupferman et al. 2000]

(1) Given a CTL formula ψ and $\mathcal{D} \subset \mathbb{N}$, there exists a HAA $\mathcal{A}_{\mathcal{D},\psi} = \langle 2^{AP}, \mathcal{D}, Q, \delta, Q_0, \alpha \rangle$, of size and depth $O(|\psi|)$, such that $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ is exactly the set of $\mathcal{D}$-trees satisfying ψ.

(2) Given a CTL* formula ψ and $\mathcal{D} \subset \mathbb{N}$, there exists a HAA $\mathcal{A}_{\mathcal{D},\psi} = \langle 2^{AP}, \mathcal{D}, Q, \delta, Q_0, \alpha \rangle$, of size $2^{O(|\psi|)}$ and depth $O(|\psi|)$, such that $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ is exactly the set of $\mathcal{D}$-trees satisfying ψ.

In Appendix 2, we describe the construction of HAA for CTL. As discussed in Kupferman et al. [2000], the product $\mathcal{A}_{K,\psi}$ of $\mathcal{A}_{\mathcal{D},\psi}$ with a transition system (module with no fairness condition) K is a 1-letter word HAA. The language of $\mathcal{A}_{K,\psi}$ is not empty if and only if $K \models \psi$. This suggests an automata-based algorithm for branching-time model checking. In the next section we extend this method for fair model checking.

3. AN AUTOMATA-THEORETIC FRAMEWORK TO FAIR MODEL CHECKING

3.1 Libi Alternating Automata

In order to perform fair model checking, we introduce here *Libi alternating automata* (LAA). An LAA is a HAA extended with a Rabin acceptance condition $\beta \subseteq 2^Q \times 2^Q$. We assume that for all pairs $\langle G_i, B_i \rangle \in \beta$, the sets G_i and B_i are disjoint. Consider a LAA $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, \alpha, \beta \rangle$ with $\beta = \{\langle G_1, B_1 \rangle, \ldots, \langle G_k, B_k \rangle\}$. For a run of $\mathcal{A}$, a path π that gets trapped within a set Q_i is accepted by the run if and only if either Q_i is an existential set, in which case π satisfies α and there exists $1 \leq j \leq k$ such that $inf(\pi) \cap G_j \neq \emptyset$ and $inf(\pi) \cap B_j = \emptyset$, or Q_i is a universal set, in which case π satisfies α or π satisfies, for all $1 \leq j \leq k$, either $inf(\pi) \cap G_j = \emptyset$ or $inf(\pi) \cap B_j \neq \emptyset$. We define the *hesitant size* of a LAA $\mathcal{A} = \langle \Sigma, \mathcal{D}, Q, \delta, q_0, \alpha, \beta \rangle$ as $|Q| + |\alpha| + |\delta|$, where $|Q|$ and $|\delta|$ are as in usual alternating automata, and $|\alpha|$ is the sum of the cardinalities of its two sets G and B. Thus, the hesitant size of $\mathcal{A}$ ignores the the size of the Rabin acceptance condition β. We define the *degree* of $\mathcal{A}$ as the number of pairs in β.

It is shown in Kupferman et al. [2000] that the 1-letter nonemptiness problem for HAA of size n and depth m can be solved in space $O(m \log^2 n)$. We now show that the nonemptiness problem for LAA has a similar space complexity, with an additional term for the automaton's Rabin fairness condition.

THEOREM 3.1. *The 1-letter nonemptiness problem for a LAA of hesitant size n, degree k, and depth m, can be solved in space $O(m(log^2 n + \log k))$.*

Alternatively, the 1-letter nonemptiness problem for LAA can be solved in efficient time.

THEOREM 3.2. *The 1-letter nonemptiness problem for a LAA is decidable in time that is linear in both its hesitant size and its degree.*

The proofs of Theorems 3.1 and 3.2 are given in Appendix 3.

3.2 Fair Model Checking

When we perform fair model checking, path quantification ranges only over fair paths. We now show how LAA handle such a quantification. Let $\top$ be an atomic proposition not in AP. We define a function f : CTL* formulas over $AP \rightarrow$ CTL* formulas over $AP \cup \{\top\}$ such that $f(\xi)$ restricts path quantification to paths in which $\top$ holds always. We define f inductively as follows.

—$f(q) = q$.
—$f(\neg\xi) = \neg f(\xi)$.
—$f(\xi_1 \vee \xi_2) = f(\xi_1) \vee f(\xi_2)$.
—$f(E\xi) = E((G\top) \wedge f(\xi))$.

—$f(A\xi) = A((F\neg\top) \vee f(\xi))$.
—$f(X\xi) = Xf(\xi)$.
—$f(\xi_1 U \xi_2) = f(\xi_1) U f(\xi_2)$.

For example, $f(EqU(AFp)) = E((G\top) \wedge (qU(A((F\neg\top) \vee Fq))))$. When ψ is a CTL formula, the formula $f(\psi)$ is not necessarily a CTL formula. Still, it has a restricted syntax: its path formulas have either a single linear-time operator or two linear-time operators connected by a Boolean operator. By [Kupferman and Grumberg 1996], formulas of this syntax have a linear translation to CTL.

Consider a CTL* formula ψ. Let M be a fair module in which $\top$ is valid. Clearly, M satisfies ψ if and only if M satisfies $f(\psi)$. Given some set $\mathcal{D}$ of branching degrees, let $\mathcal{A}_{\mathcal{D},f(\psi)}$ be the HAA corresponding to $\mathcal{D}$ and $f(\psi)$ as described in Theorem 2.5. Let $\langle T_M, V_M^\top \rangle$ be the computation tree obtained by unwinding M into an infinite tree (when M has several initial states, unwinding results in a forest of computations trees, and we need to check them all). By Theorem 2.5, $\mathcal{A}_{\mathcal{D},f(\psi)}$ accepts $\langle T_M, V_M^\top \rangle$ if and only if M satisfies ψ, with path quantification ranging over all paths. Nevertheless, unlike in the HAA $\mathcal{A}_{\mathcal{D},\psi}$, the HAA $\mathcal{A}_{\mathcal{D},f(\psi)}$ enjoys some properties that enable us to define its product with M as a 1-letter LAA that is nonempty if and only if M satisfies ψ, with path quantification ranging over fair paths only.

Let $\mathcal{A}_{\mathcal{D},f(\psi)} = \langle 2^{AP \cup \{\top\}}, \mathcal{D}, Q_\psi, \delta_\psi, Q_\psi^0, \alpha_\psi \rangle$, and let $M = \langle \Sigma, W, R, W_0, L, \beta_M \rangle$ be such that all the branching degrees of M are contained in $\mathcal{D}$. For $w \in W$, let $succ_R(w) = \langle w_0, \ldots, w_{bd(w)-1} \rangle$ be an ordered list of w's R-successors (it is technically convenient to assume that the states of W are ordered). The product automaton of $\mathcal{A}_{\mathcal{D},\psi}$ and M is the 1-letter word LAA $\mathcal{A}_{M,\psi} = \langle \{a\}, W \times Q_\psi, \delta, W_0 \times Q_\psi^0, \alpha, \beta \rangle$, where δ, α, and β are defined as follows:

—Let $q \in Q_\psi$, $w \in W$, $succ_R(w) = \langle w_0, \ldots, w_{k-1} \rangle$, and $\delta_\psi(q, L(w) \cup \{\top\}, k) = \theta$. Then, $\delta(\langle w, q \rangle, a) = \theta'$, where θ' is obtained from θ by replacing each atom (c, η) in θ by the atom $\langle w_c, \eta \rangle$.
—For $\alpha_\psi = \langle G, B \rangle$, we have that $\alpha = \langle W \times G, W \times B \rangle$.
—For $\beta_M = \{\langle G_1, B_1 \rangle, \ldots, \langle G_k, B_k \rangle\}$, $\beta = \{\langle G_1 \times Q_\psi, B_1 \times Q_\psi \rangle, \ldots, \langle G_k \times Q_\psi, B_k \times Q_\psi \rangle\}$.

Proposition 3.3.

(1) $|\mathcal{A}_{M,\psi}| = O(|M| * |\mathcal{A}_{\mathcal{D},f(\psi)}|)$.
(2) $\mathcal{L}(\mathcal{A}_{M,\psi})$ *is nonempty if and only if* $M \models \psi$.

Proof. Part (1) follows easily from the definition of $\mathcal{A}_{M,\psi}$. Indeed, $|W \times Q_\psi| = |W| * |Q_\psi|$, $|\delta| = |W| * |\delta_\psi|$, $|\alpha| = |W| * |\alpha_\psi|$, and $|\beta| = |\beta_M| * |Q_\psi|$.

We now prove part (2). Let us first mention some of the properties of $\mathcal{A}_{\mathcal{D},\psi}$ (for full details see Kupferman et al. [2000]). Since $\mathcal{A}_{\mathcal{D},\psi}$ is a HAA, each of its states may be of one of the following three types:

(1) *A transient state*: These states belong to the transient sets of $\mathcal{A}_{\mathcal{D},\psi}$, and they are associated with subformulas of ψ that are atomic propositions or Boolean combination of inner subformulas.
(2) *An existential state*: These states belong to the existential sets of $\mathcal{A}_{\mathcal{D},\psi}$, and they are associated with subformulas of ψ that are of the form $E\xi$, for a path

formula ξ. In more detail, each subformula $E\xi$ of ψ induces an existential set. The states in this set are the states of a Büchi word automaton $\mathcal{A}_\xi$ that recognizes the linear formula ξ. In each transition from such a state, $\mathcal{A}_{\mathcal{D},\psi}$ sends one copy that follows the automaton $\mathcal{A}_\xi$ along some guessed direction. In addition, $\mathcal{A}_{\mathcal{D},\psi}$ may send copies to sets that are associated with state-subformulas of ξ.

(3) *A universal state*: These states belong to the universal sets of $\mathcal{A}_{\mathcal{D},\psi}$, and they are associated with subformulas of ψ that are of the form $A\xi$. These states are dual to the existential states.

We say that a run of $\mathcal{A}_{\mathcal{D},f(\psi)}$ on $\langle T_M, V_M^\top\rangle$ is β_M-*fair* if and only if all the paths of the run that get trapped in existential sets and satisfy α_ψ correspond to computations in M that satisfy β_M, and all the paths of the run that get trapped in universal sets and do not satisfy α_ψ, correspond to computations in M that do not satisfy β_M. In order to prove (2), we show that $\mathcal{L}(\mathcal{A}_{M,\psi})$ is nonempty if and only if there exists a β_M-*fair* accepting run of $\mathcal{A}_{\mathcal{D},f(\psi)}$ on $\langle T_M, V_M^\top\rangle$. We show that this is sufficient. Consider an existential formula $E\xi$. By the definition, $f(E\xi) = E(G\top \wedge \xi)$. Thus, in the automaton $\mathcal{A}_{\mathcal{D},f(\psi)}$, there is an existential set associated with the formula $E(G\top \wedge \xi)$. When $\mathcal{A}_{\mathcal{D},f(\psi)}$ reads an input tree and visits this set, two things may happen. It may be that $E\xi$ does not hold in the input tree. Then, clearly, $E(G\top \wedge \xi)$ does not hold either, and no run of $\mathcal{A}_{\mathcal{D},f(\psi)}$ over the input tree is accepting. It may also be that $E\xi$ holds in the input tree. Then, as the proposition $\top$ is valid in M, the formula $E(G\top \wedge \xi)$ holds as well. Therefore, $\mathcal{A}_{\mathcal{D},f(\psi)}$ may accept the input tree. For that, one copy c of $\mathcal{A}_{\mathcal{D},f(\psi)}$ should follow a path ρ in the input tree (a path along which $G\top \wedge \xi$ holds) and proceeds over this path in the same way that the word automaton for the formula $G\top \wedge \xi$ does. Since the formula $G\top \wedge \xi$ involves the safety property $G\top$, the copy c continues to follow the path ρ even after satisfaction of ξ has been guaranteed. In other words, the copy c gets trapped in the existential set associated with $E(G\top \wedge \xi)$. It therefore follows that when the run of $\mathcal{A}_{\mathcal{D},f(\psi)}$ is β_M-fair, the path ρ corresponds to a fair computation in M; thus ξ is satisfied along a fair computation. Dually, all the universal formulas in $f(\psi)$ are of the form $A(F\neg\top \vee \xi)$. Therefore, a copy c of $\mathcal{A}_{\mathcal{D},f(\psi)}$ that follows a path ρ in the input tree that does not satisfy ξ is guaranteed to continue to follow the path ρ even after satisfaction of ξ has been refuted. Thus, this copy cannot be run over a fair computation.

Given a β_M-fair accepting run of $\mathcal{A}_{\mathcal{D},f(\psi)}$ over $\langle T_M, V_M^\top\rangle$, we construct an accepting run of $\mathcal{A}_{M,\psi}$. Also, given an accepting run of $\mathcal{A}_{M,\psi}$, we construct a β_M-fair accepting run of $\mathcal{A}_{\mathcal{D},f(\psi)}$ over $\langle T_M, V_M\rangle$. Assume first there exists a β_M-fair accepting run $\langle T_r, r\rangle$ of $\mathcal{A}_{\mathcal{D},\psi}$ over $\langle T_M, V_M^\top\rangle$. Recall that T_r is labeled with $\mathbb{N}^* \times Q_\psi$. A node $y \in T_r$ with $r(y) = (x,q)$ corresponds to a copy of $\mathcal{A}_{\mathcal{D},\psi}$ that is in the state q and reads the tree obtained from unwinding M from $V_M^\top(x)$. Consider the tree $\langle T_r, r'\rangle$ where T_r is labeled with $0^* \times W \times Q_\psi$ and for every $y \in T_r$ with $r(y) = (x,q)$, we have $r'(y) = (0^{|x|}, V_K(x), q)$. We show that $\langle T_r, r'\rangle$ is an accepting run of $\mathcal{A}_{M,\psi}$. First, $\langle T_r, r'\rangle$ is a "legal" run, since the W-component in r' always agrees with V_K. This agreement is the only additional requirement of δ with respect to δ_ψ. Consider a node $y \in T_r$ with $r(y) = (x,q)$, $V_M^\top(x) = w$, and $succ_R(w) = \langle w_0, \ldots, w_{k-1}\rangle$. Let $\delta_\psi(q,w,k) = \theta$. Since $\langle T_r, r\rangle$ is a run of $\mathcal{A}_{\mathcal{D},f(\psi)}$, there exists a set $\{(c_0,q_0),(c_1,q_1),\ldots,(c_n,q_n)\}$ satisfying θ, such that

the successors of y in T_r are $y \cdot i$, for $1 \leq i \leq n$, each labeled with $(x \cdot c_i, q_i)$. In $\langle T_r, r' \rangle$, by its definition, $r'(y) = (0^{|x|}, w, q)$ and the successors of y are $y \cdot i$, each labeled with $(0^{|x+1|}, w_{c_i}, q_i)$. Let $\delta(q, a) = \theta'$. By the definition of δ, the set $\{(w_{c_0}, q_0), (w_{c_1}, q_1), \ldots, (w_{c_n}, q_n)\}$ satisfies θ'. Thus, $\langle T_r, r' \rangle$ is a run of $\mathcal{A}_{D,\psi}$. It is left to see that $\langle T_r, r' \rangle$ is an accepting run. Consider an infinite path ρ of $\langle T_M, V_M^\top \rangle$. Since $\langle T_r, r \rangle$ is β_M-fair accepting, then either ρ gets trapped in an existential set, in which case it satisfies α_ψ and β_M, or ρ gets trapped in an universal set, in which case is satisfies α_ψ or does not satisfy β_M. As α is defined according to α_ψ by crossing it with W, as β is defined according to β_M by crossing it with Q_ψ, it is easy to see that ρ is accepted also in $\mathcal{A}_{M,\psi}$.

Assume now that $\mathcal{A}_{M,\psi}$ accepts a^ω. Thus, there exists an accepting run $\langle T_r, r \rangle$ of $\mathcal{A}_{M,\psi}$. Recall that T_r is labeled with $0^* \times W \times Q_\psi$. Consider the tree $\langle T_r, r' \rangle$, labeled with $\mathbb{N}^* \times Q_\psi$, where $r'(\varepsilon) = (\varepsilon, q_0)$ and for every $y \cdot c \in T_r$ with $r'(y) \in \{x\} \times Q_\psi$ and $r(y \cdot c) = (0^{|x+1|}, w, q)$, we have $r'(y \cdot c) = (x \cdot i, q)$, where i is such that $V_K(x \cdot i) = w$. As in the previous direction, it is easy to see that $\langle T_r, r' \rangle$ is a β_M-fair accepting run of $\mathcal{A}_{\mathcal{D},f(\psi)}$ over $\langle T_M, V_M^\top \rangle$. □

Let $Q_1 \leq \ldots \leq Q_m$ be the partition of the states of $\mathcal{A}_{\mathcal{D},f(\psi)}$ into sets. Then, $W \times Q_1 \ldots W \times Q_m$ is a partition of the states of $\mathcal{A}_{M,\psi}$ into sets. Thus, $\mathcal{A}_{M,\psi}$ has the same depth as $\mathcal{A}_{\mathcal{D},f(\psi)}$.

In conclusion, fair model checking of a fair module M with branching degrees in $\mathcal{D}$, with respect to a formula ψ, is reducible to checking the nonemptiness of a 1-letter word LAA of size $O(|M| * |\mathcal{A}_{\mathcal{D},f(\psi)}|)$ and of the same depth as $\mathcal{A}_{\mathcal{D},f(\psi)}$. Thus, Theorems 2.5 and 3.1, together with Proposition 3.3, imply the theorem below.

THEOREM 3.4.

(1) *The fair model-checking problem for CTL can be solved in space* $O(m \log^2(mn))$, *where* m *is the length of the formula and* n *is the size of the fair module.*

(2) *The fair model-checking problem for CTL* can be solved in time* $O(m(m + \log n)^2)$, *where* m *is the length of the formula and* n *is the size of the fair module.*

In the following sections, we show how this result can be used to derive tight space complexity bounds also for the satisfiability, validity, and implication problems of universal branching temporal logics, as well as for the branching modular model-checking problem.

Since, by Theorem 3.2, the nonemptiness problem for LAA can be checked in time that is linear in the hesitant size of the automaton and in its degree, LAA also provide a framework for time-efficient CTL and CTL* model checking. Formally, we have the following.

THEOREM 3.5.

(1) *The fair model-checking problem for CTL can be solved in time* $O(mnk)$, *where* m *is the length of the formula, and* n *and* k *are the size and the degree of the fair module.*

(2) *The fair model-checking problem for CTL* can be solved in space* $O(2^m nk)$, *where* m *is the length of the formula, and* n *and* k *are the size and the degree of the fair module.*

REMARK 3.6. One can also define LAA as HAA extended with a generalized Büchi acceptance condition. Consider a LAA $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, \alpha, \beta \rangle$ with a generalized Büchi acceptance condition $\beta = \{G_1, G_2, \ldots, G_k\}$. For a run of $\mathcal{A}$, a path π that gets trapped within a set Q_i is accepted by the run if and only if either Q_i is an existential set, in which case π satisfies α and for all $1 \leq j \leq k$, we have that $inf(\pi) \cap G_j \neq \emptyset$, or Q_i is a universal set, in which case π satisfies α or there exists $1 \leq j \leq k$ for which $inf(\pi) \cap G_j = \emptyset$. The nonemptiness procedure described in Theorem 3.1 can be easily revised to handle LAA with generalized Büchi acceptance conditions. Hence, such LAA can be used for space-efficient fair model checking of fair modules with a generalized Büchi fairness condition. Note that, unlike Rabin fair modules, it is easy to define the composition of two generalized-Büchi fair modules. While it is also easy to define the composition of two Streett fair modules, the 1-letter nonemptiness problem for LAA with Streett fairness condition probably cannot be solved in efficient space, since the emptiness problem for nondeterministic Streett automata is PTIME-complete [Kupferman and Vardi 1998]. Moreover, for LAA with a generalized Büchi acceptance condition, one can improve the algorithm described in the proof of Theorem 3.2 to run in time that is linear in $h + |\beta|$, where h is the hesitant size of the LAA and $|\beta|$ is the sum of the cardinalities of the sets appearing in the pairs in α. The complexity of the corresponding CTL model-checking problem, of modules with generalized Büchi fairness conditions, can then be solved in linear time, matching the bound in Clarke et al. [1986].

4. SATISFIABILITY, VALIDITY, AND IMPLICATION

The basic decision problems for a specification language are *satisfiability*, i.e., is the specification trivially false, *validity*, i.e., is the specification trivially true, and *implication*, i.e., does one specification logically imply another specification. While the first two enable us to make sure that our specifications are interesting, implication generalizes both and, as we show here, is also strongly related to modular model checking. For a specification language that is closed under negation, these problems are intereducible. This is not the case, however, for universal branching temporal logics.

The satisfiability problem for a universal branching temporal logic is defined as follows: given a formula φ, is there a *nonempty* fair module M such that φ holds in M? As a fair module with no fair path trivially satisfies any formula of the form $A\xi$, the nonemptiness requirement is essential. The validity problem is defined as follows: given a formula φ, does φ hold in all fair modules? (It is easy to see that φ holds in all fair modules if and only if φ holds in all nonempty fair modules.) The implication problem is defined as follows: given two formulas φ and ψ, does φ imply ψ (denoted $\varphi \rightarrow \psi$)? Namely, does ψ hold in every fair module in which φ holds?[1] In the lemma below, we assert that the satisfiability and the validity problems are special cases of the implication problem.

[1] Recall that validity of the formula $\varphi \rightarrow \psi$ means that the formula $\varphi \rightarrow \psi$ holds in all the initial states of all fair modules. Thus, if an initial state satisfies φ, it should also satisfy ψ. On the other hand, the implication $\varphi \rightarrow \psi$ means that in all fair modules, if φ is satisfied in all the initial states, so does ψ. A particular module M may not satisfy the formula $\varphi \rightarrow \psi$, and still satisfies the requirement that if φ holds in all M's initial states, so does ψ (such a module M has an initial

LEMMA 4.1. *Let φ be a formula in $\forall CTL^\star$. Then,*

*(1) The formula φ is satisfiable if and only if φ does not imply A***false***.*

(2) The formula φ is valid if and only if **true** *implies φ.*

Since the implication $\varphi \rightarrow \psi$ is valid if and only if $\varphi \wedge \neg\psi$ is not satisfiable, the implication problem combines both universal and existential branching temporal logics and is more general than each of these problems. In this section we present a method for solving the implication problem.

4.1 Maximal Models and Implication

Let φ and ψ be two $\forall$CTL* formulas. By definition, the implication $\varphi \rightarrow \psi$ is valid if and only if ψ holds in all fair modules M in which φ holds. Since the more behaviors M has the less likely it is to satisfy ψ, it makes sense to examine the implication by checking ψ in a maximal model of φ (we assume, without loss of generality, that φ and ψ are defined over the same set AP of atomic propositions). Formally, we have the following theorem. The theorem is proved in Grumberg and Long [1994] for $\forall$CTL, but the proof applies also to $\forall$CTL*.

THEOREM 4.2. *Let φ and ψ be $\forall CTL^\star$ formulas, and let M_φ be a maximal model of φ. Then φ implies ψ if and only if $M_\varphi \models \psi$.*

To complete the reduction of implication to fair model checking, we have to describe the construction of maximal models. A construction of maximal models for $\forall$CTL formulas is described in Grumberg and Long [1994] (see Theorem 2.3). We now consider formulas in $\forall$CTL*, and we wish to construct maximal models for such formulas. Unfortunately, the tableau-based technique that was used in the proof of Theorem 2.3 does not seem to extend to $\forall$CTL*. Indeed, while the satisfiability problem for CTL can be solved using the tableau-based technique [Emerson and Halpern 1985], the satisfiability problem for CTL* requires sophisticated automata-theoretic techniques [Emerson and Jutla 1988]. We now show that these automata-theoretic techniques can be used to construct maximal models for $\forall$CTL* formulas.

THEOREM 4.3. *For every $\forall CTL^\star$ formula φ, there exists a maximal model M_φ of size $2^{2^{O(|\varphi|)}}$.*

The proof of the Theorem is given in Appendix 4. We can now obtain an alternative proof of Theorem 2.3.

THEOREM 4.4. *For every $\forall CTL$ formula φ, there exists M_φ of size $2^{O(|\varphi|)}$.*

PROOF. Exactly as for $\forall$CTL*. Here, however, $\mathcal{A}_{\forall(\varphi)}$ is of size $O(|\varphi|)$, and hence $\overline{\mathcal{A}_{\forall(\varphi)}}$ is of size $2^{O(|\varphi|)}$. □

4.2 Complexity

We are now ready to combine the maximal-model technique with the space-efficient fair-model-checking algorithm to solve the implication problem. We also show that the upper bounds that follow are tight.

---

state in which φ does not hold). Nevertheless, the definition of validity and implication refers to all modules. Therefore, $\varphi \rightarrow \psi$ is valid if and only if the implication $\varphi \rightarrow \psi$ holds.

THEOREM 4.5.

(1) *The implication problem for $\forall$CTL is PSPACE-complete.*

(2) *The implication problem for $\forall$CTL* is EXPSPACE-complete.*

PROOF. By Theorem 4.2, the implication problem $\varphi \rightarrow \psi$ is reducible to model checking of ψ in M_φ. Theorems 4.3 and 4.4 provide us with a doubly exponential bound for maximal models of CTL* and an exponential bound for maximal models of CTL. Together with Theorem 3.4, this implies the upper bounds. For the precise bounds, let n and m denote the lengths of φ and ψ, respectively. Consider first the case that φ and ψ are $\forall$CTL formulas. Then, the size of M_φ is $2^{O(n)}$, and the space complexity that follows is $O(m \log^2(m2^{O(n)})) = O(m(n + \log m)^2)$. When φ and ψ are $\forall$CTL* formulas, the size of M_φ is $2^{2^{O(n)}}$, and the space complexity that follows is $O(m(m + 2^{O(n)})^2)$. We note that we can construct M_φ on-the-fly; thus the nonemptiness check proceeds without constructing M_φ in advance. This leads to an exponential saving in the space complexity.

To prove the PSPACE lower bound for the implication problem for $\forall$CTL, we prove that the satisfiability problem for $\forall$CTL is PSPACE-hard. The result then follows from Lemma 4.1. We prove hardness in PSPACE by a reduction from LTL satisfiability, proved to be PSPACE-hard in Sistla and Clarke [1985]. Given an LTL formula ξ, let ξ_A be the $\forall$CTL formula obtained from ξ by preceding each temporal operator with the path quantifier A. For example, if $\xi = FXp$ then $\xi_A = AFAXp$. It is easy to see that ξ is satisfiable if and only if ξ_A is satisfiable. Indeed, a computation that satisfies ξ can be viewed as a fair module satisfying ξ_A. For the second direction, assume that ξ_A is satisfiable in some fair module M. Consider a fair computation π of M. We can view π as a fair module of branching degree 1. Clearly, π simulates M, and thus, it satisfies ξ_A as well. Also, since its branching degree is 1, the computation π also satisfies ξ. Thus, ξ is satisfiable.

To prove the EXPSPACE lower bound for the implication problem for $\forall$CTL*, we do a reduction from the problem whether an exponential-space deterministic Turing machine T accepts an input word x. That is, given T and x, we construct two $\forall$CTL* formulas φ and ψ such that T accepts x if and only if φ does not imply ψ. The details of the construction are given in Appendix 5. □

Note that the implication problem $\varphi \rightarrow \psi$ for φ in $\forall$CTL and ψ in $\forall$CTL* is still in PSPACE. Indeed, then, the size of M_φ is exponential in the length of φ, and the space complexity that follows from Theorem 3.4 (with n and m denoting the lengths of φ and ψ, respectively) is $O(m(m + \log 2^{O(n)})^2)) = O(m(m + n)^2)$. On the other hand, the problem is already EXPSPACE-hard for φ of the form $A\xi$ for an LTL formula ξ and ψ in $\forall$CTL,

Recall that satisfiability and validity are special cases of the implication problem. As Theorem 4.6 below shows, these problems are easy special cases.

THEOREM 4.6.

(1) *The satisfiability problems for $\forall$CTL and $\forall$CTL* are PSPACE-complete.*

(2) *The validity problem for $\forall$CTL is co-NP-complete.*

(3) *The validity problem for $\forall$CTL* is PSPACE-complete.*

PROOF. The results for the satisfiability follow from the PSPACE-completeness of the satisfiability problem for LTL. We have already seen the lower bound in the proof of Theorem 4.5 For the upper bound, given a $\forall$CTL* formula φ, it is easy to see (as detailed in the proof of Theorem 4.5) that φ is satisfiable if and only if the LTL formula obtained from φ by eliminating its path quantifiers is satisfiable.

Since $\forall$CTL subsumes propositional logic, hardness in co-NP for the validity problem of $\forall$CTL is immediate. To prove membership in co-NP, we prove that the satisfiability problem for $\exists$CTL is in NP. To do this, we present a linear-size model property for $\exists$CTL; that is, we prove that a satisfiable $\exists$CTL formula φ is satisfiable also in a fair module with at most $|\varphi|$ states and $|\varphi|$ transitions. The proof of the linear-size model property for $\exists$CTL appears in Appendix 6.

It is left to prove PSPACE completeness for the validity problem of $\forall$CTL*. Since a $\forall$CTL* formula ψ is valid if and only if the implication $\mathbf{true} \rightarrow \psi$ is valid, and since the implication problem $\varphi \rightarrow \psi$ for φ in $\forall$CTL and ψ in $\forall$CTL* is still in PSPACE (as explained in the proof of Theorem 4.5), membership in PSPACE is easy. To prove hardness, we use the PSPACE-hardness of the validity problem for LTL. Clearly, an LTL formula ξ is valid if and only if the $\forall$CTL* formula $A\xi$ is valid. □

In Theorem 5.1, we show that the implication problem is strongly related to modular model checking.

5. MODULAR MODEL CHECKING

5.1 The Branching Modular Model-Checking Problem

In modular verification, one uses assertions of the form $\langle\varphi\rangle M \langle\psi\rangle$ to specify that whenever M is part of a system satisfying the universal branching temporal logic formula φ, the system satisfies the universal branching temporal logic formula ψ too. Formally, $\langle\varphi\rangle M \langle\psi\rangle$ holds if $M \| M' \models \psi$ for all M' such that $M \| M' \models \varphi$. Here φ is an assumption on the behavior of the system and ψ is the guarantee on the behavior of the fair module. Assume-guarantee assertions are used in modular proof rules of the following form:

$$\left.\begin{array}{l} \langle\varphi_1\rangle M_1 \langle\psi_1\rangle \\ \langle\mathbf{true}\rangle M_1 \langle\varphi_1\rangle \\ \langle\varphi_2\rangle M_2 \langle\psi_2\rangle \\ \langle\mathbf{true}\rangle M_2 \langle\varphi_2\rangle \end{array}\right\} \langle\mathbf{true}\rangle M_1 \| M_2 \langle\psi_1 \wedge \psi_2\rangle$$

According to the above rule, we can infer that $\psi_1 \wedge \psi_2$ holds in the composition $M_1 \| M_2$ by proving assume-guarantee assertions on M_1 and M_2 in isolation. Thus, a key step in modular verification is checking that assume-guarantee assertions hold, which we called the *branching modular model-checking problem.*

We now relate the modular model-checking problem and the implication problem. For a set AP of atomic propositions, let M_{AP} be the *full model over* AP. That is,

$$M_{AP} = \langle AP, 2^{AP}, 2^{AP} \times 2^{AP}, 2^{AP}, L, \{\langle 2^{AP}, \emptyset\rangle\}\rangle,$$

where for all $w \in 2^{AP}$ we have that $L(w) = w$.

THEOREM 5.1. *Let φ and ψ be $\forall$CTL* formulas over a set AP of atomic propositions. The implication $\varphi \rightarrow \psi$ is valid if and only if $\langle\varphi\rangle M_{AP} \langle\psi\rangle$.*

PROOF. For every fair module M, the fair modules M and $M\|M_{AP}$ simulate each other. Hence, for every $\forall$CTL* formula φ over AP we have that $M\|M_{AP} \models \varphi$ if and only if $M \models \varphi$. Thus, the implication $\varphi \rightarrow \psi$ is valid if and only if $\langle\varphi\rangle M_{AP}\langle\psi\rangle$. □

Theorem 5.1 enables us to use the techniques developed in the previous sections, maximal models and space-efficient fair model checking, in order to yield a solution to the branching modular model-checking problem.

THEOREM 5.2. *For all $\forall CTL^\star$ formulas φ and ψ, and for every module M, we have that $\langle\varphi\rangle M\langle\psi\rangle$ if and only if $M\|M_\varphi \models \psi$.*

PROOF. Assume first that $\langle\varphi\rangle M\langle\psi\rangle$. Thus, whenever M is part of a system satisfying φ, the system satisfies ψ too. By Theorem 2.2 (3), we have $M\|M_\varphi \leq M_\varphi$. Since $M_\varphi \models \varphi$, it follows, by Theorem 2.1 (2), that $M\|M_\varphi$ satisfies φ. Consequently, $M\|M_\varphi$ satisfies ψ.

Assume now that $M\|M_\varphi \models \psi$, and let $M\|M'$ be such that $M\|M' \models \varphi$. Then, $M\|M' \leq M_\varphi$, which implies, by Theorem 2.2 (1), that $M\|M\|M' \leq M\|M_\varphi$. Thus, by Theorem 2.2 (2), $M\|M' \leq M\|M_\varphi$, and therefore, by Theorem 2.1 (2), $M\|M' \models \psi$. Hence, $\langle\varphi\rangle M\langle\psi\rangle$. □

It follows that branching modular model checking can be reduced to fair model checking. In Theorem 5.3 below we apply the reduction to the logics $\forall$CTL and $\forall$CTL*. We also show that the upper bounds that follow are tight.

THEOREM 5.3.

(1) *The branching modular model-checking problem for $\forall CTL$ is PSPACE-complete, more precisely, determining whether $\langle\varphi\rangle M\langle\psi\rangle$, for φ and ψ in $\forall CTL$, can be done in space $O(m(l+\log mk)^2)$, where l is the length of φ, k is the size of M, and m is the length of ψ.*

(2) *The branching modular model-checking problem for $\forall CTL^\star$ is EXPSPACE-complete, more precisely, determining whether $\langle\varphi\rangle M\langle\psi\rangle$, for φ and ψ in $\forall CTL^\star$, can be done in space $O(m(m+\log k+2^{O(l)})^2)$, where l is the length of φ, k is the size of M, and m is the length of ψ.*

PROOF. We start with the upper bounds. By Theorem 5.2, the problem of determining whether $\langle\varphi\rangle M\langle\psi\rangle$ is reducible to model checking of ψ in $M\|M_\varphi$. By Theorem 3.4, the fair model-checking problem for CTL can be solved in space $O(m\log^2(mn))$, where m is the length of the formula and n is the size of the fair module. By Theorem 4.4, the size of M_φ, for φ in $\forall$CTL, is $2^{O(l)}$. Hence, the size of the fair module $M\|M_\varphi$ is $k2^{O(l)}$. It follows that the problem of determining whether $\langle\varphi\rangle M\langle\psi\rangle$ can be solved in space $O(m\log^2 mk2^{O(l)}) = O(m(l+\log mk)^2)$. Theorem 3.4 also says that the fair model-checking problem for CTL* can be solved in space $O(m(m+\log n)^2)$. By Theorem 4.3, the size of M_φ, for φ in $\forall$CTL*, is $2^{2^{O(l)}}$. Hence, the size of the fair module $M\|M_\varphi$ is $k2^{2^{O(l)}}$. It follows that the the problem of determining whether $\langle\varphi\rangle M\langle\psi\rangle$ can be solved in space $O(m(m+\log k2^{2^{O(l)}})^2) = O(m(m+\log k+2^{O(l)})^2)$.

We now turn to the lower bounds. For both bounds, we do a reduction from the implication problem. By Theorem 5.1, for every two $\forall$CTL* formulas φ and ψ over a set AP of atomic propositions, the implication $\varphi \rightarrow \psi$ is valid if and only

if $\langle\varphi\rangle M_{AP}\langle\psi\rangle$. The complexity of the reduction depends on the size of M_{AP}. We show that for both $\forall$CTL and $\forall$CTL*, the size of M_{AP} is fixed. The satisfiability problem for LTL is PSPACE-hard already for formulas with a fixed number of atomic propositions. The PSPACE-hardness proof in Sistla and Clarke [1985] uses temporal formulas with an unbounded number of atomic propositions. Nevertheless, By using a Turing machine M that accepts a PSPACE-complete language, it is possible to bound the number of atomic propositions used to the size of the working alphabet of M. Since it is possible to encode the truth values of m atomic propositions in one state by the truth values of a single atomic proposition along $\log m$ states, it follows that satisfiability of temporal formulas with a single atomic proposition is also PSPACE-hard. It follows that the implication problem for $\forall$CTL is PSPACE-hard already for formulas with a fixed number of atomic propositions. Thus, the size of M_{AP} is fixed. Also, since we took, in the reduction in the proof of Theorem 4.5, a fixed Turing machine, the set of atomic propositions required for defining ξ there is fixed. Hence, so is the size of M_{AP}. □

Note that the crucial factor in the complexity of the branching modular model checking problem is the assumption part of the specification. In particular, the branching modular model-checking problem for assumptions in $\forall$CTL and guarantees in $\forall$CTL* is still in PSPACE-complete. Moreover, the complexities given in Theorem 5.3 remain true even is we fix the guarantee part of the specification. This suggests that modular model checking in the branching temporal framework can be practical only for very small assumptions. Indeed, in many examples the assumptions do tend to be of a very small size [Josko 1987b; 1989; Grumberg and Long 1994]; see also Abadi and Lamport [1993].

Since modular model checking with assumption φ and guarantee ψ in LTL reduces to model checking the formula $\varphi \rightarrow \psi$ [Pnueli 1985a; Jonsson and Tsay 1995], it follows that determining whether M guarantees ψ under the assumption φ can be done in time $k2^{O(l+m)}$ and space $O((\log k + l + m)^2)$, where k is the size of M, l is the length of φ, and m is the length of ψ. Thus, while the complexity of branching modular model checking with assumptions in $\forall$CTL is higher than the complexity of CTL model checking, it is comparable to the complexity of LTL modular model checking. How do LTL and $\forall$CTL compare from the expressiveness point of view? While $\forall$CTL and LTL have incomparable expressive power, in practice one often finds LTL to be more expressive, as the specifications that can be expressed in $\forall$CTL but not in LTL rarely arise in practical settings. Since the complexity of CTL model checking is lower than that of LTL model checking, we are often willing to settle for the lower expressiveness of CTL; that is, we are willing to verify the design with respect to weaker specifications, with the hope that design errors will be discovered in the process. For example, a significant portion of verified properties are *safety* properties that can be expressed as $AG\varphi$, where φ is a propositional formula.

While we might be willing to settle for a weak guarantee, we cannot, however, settle for weak assumptions. In many cases one needs to adopt rather strong assumptions in order to verify even a very weak guarantee. Very often such assumptions are simply not expressible in $\forall$CTL. For example, the assumption $AFG\varphi$, where φ is a propositional formula, cannot be expressed in $\forall$CTL [Emerson and

Halpern 1986]. Thus, ∀CTL is simply not expressive enough as a specification language for modular model checking. In the next section we will consider using LTL as a specification language for assumptions in modular model checking.

5.2 The Linear-Branching Modular Model-Checking Problem

The modular proof rule in the preceding section uses branching assumptions and guarantees. As mentioned in the introduction, there is another approach in which the assumption is a linear temporal formula, while the guarantee is a branching temporal formula. In this approach, the assumption in the assume-guarantee pair concerns the interaction of the module with its environment along each computation, and is therefore more naturally expressed in a linear temporal logic. We denote this kind of assertion by $[\varphi]M\langle\psi\rangle$. The meaning of such an assertion is that the branching temporal formula ψ holds in the computation tree that consists of all computations of the program that satisfy the linear temporal formula φ. Verifying such assertions is called the *linear-branching modular model-checking problem.*

In order to define the linear-branching model checking problem formally, we define *extended modules.* An extended module $\langle M, P\rangle$ is a module $M = \langle AP, W, R, W_0, L\rangle$ extended by a language $P \subseteq (2^{AP})^\omega$. We regard P as a fairness condition: a computation π of M is fair if and only if $L(\pi) \in P$. Unlike the Rabin fairness condition, fairness of a computation π with respect to P cannot be determined by the fairness of any of π's suffixes. Therefore, in order to define the semantics of CTL* formulas with respect to extended modules, we first associate with each module M a *tree module* M^t. Intuitively, M^t is obtained by unwinding M to an infinite tree. Let $M = \langle AP, W, R, W_0, L\rangle$. A *partial path* ζ in M is a finite prefix, $w_0, w_1, \ldots, w_k$, of a computation in M, where $w_0 \in W_0$. We denote the set of partial paths of M by $ppath(M)$. The tree module of M is $M^t = \langle AP, ppath(M), R^t, \{w_0\}, L^t\rangle$, where for every partial path $\zeta = w_0, \ldots, w_k \in ppath(M)$, we have

—$R^t(\zeta, \zeta')$ if and only if there exists $w_{k+1} \in W$ such that $R(w_k, w_{k+1})$ and $\zeta' = w_0, \ldots, w_k, w_{k+1}$. That is, the partial path ζ' extends the partial path ζ by a single transition in M.

—$L^t(\zeta) = L(w_k)$.

Note that M^t is indeed a tree; every state has a unique predecessor. A computation $\zeta_0, \zeta_1, \ldots$ in M^t is an *anchored path* if and only if ζ_0 is in W_0.

The semantics of CTL* with respect to tree modules extended by a fairness condition $P \subseteq (2^{AP})^\omega$ is defined as the usual semantics of CTL*, with path quantification ranging only over anchored paths that are labeled by a word in P. Thus, for example,

—$\zeta \models E\xi$ if there exists an anchored path $\pi = \zeta_0, \ldots, \zeta_i, \zeta_{i+1}, \ldots$ of M^t and $i \geq 0$ such that $L(\pi) \in P$, $\zeta_i = \zeta$, and $\pi^i \models \xi$.

—$\zeta \models A\xi$ if for all anchored paths $\pi = \zeta_0, \ldots, \zeta_i, \zeta_{i+1}, \ldots$ of M^t and $i \geq 0$ such that $L(\pi) \in P$ and $\zeta_i = \zeta$, we have $\pi^i \models \xi$.

Note that by defining the truth of formulas on the nodes of the computation tree M^t, we guarantee that only one path leads to each node. The extended tree module $\langle M^t, P_\varphi\rangle$ satisfies a formula ψ if and only if $\{w_0\} \models \psi$. We say that $\langle M, P\rangle \models \psi$ if

and only if $\langle M^t, P\rangle \models \psi$. Now, $[\varphi]M\langle\psi\rangle$ holds if and only if $\langle M, P_\varphi\rangle \models \psi$, where P_φ is the set of all that computations that satisfy φ.[2].

We first show that when the language P is given by a deterministic Rabin automaton, we can translated the extended modules $\langle M, P\rangle$ to an equivalent fair module.

LEMMA 5.4. *Let $\langle M, P\rangle$ be an extended module, and let $\mathcal{A}_P$ be a deterministic Rabin automaton such that $\mathcal{L}(\mathcal{A}_P) = P$. We can construct a fair module M' such that for every CTL* formula ψ, we have that $\langle M, P\rangle \models \psi$ if and only if $M' \models \psi$. Moreover, $M' \leq M\|M'$.*

PROOF. Let $M = \langle AP, W, R, W_0, L\rangle$ and $\mathcal{A}_P = \langle 2^{AP}, Q, \delta, q_0, F\rangle$.
We define $M' = \langle AP, W \times Q, R', W_0 \times \{q_0\}, L', \alpha\rangle$, where

—$R'(\langle w, q\rangle, \langle w', q'\rangle)$ if and only if $R(w, w')$ and $\delta(q, L(w)) = q'$.
—$L'(\langle w, q\rangle) = L(w)$.
—$\alpha = \{\langle W \times G, W \times B\rangle : \langle G, B\rangle \in F\}$.

We prove that $\langle M^t, P\rangle$ and M' satisfy the same CTL* formulas. For a state $\zeta = w_0, \ldots, w_k \in \mathit{ppath}(M)$, we denote by $\mathit{last}(\zeta)$ the state $w_k \in W$, and denote by $\sigma(\zeta)$ the finite word $L(w_0)\cdots L(w_k) \in (2^{AP})^*$. Also, for a finite word $\sigma \in (2^{AP})^*$, let $\delta(q_0, \sigma)$ be the state that $\mathcal{A}_P$ reaches after reading σ.

The fact that every state in M^t is associated with a single partial path of M enables us to relate the states of M^t with the states of M'. Formally, we claim the following. For every CTL* formula ψ and state ζ in $\langle M^t, P\rangle$, we have that $\zeta \models \psi$ in $\langle M^t, P\rangle$ if and only if $\langle \mathit{last}(\zeta), \delta(q_0, \sigma(\zeta))\rangle \models \psi$ in M'. In particular, $\{w_0\} \models \psi$ in $\langle M^t, P\rangle$ if and only if $\langle w_0, q_0\rangle \models \psi$ in M'. The proof proceeds by induction on the structure of ψ. The interesting case is $\psi = A\xi$ or $\psi = E\xi$, for a path formula ξ. Let $\psi = A\xi$. Assume first that $\zeta \models \psi$ in $\langle M^t, P\rangle$. Then, for every anchored path $\pi = \zeta_0, \ldots, \zeta_i, \zeta_{i+1}, \ldots$ of M^t such that $L(\pi) \in P$ and $\zeta_i = \zeta$, we have that $\pi^i \models \xi$ in $\langle M^t, P\rangle$. Consider a fair computation $\rho = \langle w_0, q_0\rangle, \ldots, \langle w_i, q_i\rangle, \langle w_{i+1}, q_{i+1}\rangle, \ldots$ in M' for which $\langle w_i, q_i\rangle = \langle \mathit{last}(\zeta), \delta(q_0, \sigma(\zeta))\rangle$. Let $\pi = \zeta_0, \ldots, \zeta_i, \zeta_i \cdot w_{i+1}, \zeta_i \cdot w_{i+1} \cdot w_{i+2}, \ldots$ be the anchored path in M^t that corresponds to ρ. Since ρ is fair, $L(\pi) \in P$. Hence, $\zeta_i, \zeta_i \cdot w_{i+1}, \zeta_i \cdot w_{i+1} \cdot w_{i+2}, \ldots$ satisfies ξ. Then, by the induction hypothesis, $\langle w_i, q_i\rangle, \langle w_{i+1}, q_{i+1}\rangle, \ldots$ satisfies ξ as well, and we are done. The proof for $\psi = E\xi$ is similar.

It is left to see that $M' \leq M\|M'$. Recall that the state space of $M\|M'$ is $W \times W \times Q$. Intuitively, since M' is a restriction of M, composing M' with M does not restrict it further. Formally, it is easy to see that the relation

$$H = \{\langle\langle w, q\rangle, \langle w, w, q\rangle\rangle : \langle w, q\rangle \in W \times Q\}$$

is a fair simulation from M' to $M\|M$.

□

Before we turn to solve the the linear-branching model-checking problem, we show that the branching modular framework is more general than the linear-branching

[2]We note that the formal definitions of $[\varphi]M\langle\psi\rangle$ in Josko [1987a; 1987b; 1989] apply only to restricted linear temporal assumptions and involve a complicated syntactic construction.

modular framework. Thus, the algorithms discussed in Section 5.1 are applicable also here.

THEOREM 5.5. *For every LTL formula φ, fair module M, and a $\forall$CTL* formula ψ, we have that $\langle A\varphi\rangle M\langle\psi\rangle$ if and only if $[\varphi]M\langle\psi\rangle$.*

PROOF. Given φ, M, and ψ, assume first that $[\varphi]M\langle\psi\rangle$ holds. Let P_φ be the set of computations satisfying φ. Thus, the extended module $\langle M, P_\varphi\rangle$ satisfies ψ. Consider the composition $M\|M'$ of M with some module M'. Recall that for M and M' with state spaces W and W', respectively, the state space W'' of $M\|M'$ consists of all the pairs $\langle w, w'\rangle$ for which w and w' agree on the labels of the atomic propositions joint to M and M'. Then, the relation

$$H = \{\langle\langle w, w'\rangle, w\rangle : \langle w, w'\rangle \in W''\}$$

is a simulation relation from $M\|M'$ to M. It is easy to see that H is also a simulation relation from $\langle M\|M', P_\varphi\rangle$ to $\langle M, P_\varphi\rangle$. Hence, $\langle M\|M', P_\varphi\rangle$ satisfies ψ as well. Let M' be such that $M\|M' \models A\varphi$. That is, all the computations in $M\|M'$ satisfy φ. Hence, the identity relation is a simulation relation from $M\|M'$ to $\langle M\|M', P_\varphi\rangle$. Therefore, as $\langle M\|M', P_\varphi\rangle$ satisfies ψ, so does $M\|M'$, and we are done.

Assume now that $\langle A\varphi\rangle M\langle\psi\rangle$ holds. Let $M_{A\varphi}$ be the maximal model of $A\varphi$. Since $M_{A\varphi} \models A\varphi$ and $M\|M_{A\varphi} \leq M_{A\varphi}$, we have that $M\|M_{A\varphi} \models A\varphi$, and therefore, by the assumption, $M\|M_{A\varphi} \models \psi$. Let M' be the fair module equivalent to $\langle M, P_\varphi\rangle$, as defined in Lemma 5.4. That is, $\langle M, P_\varphi\rangle$ and M' satisfy the same CTL* formulas. Since $\langle M, P_\varphi\rangle \models A\varphi$, we have that $M' \models A\varphi$. Hence, $M' \leq M_{A\varphi}$, and therefore, by Theorem 2.1 (2), $M\|M' \leq M\|M_{A\varphi}$. Hence, as $M\|M_{A\varphi} \models \psi$, we have that $M\|M' \models \psi$. Since, by Lemma 5.4, $M' \leq M'\|M$, it follows that M' satisfies ψ as well. Hence, $\langle M, P_\varphi\rangle$ also satisfies ψ, and we are done. □

It is known that the $\forall$CTL formula $AFAGp$ is not equivalent to any formula of the form $A\varphi$, where φ is an LTL formula. Thus, the branching modular framework is strictly more expressive than the linear-branching modular framework, with no increase in worst-case computational complexity (we have seen, in the proof of Theorem 5.3, that the EXPSPACE lower bound holds already for assumptions of the form $A\xi$ for an LTL formula ξ). We now describe a second algorithm for solving the linear-branching model-checking problem. This algorithm handles any guarantee in CTL and CTL* (that is, it is not restricted to their universal fragments), and is simpler than the algorithm for branching modular model checking, as it avoids the construction of maximal models in Section 4.1. An different construction, which avoids the use of determinization, is described in Vardi [1995].

THEOREM 5.6. *The linear-branching model-checking problem is EXPSPACE-complete for guarantees in both CTL and CTL*.*

PROOF. Proving the EXPSPACE lower bound in Theorem 5.3, we use an assumption of the form $A\xi$ for an LTL formula ξ and a guarantee in $\forall$CTL. Therefore, the lower bound proved there is valid also here.

For the upper bound, we reduce the linear-branching model-checking problem to the fair model-checking problem. Given M, φ, and ψ, let $\mathcal{A}_\varphi$ be a deterministic

Rabin automaton that accepts exactly all computations that satisfy φ. By Theorem 2.4, we can first construct for φ a nondeterministic Büchi automaton. Then, by Safra [1988], we determinize this automaton and get a deterministic Rabin automaton with $2^{2^{O(|\varphi|)}}$ states. Following Lemma 5.4, we can now construct from M and $\mathcal{A}_\varphi$ a fair module M' such that $[\varphi]M\langle\psi\rangle$ if and only if $M' \models \psi$. If M has n states, and the length of φ is l, then M' has $n \cdot 2^{2^{O(l)}}$ states. The result then follows from the complexity of the fair-model checking problem (Theorem 3.4).

A careful analysis of the complexity shows that the algorithm uses space $O(m(\log mn + 2^{O(l)})^2)$, where m is the size of ψ, for ψ in CTL, and uses space $O(m(m + \log n + 2^{O(l)})^2)$, for ψ in CTL*. □

6. CONCLUDING REMARKS

The results of the paper indicate that modular model checking for general universal or linear assumptions is rather intractable. In view of these discouraging results, is there hope for modular model checking? One should keep in mind that the bounds in the paper are worst-case bounds. In practice, the constructions we used in the proofs (maximal models, translations to automata, and the subset constructions) need not yield an exponential blowup. In addition, heuristics may be employed to avoid unnecessary states in these constructions. If the size of the assumption is not too large and the doubly exponential blowup is avoided, then our algorithms might not be always impractical.

Our result provide an *a posteriori* justification for Josko's restriction on the linear temporal assumption [Josko 1987a; 1987b; 1989]. Essentially, because of the restriction imposed on the linear temporal assumption, one can get more economical automata-theoretic construction (exponential rather than doubly exponential) of the maximal model associated with the CTL* formula $A\varphi$ as well as with M_φ. It will be interesting to find other fragments of LTL (perhaps the fragment studied in Sistla and Zuck [1993]) for which we can obtain such a complexity bound. We note that it is argued in Lichtenstein and Pnueli [1985] that an exponential time complexity in the size of the specification might be tolerable in practical applications.

There is, however, a fundamental difference between the impact that the guarantee and the assumption have on the complexity of model checking. Both assumption and guarantee are often given as a conjunction of formulas. That is, we are often trying to verify assume-guarantee assertions of the form

$$\langle\varphi_1 \wedge \ldots \wedge \varphi_l\rangle M \langle\psi_1 \wedge \ldots \wedge \psi_m\rangle.$$

Each conjunct expresses a certain property about a module or its environment. Typically, each conjunct is of a rather small size. While it is possible to decompose the guarantee and reduce the problem to verifying assertions of the form $\langle\varphi_1 \wedge \ldots \wedge \varphi_l\rangle M \langle\psi\rangle$, where ψ is of small size, it is not possible in general to decompose the assumption in a similar fashion. Thus, it may seem that in trying to employ modular verification in order to overcome the state-explosion problem, we are merely replacing it with the *assumption-explosion problem*.

This observation provides a justification to the approach taken in Clarke et al. [1989] to avoid the assume-guarantee paradigm. Instead of describing the interaction of the module by an LTL formula, it is proposed there to model the environ-

ment by *interface* processes. As is shown there, these processes are typically much simpler than the full environment of the module. By composing a module with its interface processes and then verifying properties of the composition, it can be guaranteed that these properties will be preserved at the global level.

Regardless of how one interprets our complexity results, we believe that they should renew the discussion on the relative merits of linear versus branching time. For many years, one of the beliefs dominating this discussion has been "model checking for CTL is easy, while model checking for LTL is hard." Our results show that this belief is not valid when one considers modular verification (furthermore, we show that modular model checking is computationally hard even when both assumptions and guarantees are given in ∀CTL). This suggests that the trade-off between CTL and LTL is not a simple trade-off between complexity and expressiveness.

APPENDIX

1. A CONSTRUCTION OF HAA FOR CTL

We describe here the construction of HAA for CTL. Given $\mathcal{D}$ and ψ, we define

$$\mathcal{A}_{\mathcal{D},\psi} = \langle 2^{AP}, \mathcal{D}, cl(\psi), \delta, \{\psi\}, \langle E\tilde{U}(\psi), AU(\psi)\rangle\rangle,$$

where $E\tilde{U}(\psi)$ and $AU(\psi)$ denote the sets of subformulas of ψ of the from $E\varphi_1\tilde{U}\varphi_2$ and $A\varphi_1 U\varphi_2$, respectively, and the transition function δ is defined, for all $\sigma \in 2^{AP}$ and $k \in \mathcal{D}$, as follows.

—$\delta(p,\sigma,k) = \mathbf{true}$ if $p \in \sigma$.
—$\delta(p,\sigma,k) = \mathbf{false}$ if $p \notin \sigma$.
—$\delta(\neg p,\sigma,k) = \mathbf{true}$ if $p \notin \sigma$.
—$\delta(\neg p,\sigma,k) = \mathbf{false}$ if $p \in \sigma$.
—$\delta(\varphi_1 \vee \varphi_2,\sigma,k) = \delta(\varphi_1,\sigma,k) \vee \delta(\varphi_2,\sigma,k)$.
—$\delta(\varphi_1 \wedge \varphi_2,\sigma,k) = \delta(\varphi_1,\sigma,k) \wedge \delta(\varphi_2,\sigma,k)$.
—$\delta(EX\varphi,\sigma,k) = \bigvee_{c=0}^{k-1}(c,\varphi)$.
—$\delta(AX\varphi,\sigma,k) = \bigwedge_{c=0}^{k-1}(c,\varphi)$.
—$\delta(E\varphi_1 U\varphi_2,\sigma,k) = \delta(\varphi_2,\sigma,k) \vee (\delta(\varphi_1,\sigma,k) \wedge \bigvee_{c=0}^{k-1}(c,E\varphi_1 U\varphi_2))$.
—$\delta(A\varphi_1 U\varphi_2,\sigma,k) = \delta(\varphi_2,\sigma,k) \vee (\delta(\varphi_1,\sigma,k) \wedge \bigwedge_{c=0}^{k-1}(c,A\varphi_1 U\varphi_2))$.
—$\delta(E\varphi_1\tilde{U}\varphi_2,\sigma,k) = \delta(\varphi_2,\sigma,k) \wedge (\delta(\varphi_1,\sigma,k) \vee \bigvee_{c=0}^{k-1}(c,E\varphi_1\tilde{U}\varphi_2))$.
—$\delta(A\varphi_1\tilde{U}\varphi_2,\sigma,k) = \delta(\varphi_2,\sigma,k) \wedge (\delta(\varphi_1,\sigma,k) \vee \bigwedge_{c=0}^{k-1}(c,A\varphi_1\tilde{U}\varphi_2))$.

To show that the automaton $\mathcal{A}_{\mathcal{D},\psi}$ is indeed hesitant, we define the following partition of $cl(\psi)$ and a partial order over the sets it induces. Each formula $\varphi \in cl(\psi)$ constitutes a (singleton) set $\{\varphi\}$ in the partition. The partial order is then defined by $\{\varphi_1\} \leq \{\varphi_2\}$ if and only if $\varphi_1 \in cl(\varphi_2)$. Since each transition of the automaton from a state φ leads to states associated with formulas in $cl(\varphi)$, the first condition holds. Consider a set $\{\varphi\}$ in the partition. It is easy to see that when φ is an atomic proposition or its negation, a Boolean assertion, or is of the form $EX\varphi'$ or $AX\varphi'$, the set $\{\varphi\}$ is transient. Indeed, the transition from the state φ leads to states associated with strict subformulas of φ. Also, when φ is of the form

$E\varphi_1 U\varphi_2$ or $E\varphi_1 \tilde{U}\varphi_2$, the set $\{\varphi\}$ is existential. Indeed, if we write the transition from the state φ in a disjunctive normal form, each disjunct has at most one atom whose state is φ, and all other atoms involve states that are strict subformulas of φ. Similarly, when φ is of the form $A\varphi_1 U\varphi_2$ or $A\varphi_1 \tilde{U}\varphi_2$, the set $\{\varphi\}$ is universal.

2. THE 1-LETTER NONEMPTINESS PROBLEM FOR LAA

THEOREM 2.1. *The 1-letter nonemptiness problem for a LAA of hesitant size n, degree k, and depth m can be solved in space $O(m(log^2 n + \log k))$.*

PROOF. Consider a LAA with $\alpha = \langle B, G\rangle$ and $\beta = \{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$. The property of LAA we use is that, from a state in Q_i, it is possible to search for another reachable state in the same Q_i using space $O(m \log^2 n)$. For a transient Q_i, there are no such states. For universal and existential Q_i, the exact notion of reachability we use is the transitive closure of the following notion of *immediate reachability*. Consider a set Q_i and assume that we have already assigned a Boolean value for all states in sets lower than Q_i. Then, a state $q' \in Q$ is immediately reachable from a state q, if it appears in the transition from q when this transition has been simplified using the assigned Boolean values for states in lower Q_i's. Note that the simplified transition is always a disjunction for a state of an existential Q_i, and a conjunction for a state of a universal Q_i.

We call a state q' of Q_i *provably true* if, when the procedure is applied to the successors of q' that are not in Q_i, and the Boolean expression for the transition from q' is simplified, it is identically true. States that are *provably false* are defined analogously.

The following recursive procedure labels the states of the automaton with 'T' (accepts) or 'F' (does not accept). The language of $\mathcal{A}$ is then nonempty if and only if some initial state is labeled with 'T'.

(1) Start at the initial state.

(2) At a transient state q, evaluate the transition from q by recursively applying the procedure to the successor states of q. Label the state with the Boolean value that is obtained for the transition.

(3) At a state q of an existential Q_i, proceed as follows.
   (a) Search for a reachable state q' of the same Q_i that is provably true (note that this requires applying the procedure recursively to all states from lower Q_i's that are touched by the search). If such a state q' is found, label q with 'T'.
   (b) If no such state exists, guess $1 \leq j \leq k$ and search for states q' and q'' of Q_i such that the following holds:
      i. $q' \in G$.
      ii. $q'' \in G_j$.
      iii. q' is reachable from q.
      iv. q'' is $Q_i \setminus B_j$-reachable from q'.
      v. q' is $Q_i \setminus B_j$-reachable from q''.

      If such q' and q'' are found (possibly $q' = q''$), label q with 'T'.
   (c) if none of the first two cases apply, label q with 'F'.

(4) At a state q of a universal Q_i, proceed as follows.

(a) Search for a reachable state q' of the same Q_i that is provably false. If such a state q' is found, label q with 'F'.
(b) If no such state exists, guess $1 \leq j \leq k$ and search for states q' and q'' of Q_i such that the following holds:
   i. $q' \in B$.
   ii. $q'' \in G_j$.
   iii. q' is reachable from q.
   iv. q'' is $Q_i \setminus B_j$-reachable from q'.
   v. q' is $Q_i \setminus B_j$-reachable from q''.
   If such q' and q'' are found, label q with 'F'.
(c) if none of the first two cases apply, label q with 'T'.

With every state q we can associate a finite integer, $rank(q)$, corresponding to the depth of the recursion required in order to label q. We prove, by induction on $rank(q)$, that q is labeled correctly. Assume first that $rank(q) = 0$. Then, q belongs to a transient set, and $\delta(q, a)$ is equivalent to **true** or **false**. Hence, according to Step (2) of the algorithm, q is labeled correctly. Assume now that $rank(q) = l + 1$, and that all the states q' with $rank(q') \leq l$ are labeled correctly. We distinguish between three cases.

(1) When q belongs to a transient set, the recursive application of the algorithm that is performed on Step (2) returns, by the induction hypothesis, correct values to all the states reachable from q, and hence q is labeled correctly.
(2) When q belongs to an existential set Q_i, the algorithm follows Step (3). If the state q' that searched for in Step (3.a) is found, then, by the induction hypothesis, q' is accepting, and hence, as Q_i is an existential set and q' is reachable from q, the state q is accepting as well. If such state q' is not found, yet states q' and q'' as required in Step (3.b) are found, it follows that Q_i contains a cycle reachable from q, such that the infinite path obtained by unwinding the cycle satisfies both α and β. Hence, the state q is accepting. Finally, if both Steps (3.a) and (3.b) fail, then all the paths that start in q either leave Q_i and reach rejecting states, or stay in Q_i and not satisfy either α or β. Hence, q is rejecting.
(3) When q belongs to a universal set Q_i, the algorithm follows Step (4), and the arguments are dual to these in Step (3).

We now consider the complexity of our algorithm. We show that for each state q, the labeling of q involves a recursion of depth at most m, where each step of the recursion can be executed deterministically in space $O(\log^2 n)$. We start with q which belongs to a transient set. There, we have to evaluate the transition from q. It is known that evaluation of Boolean expressions can be done using space that is logarithmic in the size of the expression [Lynch 1977]. Here, we evaluate expressions over Q, using a recursion to get their Boolean value. Since each recursive call takes us to a lower Q_i, the depth of the recursion is bounded by m.

Consider now a state $q \in Q_i$ for an existential set Q_i. For each state $q' \in Q_i$, we have that q' is provably true if and only if the transition from q' evaluates to **true** when we assign **false** to all the states in Q_i (and evaluates, using a recursion, states in lower sets). Checking the latter is as simple as evaluating a transition

from a transient state. Thus, assuming all values detected by recursion are already assigned, determining whether a certain state is provably true can be done, as in Lynch [1977], in logarithmic space.

For each two states q' and q'' in Q_i, we have that q'' is immediately reachable from q' if and only if the following two conditions hold. First, the transition from q' evaluates to **true** when we assign **true** to q'' and assign **false** to all the other states in Q_i (and evaluate recursively states from lower sets). Second, the transition from q' evaluates to **false** when we assign **false** to all the states in Q_i (and evaluate recursively states from lower sets). Again, checking this is as simple as evaluating a transition from a transient state. Thus, assuming all values detected by recursion are already assigned, determining whether q'' is immediately reachable from q' can be done, as in Lynch [1977], in logarithmic space.

It is known, by Jones [1975], that the graph accessibility problem is in NLOGSPACE. Thus, by Savitch's Theorem [Savitch 1970], checking that q'' is accessible from q' can be done deterministically in space $O(\log^2 n)$. Now, to check whether q'' is reachable from q', we restrict the graph accessibility test, replacing immediate accessibility with immediate reachability. Thus, assuming all values detected by recursion are already assigned, determining whether a certain state in Q_i is reachable from another certain state in Q_i can be done in space $O(\log^2 n)$. Since the reachability checks in step (3.b) are done independently for each guessed $1 \leq j \leq k$, a determinization of this step involves additional $\log k$ bits, required for a sequential check of all pairs. Hence, as the labeling of $q \in Q_i$ only involves a search for reachable states, we can determine its labeling in space $O(m(\log^2 n + \log k))$.

The case where $q \in Q_i$ for a universal set Q_i is symmetric. □

THEOREM 2.2. *The 1-letter nonemptiness problem for a LAA is decidable in time that is linear in both its hesitant size and its degree.*

PROOF. Given a LAA $\mathcal{A} = \langle \{a\}, Q, \delta, q_0, \langle G, B\rangle, \beta\rangle$ with $\beta = \{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$, let h be the hesitant size of $\mathcal{A}$. We present an algorithm with running time $O(hk)$ for checking the nonemptiness of the language of $\mathcal{A}$. The algorithm is similar to the one described in Kupferman et al. [2000] for HAA. The only change is that here, being trapped in an existential set, the automaton should be able to visit states from G infinitely often, and in addition, there should exist $1 \leq l \leq k$ such that the automaton visits G_l infinitely often and visits B_l only finitely often. Dually, being trapped in a universal set, the automaton should be able to visit states from B only finitely often, or, for all $1 \leq l \leq k$, the automaton should be able to either visit G_l only finitely often or visit B_l infinitely often. These checks are similar to these required in order to check the nonemptiness of a nondeterministic Rabin automaton over words.

The algorithm labels the states of $\mathcal{A}$ with either 'T' (accept) or 'F' (does not accept). As $\mathcal{A}$ is a LAA, there exists a partition of Q into disjoint sets Q_i such that there exists a partial order $\leq$ on the collection of the Q_i's and such that for every $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, a)$, we have that $Q_j \leq Q_i$. Thus, transitions from a state in Q_i lead to states in either the same Q_i or a lower one. The algorithm works in phases and proceeds up the partial order. Thus, when the algorithm reaches a set Q_i, all the states in all sets Q_j's, for which $Q_j < Q_i$, have already been labeled. Whenever a state q' is labeled, transitions of other states q

for which q' occurs in $\delta(q,a)$ are simplified accordingly. Note that as a result, q may be labeled too.

Let $Q_1 \leq \ldots \leq Q_n$ be an extension of the partial order to a total order. In each phase i, the algorithm handles states from the minimal set Q_i that still has not been labeled. For a transient set Q_i, the above implies that the states in Q_i have already been labeled too. For existential and universal sets, the algorithm proceeds as follows.

—In an existential set Q_i, all the transitions from states in Q_i only contain disjunctively related elements of Q_i. So, when the algorithm reaches Q_i, all its simplified transitions are disjunctions and induce an OR-graph that has states in Q_i as nodes. For each pair $\langle G_l, B_l\rangle \in \beta$, the algorithm proceeds as follows. The algorithm partitions the graph $Q_i \setminus B_l$ into maximal strongly connected components Q_i^j. Since the partition is maximal, there exists a partial order over these component ($Q_i^j \leq Q_i^k$ if and only if there exists a transition from Q_i^k to Q_i^j) and there exists an extension of it into a total order $Q_i^1 \leq \ldots \leq Q_i^{m_i}$. The algorithm proceeds up this total order. For each such component Q_i^j, the algorithm checks whether $Q_i^j \cap G \neq \emptyset$ and $Q_i^j \cap G_i \neq \emptyset$. If these hold, all the states in Q_i^j are labeled 'T'. Once a state q is labeled with 'T', transition functions in which q occurs are simplified accordingly, i.e., a disjunction with a disjunct 'T' is simplified to 'T'. Consequently, a transition function $\delta(q',a)$ for some q' (not necessarily from Q_i) can be simplified to **true**. The state q' is then labeled, and simplification propagates further. Once all the pairs in β are handled, the algorithm labels with 'F' all the states in Q_i that were not labeled 'T', and, as before, labeling is propagated.

—For a universal set, the transitions induce an AND-graph, and the algorithm proceeds in a dual way. For each pair $\langle G_l, B_l\rangle \in \beta$, the algorithm partitions the graph $Q_i \setminus B_l$ into maximal strongly connected components Q_i^j. For each such component Q_i^j, the algorithm checks whether $Q_i^j \cap B \neq \emptyset$ and $Q_i^j \cap G_i \neq \emptyset$. If these hold, all the states in Q_i^j are labeled 'F', and labeling is propagated. Once all the pairs in β are handled, the algorithm labels with 'T' all the states in Q_i that were not labeled 'F', and, as before, labeling is propagated.

Consider the total order $Q_1^1 \leq Q_1^2 \leq \ldots Q_1^{m_1} \leq Q_2^1 \leq \ldots \leq Q_n^{m_n}$. One can prove, that for all $1 \leq i \leq n$ and $1 \leq j \leq m_i$, all the states in Q_i^j are labeled correctly. The proof proceeds by induction on (i,j) (with the ordering $(i_1,j_1) < (i_2,j_2)$ if and only if $i_1 < i_2$ or $i_1 = i_2$ and $j_1 < j_2$), and it is similar to these used in the proof described in Kupferman et al. [2000] for HAA. Using an AND/OR graph, as suggested in Beeri [1980], the algorithm can be implemented in linear running time. The graph, G, induced by the transition function, maintains the labeling and the propagation of labeling performed during the algorithm execution. □

3. MAXIMAL MODELS FOR CTL* FORMULAS

THEOREM 3.1. *For every $\forall$CTL* formula φ, there exists a maximal model M_φ of size $2^{2^{O(|\varphi|)}}$.*

PROOF. For a $\forall$CTL* formula φ, let $sf(\varphi)$ denote the set of state subformulas of φ. Given φ, let $\forall(\varphi) \subseteq sf(\varphi)$ denote the set of all the state subformulas of φ of

the form $A\xi$. Let $\mathcal{A}_{\forall(\varphi)}$ be a Büchi ω-automaton over $\Sigma = 2^{sf(\varphi)}$ such that $\mathcal{A}_{\forall(\varphi)}$ accepts an infinite word $\pi = w_0, w_1, \ldots$ if and only if there exists a suffix $w_i, w_{i+1}, \ldots$ of π and a formula $A\xi \in \forall(\varphi)$ such that $A\xi \in w_i$ and $w_i, w_{i+1}, \ldots$ does not satisfy ξ. Technically, $\mathcal{A}_{\forall(\varphi)}$ nondeterministically guesses a location i and a formula $A\xi$ and then follows the Büchi ω-automaton of $\neg\xi$. Consequently, if $w_i, w_{i+1}, \ldots$ does not satisfy ξ, the automaton $\mathcal{A}_{\forall(\varphi)}$ would accept π. By Theorem 2.4, such $\mathcal{A}_{\forall(\varphi)}$ of size $2^{O(|\varphi|)}$ exists. Note that though ξ is a path formula of a branching temporal logic, we interpret it here over linear sequences. Since these sequences are labeled with all the state subformulas of ξ, this causes no difficulty, as we can regard the state subformulas of ξ as atomic propositions and regard ξ as a linear temporal logic formula.

We now take $\mathcal{A}_{\forall(\varphi)}$ and co-determinize it (that is, we construct a deterministic automaton that complements it). The automaton we get, called $\overline{\mathcal{A}_{\forall(\varphi)}}$, is a deterministic Rabin automaton that accepts exactly all the words $\pi = w_0, w_1, \ldots$ for which, if a state w_i is labeled with some $A\xi \in \forall(\varphi)$, then ξ is satisfied in the suffix $w_i, w_{i+1}, \ldots$ of π. By Safra [1989], the automaton $\overline{\mathcal{A}_{\forall(\varphi)}}$ is of size $2^{2^{O(|\varphi|)}}$.

For a set $s \subseteq sf(\varphi)$, we say that s is *consistent* if and only if the following four conditions hold:

(1) For every $p \in AP$, if $p \in s$, then $\neg p \notin s$.

(2) For every $p \in AP$, if $\neg p \in s$, then $p \notin s$.

(3) For every $\varphi_1 \wedge \varphi_2 \in s$, we have that $\varphi_1 \in s$ and $\varphi_2 \in s$.

(4) For every $\varphi_1 \vee \varphi_2 \in s$, we have that $\varphi_1 \in s$ or $\varphi_2 \in s$.

Let $c(\varphi)$ denote the set of all consistent subsets of $sf(\varphi)$. Consider the module

$$M = \langle AP, c(\varphi), c(\varphi) \times c(\varphi), W_0, L \rangle,$$

where the initial set W_0 includes all states $w \in c(\varphi)$ for which $\varphi \in w$ (note that if φ is satisfiable, the set W_0 is not empty), and for every $w \in c(\varphi)$, we have that $L(w) = w \cap AP$. That is, M is more general than any model of φ, yet it is not necessarily a model of φ. To make it a maximal model, we take the product of M with $\overline{\mathcal{A}_{\forall(\varphi)}}$ as follows. Let $\overline{\mathcal{A}_{\forall(\varphi)}} = \langle \Sigma, Q, \delta, q_0, \beta \rangle$, where $\beta = \{\langle G_1, B_1 \rangle, \ldots, \langle G_k, B_k \rangle\}$. Then, $M_\varphi = \langle AP, c(\varphi) \times Q, R, W_0 \times \{q_0\}, L', \beta' \rangle$, where R, L', and β' are defined as follows.

—$R = \{\langle \langle w, q \rangle, \langle w', q' \rangle \rangle : \delta(q, w) = q'\}$.

—For all $w \in c(\varphi)$ and $q \in Q$, we have $L'(\langle w, q \rangle) = L(w)$.

—$\beta' = \{\langle c(\varphi) \times G_1, c(\varphi) \times B_1 \rangle, \ldots, \langle c(\varphi) \times G_k, c(\varphi) \times B_k \rangle\}$.

We now prove the correctness of our construction. That is, we show that $M_\varphi \models \varphi$ and that for all fair modules M, we have $M \models \varphi$ only if $M \leq M_\varphi$ We first prove that $M_\varphi \models \varphi$. More precisely, we prove that for every reachable state $\langle w, q \rangle \in c(\varphi) \times Q$, and for every formula $\psi \in w$, we have that $\langle w, q \rangle \models \psi$. The proof proceeds easily by induction on the structure of ψ. In particular, satisfaction of formulas of the form $A\xi$ follows from the product with $\overline{\mathcal{A}_{\forall(\varphi)}}$. To see this, consider a state $\langle w, q \rangle$ and a formula $A\xi \in w$. Let $\langle w_1, q_1 \rangle, \langle w_2, q_2 \rangle, \ldots$ be a fair computation of M_φ that starts in $\langle w, q \rangle$, i.e., $\langle w_1, q_1 \rangle = \langle w, q \rangle$. By the definition of R and β', the sequence $w_1, w_2, \ldots$

is a suffix of a word accepted by $\overline{\mathcal{A}_{\forall(\varphi)}}$. Hence, for all formulas of the form $A\xi' \in w$, the computation $w_1, w_2, \ldots$ satisfies ξ'. Thus, in particular, $w_1, w_2, \ldots$ satisfies ξ.

Consider now a fair module $M = \langle AP, W_M, R_M, W_M^0, L_M, \alpha_M \rangle$, and assume that $M \models \varphi$. We show a simulation H from M to M_φ. For every state $w \in W_M$, define $f(w)$ to be the set in $c(\varphi)$ of state formulas that are true in w. The simulation H is the smallest set that satisfies the following:

—For every $w \in W_M^0$, we have $H(w, \langle f(w), q_0 \rangle)$.

—For every w_1, w_2 in W_M and $\langle f(w_1), q_1 \rangle \in c(\varphi) \times Q$ such that $\langle w_1, w_2 \rangle \in R_M$ and $H(w_1, \langle f(w_1), q_1 \rangle)$, we have $H(w_2, \langle f(w_2), \delta(q_1, f(w_1)) \rangle)$.

We prove that H is indeed a simulation from M to M_φ. That is, we prove that for all $w \in W_M^0$, there exists $w' \in W_0 \times \{q_0\}$ such that H is a simulation relation from $\langle M, w \rangle$ to $\langle M_\varphi, w' \rangle$. Consider a state $w \in W_M^0$. Since $M \models \varphi$, then, by the definition of $f(w)$ and W_0, we have $\langle f(w), q_0 \rangle \in W_0 \times \{q_0\}$, and hence, by the definition of H, we have $H(w, \langle f(w), q_0 \rangle)$. Now, let $w \in W_M$ and $\langle f(w), q \rangle \in c(\varphi) \times Q$ be such that $H(w, \langle f(w), q \rangle)$. By the definition of H, all the pairs in H are of this form. By the definition of L', we have that $L'(\langle f(w), q \rangle) = L_M(w)$. So, the first requirement on pairs in a simulation holds. For the second requirement, assume $H(w, \langle f(w), q \rangle)$, and let $\pi = w_0, w_1, \ldots$ be a fair computation in M with $w_0 = w$. Consider the computation $\pi' = \langle f(w), q \rangle, \langle f(w_1), q_1 \rangle, \ldots$ where $q_1 = \delta(q, f(w))$ and where for every $i \geq 1$ we have $q_{i+1} = \delta(q_i, f(w_i))$. By the definition of H, we have that for all $i \geq 1$ we have $H(w_i, \langle f(w_i), q_i \rangle)$. So, it remains to show that π' is fair in M_φ. Since $M \models \varphi$ and π is fair, then for each state w_i and formula $A\xi \in \forall(\varphi)$ such that $A\xi \in f(w_i)$, we have that ξ is satisfied in $w_i, w_{i+1}, \ldots$. Thus, $q_i, q_{i+1}, \ldots$ is an accepting run of $\overline{\mathcal{A}_{\forall(\varphi)}}$ with q_i as an initial state over $w_i, w_{i+1}, \ldots$. Therefore, by the definition of β', the computation π' is fair. □

4. EXPSPACE LOWER BOUND FOR THE IMPLICATION PROBLEM FOR ACTL*

To prove the EXPSPACE lower bound for the implication problem for ∀CTL*, we do a reduction from the problem whether an exponential-space deterministic Turing machine T accepts an input word x. That is, given T and x, we construct two ∀CTL* formulas φ and ψ such that T accepts x if and only if φ does not imply ψ. In fact we will prove a stronger lower bound. Given T and x, we construct an LTL formula ξ and an ∃CTL formula θ such that the length of ξ is polynomial in the size of T and the length of x, the length of θ is fixed, and T accepts an input word x if and only if the formula $A\xi \wedge \theta$ is satisfiable. Thus, taking $\varphi = A\xi$ and $\psi = \neg\theta$, we have that T accepts x if and only if the implication $\varphi \rightarrow \psi$ does not hold. We call the LTL formula ξ the *assumption* and the ∃CTL formula θ the *guarantee*. By taking T to be a Turing machine that accepts an EXPSPACE-complete language, we can fix T and vary only x.

Let $T = \langle \Gamma, Q, \rightarrow, q_0, F \rangle$, where Γ is the alphabet; Q is the set of states; $\rightarrow \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L, R\}$ is the transition relation (we use $(q, a) \rightarrow (q', b, \Delta)$ to indicate that when T is in state q and it reads the input a in the current tape cell, it moves to state q', writes b in the current tape cell, and its reading head moves one cell to the left or to the right, according to Δ); q_0 is the initial state; and $F \subseteq Q$ is the set of accepting states. Let $n = a \cdot |x|$, for some constant a, be such that the working tape of T has 2^n cells. We encode a configuration of T by a word

$\gamma_1\gamma_2 \ldots (q, \gamma_i) \ldots \gamma_{2^n}$. That is, all the letters in the configuration are in Γ, except for one letter in $Q \times \Gamma$. The meaning of such a configuration is that the j's cell of T, for $1 \leq j \leq 2^n$, is labeled γ_j, the reading head points on cell i, and T is in state q. For example, the initial configuration of T is $(q_0, x_1)x_2 \cdots x_n \# \# \cdots$, where $\#$ stands for the empty cell. We can now encode a computation of T by a sequence of configurations.

Let $\Sigma = \Gamma \cup (Q \times \Gamma)$. We can encode letters in Σ by a set $AP(T) = \{p_1, \ldots, p_m\}$ (with $m = \lceil log|\Sigma| \rceil$) of atomic propositions. We define our formulas over the set $AP = AP(T) \cup \{b, c, d, l, r\}$ of atomic propositions. The task of the last five atoms will be explained shortly. Since T is fixed, so is Σ, and hence so is the size of AP.

Consider an infinite sequence π over 2^{AP}. For an atomic proposition $p \in AP$ and a node u in π, we use $p(u)$ to denote the truth value of p at u. That is, $p(u)$ is 1 if p holds at u and is 0 if p does not hold at u. We divide the sequence π to blocks of length n. Every such block corresponds to a single tape cell of the machine T. Consider a block $u_1, \ldots, u_n$ that corresponds to a cell ρ. We use the node u_n to encode the content of cell ρ. Thus, the bit vector $p_1(u_n), \ldots, p_m(u_n)$ encodes the letter (in $\Gamma \cup (Q \times \Gamma)$) that corresponds to cell ρ. We use the atomic proposition b to mark the beginning of the block; that is, b should hold on u_1 and fail on $u_2, \ldots, u_n$. This is be enforced by the LTL assumption ξ. Thus, ξ contains a conjunct

$$b \wedge X(\neg b \wedge X(\neg b \wedge \cdots \wedge X\neg b) \cdots) \wedge G(b \leftrightarrow X^n b).$$

Recall that the letter with which cell ρ is labeled is encoded at the node u_n of the block $u_1, \ldots, u_n$ that corresponds to ρ. Why then do we need a block of length n to encode a single letter? The block also encodes the location of the cell ρ on the tape. Since T is an exponential-space Turing machine, this location is a number between 0 and $2^n - 1$. Encoding the location eliminates the need for exponentially many X operators when we attempt to relate two successive configurations. Encoding is done by the atomic proposition c, called *counter*. Let $c(u_n), \ldots, c(u_1)$ encode the location of ρ. Note that, for technical convenience, the least significant bit of the counter is in u_1. A sequence of 2^n blocks corresponds to 2^n cells and encodes a configuration of T. The value of the counters along this sequence goes from 0 to $2^n - 1$, and then start again from 0. This is enforced by the LTL assumption ξ. To keep the size of φ be $O(n)$, we need also an atomic proposition d that acts as a "carry" bit. Formally, ξ contains the following conjuncts.

(1) The counter starts at 0.
—$\neg c \wedge X(\neg c \wedge X(\neg c \ldots \wedge X\neg c) \cdots)$.

(2) The counter is increased properly. Note that as we always want to increase it by 1 we take b as a carry to the least significant bit.
—$G(((b \vee d) \wedge \neg c) \rightarrow (X(\neg d) \wedge X^n c))$.
—$G((\neg(b \vee d) \wedge \neg c) \rightarrow (X(\neg d) \wedge X^n \neg c))$.
—$G(((b \vee d) \wedge c) \rightarrow (Xd \wedge X^n \neg c))$.
—$G((\neg(b \vee d) \wedge c) \rightarrow ((X\neg d) \wedge X^n c))$.

An atomic proposition l marks the last node of a configuration, i.e., l holds in a node u_n of a block $u_1, \ldots, u_n$ if and only if c holds on all nodes in the block. This is enforced by the following conjunct in ξ, stating that l holds in a node u_n that

precedes a block with counter 0:

$$G(l \leftrightarrow Xb \wedge (X(\neg c) \wedge X((\neg c) \wedge X(\cdots \neg c)\cdots))).$$

Let $\sigma_1 \ldots \sigma_{2^n}, \sigma'_1 \ldots \sigma'_{2^n}$ be two successive configurations of T. For each triple $\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle$ with $1 \leq i \leq 2^n$ (taking σ_{2^n+1} to be σ'_1 and σ_0 to be the label of the last cell in the configuration before $\sigma_1 \ldots \sigma_{2^n}$, or some special label when $\sigma_1 \ldots \sigma_{2^n}$ is the initial configuration), we know, by the deterministic transition relation of T, what σ'_i should be. Let $next(\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle)$ denote our expectation for σ'_i, i.e., [3],

—$next(\langle \gamma_{i-1}, \gamma_i, \gamma_{i+1} \rangle) = \gamma_i$.

—$next(\langle (q, \gamma_{i-1}), \gamma_i, \gamma_{i+1} \rangle) = \begin{cases} \gamma_i & \text{If } (q, \gamma_{i-1}) \rightarrow (q', \gamma'_{i-1}, L). \\ (q', \gamma_i) & \text{If } (q, \gamma_{i-1}) \rightarrow (q', \gamma'_{i-1}, R). \end{cases}$

—$next(\langle \gamma_{i-1}, (q, \gamma_i), \gamma_{i+1} \rangle) = \gamma'_i$ where $(q, \gamma_i) \rightarrow (q', \gamma'_i, \Delta)$.

—$next(\langle \gamma_{i-1}, \gamma_i, (q, \gamma_{i+1}) \rangle) = \begin{cases} \gamma_i & \text{If } (q, \gamma_{i+1}) \rightarrow (q', \gamma'_{i+1}, R). \\ (q', \gamma_i) & \text{If } (q, \gamma_{i+1}) \rightarrow (q', \gamma'_i, L). \end{cases}$

Consistency with $next$ will give us a necessary condition for a word to encode a legal computation. In addition, the computation should start with the initial configuration. Finally, if the computation ends with an accepting configuration (that is, one with (q, γ_i) with $q \in F$), then T accepts x. It is easy to specify in LTL the requirements about the initial and accepting configurations. For a letter $\sigma \in \Sigma$, let $\eta(\sigma)$ be the propositional formula over AP that encodes σ. That is, $\eta(\sigma)$ holds in node u_n of a block that encodes cell ρ if and only if the cell ρ is labeled σ. Then,

$$X^n(\eta(q_0, x_1) \wedge X^n(\eta(x_2) \cdots \wedge X^n(\eta(x_n) \wedge X^n(\eta(\#) \wedge [(\eta(\#) \rightarrow X^n \eta(\#)) U (\eta(\#) \wedge l)])) \cdots))$$

requires the computation to start with the initial configuration, and

$$F((Xb) \wedge \bigvee_{q \in F, \gamma \in \Gamma} \eta(q, \gamma))$$

requires the computation to reach an accepting configuration.

The difficult part in the reduction is in guaranteeing that the sequence of configurations is indeed consistent with $next$. To enforce this, we have to relate σ_{i-1}, σ_i, and σ_{i+1} with σ'_i for any i in any two successive configurations $\sigma_1 \ldots \sigma_{2^n}, \sigma'_1 \ldots \sigma'_{2^n}$. One natural way to do so is by a conjunction of formulas like "whenever we meet a cell with counter $i-1$ and the labeling of the next three cells forms the triple $\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle$, then the next time we meet a cell with counter i, this cell is labeled $next(\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle)$." The problem is that as i can take any value from 1 to 2^n, there are exponentially many such conjuncts. Therefore, we should find a way to relate $\langle \sigma_{i-1}, \sigma_i, \sigma_{i+1} \rangle$ with σ'_i without saying explicitly what i is. To do this, we should be able to point simultaneously on two (exponentially far) points in the computations. The way to do it, as suggested in Vardi and Stockmeyer [1985], is by marking one of the two positions with a "pebble," as explained below.

Consider the horizontal computation in Figure 2. Each of its nodes branches into a diagonal suffix. Let us call the bold computation a *real* computation and

[3] We assume that T's head does not "fall" from the right or the left boundaries of the tape. Thus, the case where $i = 1$ and $(q, \gamma_i) \rightarrow (q', \gamma'_i, L)$ and the dual case where $i = 2^n$ and $(q, \gamma_i) \rightarrow (q', \gamma'_i, R)$ are not possible.

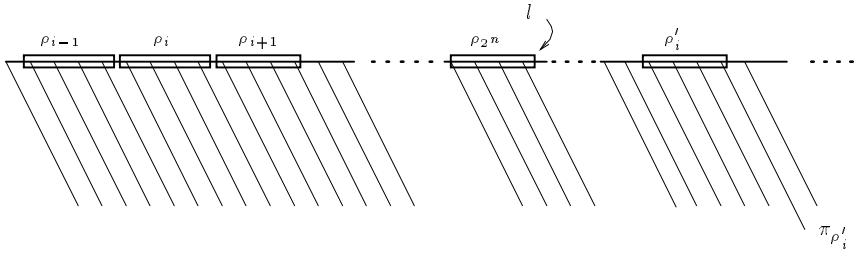

Fig. 2. A real computation branching into auxiliary computations.

call all the computations that branch to a diagonal suffix *auxiliary* computations. Let us assume that the atomic proposition r holds at each of the nodes of the real computation and does not hold in each of the nodes of the diagonal suffixes. Then, the real computation satisfies the LTL formula Gr and the auxiliary computations satisfy the LTL formula $F(r \wedge XG\neg r)$. There is exactly one point in each auxiliary computation in which the eventuality of the above formula is satisfied: the point where the computation branches from the real path.

Consider now the three successive blocks $\rho_{i-1}, \rho_i, \rho_{i+1}$ along the real computation, and the block ρ'_i that follows (we associate blocks with the cells they encode). Assume that there is only one position between them in which l holds. Then, the labeling σ'_i of ρ'_i should be $next(\sigma_{i-1}, \sigma_i, \sigma_{i+1})$, where σ_{i-1}, σ_i, and σ_{i+1} are the labeling of ρ_{i-1}, ρ_i, and ρ_{i+1}, respectively. Let $\pi_{\rho'_i}$ be the auxiliary computation that leaves the real computation at the last position of the block ρ'_i. We use this auxiliary computation in order to check that σ'_i indeed equals $next(\sigma_{i-1}, \sigma_i, \sigma_{i+1})$. In the assumption ξ, we have a conjunction that states the following: it is always true that if

(1) we are in an auxiliary computation, still in its "real prefix",

(2) we are in a beginning of a block,

(3) the value of the counter in the next block equals the value of the counter in the block where computation becomes auxiliary, and

(4) there is only one occurrence of l between the current position and the position where the computation becomes auxiliary,
then

(5) if the labels of the next three blocks are σ_{i-1}, σ_i, and σ_{i+1}, then the block where the computation becomes auxiliary is labeled by $next(\sigma_{i-1}, \sigma_i, \sigma_{i+1})$.

Using b, c, l, and r, we can easily express (1)-(5) with formulas of length polynomial in n (recall that as the set Σ is fixed, scanning all the possible labels of a cell can be done with a formula of a fixed length). Note that for expressing (3), we need to compare the value of the two counters bit by bit. Note also that the same formula takes care of all the auxiliary computations; thus one formula checks consistency with *next* for all the cells along the real computation. Formally, we have in ξ the implication $G(((1) \wedge (2) \wedge (3) \wedge (4)) \rightarrow (5))$, where (1)-(5) are as follows.

(1) $r \wedge F\neg r$

(2) b

(3)

$$\begin{array}{c} X^n[(c \rightarrow F(c \wedge X^n(r \wedge XG\neg r))) \wedge ((\neg c) \rightarrow F((\neg c) \wedge X^n(r \wedge XG\neg r)))\wedge \\ X(c \rightarrow F(c \wedge X^{n-1}(r \wedge XG\neg r))) \wedge ((\neg c) \rightarrow F((\neg c) \wedge X^{n-1}(r \wedge XG\neg r)))\wedge \\ X(c \rightarrow F(c \wedge X^{n-2}(r \wedge XG\neg r))) \wedge ((\neg c) \rightarrow F((\neg c) \wedge X^{n-2}(r \wedge XG\neg r)))\wedge \\ \vdots \\ X(c \rightarrow F(c \wedge r \wedge XG\neg r)) \wedge ((\neg c) \rightarrow F((\neg c) \wedge r \wedge XG\neg r))] \end{array}$$

(4) $(\neg l)U(l \wedge X((\neg l)U(Xb \wedge r \wedge XG\neg r)))$.

(5) $\bigwedge_{\sigma_1,\sigma_2,\sigma_3 \in \Sigma} X^{n-1}(\eta(\sigma_1) \wedge X^n(\eta(\sigma_2) \wedge X^n\eta(\sigma_3))) \wedge F(\eta(next(\sigma_1,\sigma_2,\sigma_3)) \wedge r \wedge XG\neg r)$

Consider an infinite 2^{AP}-labeled tree. If all the computations of the tree satisfy the assumption, then a real computation in the tree that branches to auxiliary computations whenever it is in a location that corresponds to an end of a block encodes an accepting computation of T on x. The existence of such a computation is enforced by the $\exists$CTL guarantee $\theta = EG(r \wedge EXEG\neg r)$ (for simplicity, we require the real computation to branch to auxiliary computations everywhere). Hence, $A\xi \wedge \theta$ is satisfiable if and only if T accepts x.

5. LINEAR-SIZE PROPERTY FOR $\exists$CTL

Each $\exists$CTL formula has one of the following forms: **true**, **false**, $p \in AP$, $\neg p$ for $p \in AP$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, $EX\varphi_1$, $E\varphi_1 U \varphi_2$, or $E\varphi_1 \tilde{U} \varphi_2$, where φ_1 and φ_2 are $\exists$CTL formulas. With each $\exists$CTL formula φ, we associate a set S_φ of fair modules that satisfy φ. The definition of S_φ proceeds by induction on the structure of φ. Each fair module in S_φ has a single initial state. We use $S_{\varphi_1} \rightarrow S_{\varphi_2}$ to denote the set of all fair modules obtained by taking a fair module M_1 from S_{φ_1}, a fair module M_2 from S_{φ_2}, adding a transition from the initial state of M_1 to the initial state of M_2, and fixing the initial state to be the one of M_1. We use $S_{\varphi_2}\downarrow$ to denote the set of all fair modules obtained by taking a fair module from S_{φ_2} and adding a self loop to its initial state.

—$S_{\mathbf{true}}$ is the set of all one-state fair modules over AP.

—$S_{\mathbf{false}} = \emptyset$.

—S_p for $p \in AP$ is the set of all one-state fair modules over AP in which p holds.

—$S_{\neg p}$ for $\neg p \in AP$ is the set of all one-state fair modules over AP in which p does not hold.

—$S_{\varphi_1 \vee \varphi_2} = S_{\varphi_1} \cup S_{\varphi_2}$.

—$S_{\varphi_1 \wedge \varphi_2} = S_{\varphi_1} \cap S_{\varphi_2}$.

—$S_{EX\varphi_1} = S_{\mathit{true}} \rightarrow S_{\varphi_1}$.

—$S_{E\varphi_1 U \varphi_2} = S_{\varphi_2} \cup (S_{\varphi_1} \rightarrow S_{\varphi_2})$.

—$S_{E\varphi_1 \tilde{U} \varphi_2} = S_{\varphi_2}\downarrow$.

The fair modules in S_φ are "economical" with respect to states that are required for satisfaction of formulas that refer to the strict future. For example, since the

initial state of each fair module that satisfies $E\varphi_1 U \varphi_2$ must satisfy either φ_1 or φ_2, the set of initial states of fair modules in $S_{E\varphi_1 U \varphi_2}$ is equal to the set of initial states of fair modules in S_{φ_1} or in S_{φ_2}. In addition, fair modules in $S_{E\varphi_1 U \varphi_2}$ that do not satisfy φ_2 in their initial state are required to satisfy φ_2 in a successor state. Following the same idea, since the initial state of each fair module that satisfies $E\varphi_1 \tilde{U} \varphi_2$ must satisfy φ_2, the set of initial states of fair modules in $S_{E\varphi_1 \tilde{U} \varphi_2}$ is equal to the set of initial states of fair modules in S_{φ_2}. In addition, all the fair modules in $S_{E\varphi_1 \tilde{U} \varphi_2}$ satisfy $EG\varphi_2$, which is the most economical way to satisfy $E\varphi_1 \tilde{U} \varphi_2$. It is easy to prove, by an induction on the structure of φ, that each fair module in S_φ satisfies φ and has at most $|\varphi|$ states and $|\varphi|$ transitions. We prove here that if φ is satisfiable, then $S_\varphi \neq \emptyset$. For that, we prove, by an induction on the structure of φ, the following two claims.

(1) For every fair module M with a single initial state that satisfies φ, there exists a fair module $M' \in S_\varphi$ such that M and M' agree on the labeling of their initial states.

(2) If φ is satisfiable, then $S_\varphi \neq \emptyset$.

The proof is easy for φ of the form $\mathbf{true}, \mathbf{false}, p, \neg p, \varphi_1 \vee \varphi_2$ or $\varphi_1 \wedge \varphi_2$. For the other forms, we proceed as follows.

—Let $\varphi = EX\varphi_1$. Since $S_{\mathbf{true}}$ is the set of all one-state fair modules over AP, (1) is immediate. When φ is satisfied, it must be that φ_1 is satisfied as well. Then, by the induction hypothesis, S_{φ_1} is not empty, and hence, so is S_φ.

—Let $\varphi = E\varphi_1 U \varphi_2$. As satisfiability of φ implies the satisfiability of φ_2, (2) follows easily from the induction hypothesis and definition of S_φ to contain S_{φ_2}. In order to prove (1), note that the initial state of any model that satisfies φ satisfies either φ_2 or φ_1. Hence, as S_φ contains not only S_{φ_2} but also $S_{\varphi_1} \rightarrow S_\varphi$, (1) follows easily from the induction hypothesis.

—Let $\varphi = E\varphi_1 \tilde{U} \varphi_2$. As satisfiability of φ implies the satisfiability of φ_2, (2) follows easily from the induction hypothesis. Since the initial state of any model that satisfies φ must satisfy φ_2, (1) also follows easily from the induction hypothesis.

Acknowledgment: We thank Martin Abadi, Orna Grumberg, Pierre Wolper, and Libi for helpful suggestions and discussions. Libi will be fondly remembered. We also thank the anonymous referees for their helpful comments.

REFERENCES

Abadi, M. and Lamport, L. 1993. Composing specifications. *ACM Transactions on Programming Languages and Systems 15,* 1, 73–132.

Aziz, A., Shiple, T., Singhal, V., and Sangiovanni-Vincentelli, A. 1994. Formula-dependent equivalence for compositional CTL model checking. In *Proc. 6th Conf. on Computer Aided Verification.* Lecture Notes in Computer Science, vol. 818. Springer-Verlag, Stanford, CA, 324–337.

Beeri, C. 1980. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Trans. on Database Systems 5,* 241–259.

Burch, J., Clarke, E., McMillan, K., Dill, D., and Hwang, L. 1990. Symbolic model checking: 10^{20} states and beyond. In *Proc. 5th Symposium on Logic in Computer Science.* Philadelphia, 428–439.

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

Clarke, E., Grumberg, O., and Peled, D. 1999. *Model Checking.* MIT Press.

Clarke, E., Long, D., and McMillan, K. 1989. Compositional model checking. In *Proc. 4th IEEE Symposium on Logic in Computer Science*, R. Parikh, Ed. IEEE Computer Society Press, 353–362.

Damm, W., Döhmen, G., Gerstner, V., and Josko, B. 1989. Modular verification of Petri nets: the temporal logic approach. In *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness (Proceedings of REX Workshop).* Lecture Notes in Computer Science, vol. 430. Springer-Verlag, Mook, The Netherlands, 180–207.

Dams, D., Grumberg, O., and Gerth, R. 1993. Generation of reduced models for checking fragments of CTL. In *Proc. 5th Conf. on Computer Aided Verification.* Lecture Notes in Computer Science, vol. 697. Springer-Verlag, 479–490.

Emerson, E. and Halpern, J. 1985. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences 30*, 1–24.

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

Emerson, E. and Jutla, C. 1988. The complexity of tree automata and logics of programs. In *Proc. 29th IEEE Symposium on Foundations of Computer Science.* White Plains, 328–337.

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

Emerson, E. and Lei, C.-L. 1987. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming 8*, 275–306.

Grumberg, O. and Long, D. 1991. Model checking and modular verification. In *Proc. 2nd Conference on Concurrency Theory.* Lecture Notes in Computer Science, vol. 527. Springer-Verlag, 250–265.

Grumberg, O. and Long, D. 1994. Model checking and modular verification. *ACM Trans. on Programming Languages and Systems 16,* 3, 843–871.

Jones, C. 1983. Specification and design of (parallel) programs. In *Information Processing 83: Proc. IFIP 9th World Congress*, R. Mason, Ed. IFIP, North-Holland, 321–332.

Jones, N. 1975. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences 11*, 68–75.

Jonsson, B. and Tsay, Y.-K. 1995. Assumption/guarantee specifications in linear-time temporal logic. In *TAPSOFT '95: Theory and Practice of Software Development*, P. Mosses, M. Nielsen, and M. Schwartzbach, Eds. Lecture Notes in Computer Science, vol. 915. Springer-Verlag, Aarhus, Denmark, 262–276.

Josko, B. 1987a. MCTL – an extension of CTL for modular verification of concurrent systems. In *Temporal Logic in Specification, Proceedings.* Lecture Notes in Computer Science, vol. 398. Springer-Verlag, Altrincham,UK, 165–187.

Josko, B. 1987b. Model checking of CTL formulae under liveness assumptions. In *Proc. 14th Colloq. on Automata, Programming, and Languages (ICALP).* Lecture Notes in Computer Science, vol. 267. Springer-Verlag, 280–289.

Josko, B. 1989. Verifying the correctness of AADL modules using model chekcing. In *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness (Proceedings of REX Workshop).* Lecture Notes in Computer Science, vol. 430. Springer-Verlag, Mook, The Netherlands, 386–400.

Kupferman, O. and Grumberg, O. 1996. Buy one, get one free!!! *Journal of Logic and Computation 6,* 4, 523–539.

Kupferman, O. and Vardi, M. 1995. On the complexity of branching modular model checking. In *Proc. 6th Conference on Concurrency Theory.* Lecture Notes in Computer Science, vol. 962. Springer-Verlag, Philadelphia, 408–422.

KUPFERMAN, O. AND VARDI, M. 1998. Verification of fair transition systems. *Chicago Journal of Theoretical Computer Science 1998,* 2 (March).

KUPFERMAN, O., VARDI, M., AND WOLPER, P. 2000. An automata-theoretic approach to branching-time model checking. *Journal of the ACM 47,* 2 (March).

KURSHAN, R. 1987. Reducibility in analysis of coordination. Lecture Notes in Computer Science, vol. 103. Springer-Verlag, 19–39.

LAMPORT, L. 1980. Sometimes is sometimes "not never" - on the temporal logic of programs. In *Proc. 7th ACM Symposium on Principles of Programming Languages.* 174–185.

LAMPORT, L. 1983. Specifying concurrent program modules. *ACM Trans. on Programming Languages and Systenms 5,* 190–222.

LICHTENSTEIN, O. AND PNUELI, A. 1985. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symposium on Principles of Programming Languages.* New Orleans, 97–107.

LYNCH, N. 1977. Log space recognition and translation of parenthesis languages. *Journal ACM 24,* 583–590.

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

MISRA, B. AND CHANDY, K. 1981. Proofs of networks of processes. *IEEE Trans. on Software Engineering 7,* 417–426.

MULLER, D., SAOUDI, A., AND SCHUPP, P. 1986. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th Int. Colloquium on Automata, Languages and Programming.* Springer-Verlag.

MULLER, D. AND SCHUPP, P. 1987. Alternating automata on infinite trees. *Theoretical Computer Science 54,* 267–276.

PNUELI, A. 1977. The temporal logic of programs. In *Proc. 18th IEEE Symposium on Foundation of Computer Science.* 46–57.

PNUELI, A. 1981. The temporal semantics of concurrent programs. *Theoretical Computer Science 13,* 45–60.

PNUELI, A. 1985a. 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.* Volume 224, LNCS, Springer-Verlag, Berlin, 510–584.

PNUELI, A. 1985b. In transition from global to modular temporal reasoning about programs. In *Logics and Models of Concurrent Systems,* K. Apt, Ed. NATO Advanced Summer Institutes, vol. F-13. Springer-Verlag, 123–144.

QUEILLE, J. AND SIFAKIS, J. 1981. Specification and verification of concurrent systems in Cesar. In *Proc. 5th International Symp. on Programming.* Vol. 137. Springer-Verlag, Lecture Notes in Computer Science, 337–351.

SAFRA, S. 1988. On the complexity of ω-automata. In *Proc. 29th IEEE Symposium on Foundations of Computer Science.* White Plains, 319–327.

SAFRA, S. 1989. Complexity of automata on infinite objects. Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel.

SAVITCH, W. 1970. Relationship between nondeterministic and deterministic tape complexities. *Journal on Computer and System Sciences 4,* 177–192.

SISTLA, A. AND CLARKE, E. 1985. The complexity of propositional linear temporal logic. *Journal ACM 32,* 733–749.

SISTLA, A. AND ZUCK, L. 1993. Reasoning in a restricted temporal logic. *Information and Computation 102,* 2, 167–195.

STARK, E. 1984. Foundations of theory of specifications for distributed systems. Ph.D. thesis, M.I.T.

VARDI, M. 1995. On the complexity of modular model checking. In *Proc. 10th IEEE Symposium on Logic in Computer Science.* 101–111.

VARDI, M. AND STOCKMEYER, L. 1985. Improved upper and lower bounds for modal logics of programs. In *Proc 17th ACM Symp. on Theory of Computing.* 240–251.

Vardi, M. and Wolper, P. 1986. An automata-theoretic approach to automatic program verification. In *Proc. First Symposium on Logic in Computer Science*. Cambridge, 332–344.

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

Received June 1998; revised January 1999; accepted September 1999