Modular Model Checking *

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

[1] EECS Department, UC Berkeley, Berkeley CA 94720-1770, U.S.A.
Email: orna@eecs.berkeley.edu URL: http://www.eecs.berkeley.edu/~orna
[2] Rice University, Department of Computer Science, Houston, TX 77251-1892, U.S.A.
Email: vardi@cs.rice.edu, URL: http://www.cs.rice.edu/~vardi

Abstract. In *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. 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 describe a reduction of modular model checking to standard model checking. Using the reduction, we show that modular model checking is PSPACE-complete for $\forall$CTL and is EXPSPACE-complete for $\forall$CTL*. We then show that the case of LTL assumption is a special case of the case of $\forall$CTL* assumption, but that the EXPSPACE-hardness result apply already to assumptions in LTL.

1 Introduction

Temporal logics, which are modal logics geared towards the description of the temporal ordering of events, have been adopted as a powerful tool for specifying and verifying concurrent programs [Pnu77, Pnu81]. One of the most significant developments in this area is the discovery of algorithmic methods for verifying temporal-logic properties of *finite-state* programs [CES86, LP85, QS81]. This

* This paper is based on "On the complexity of modular model checking", by M.Y. Vardi, *Proc. 10th IEEE Symp. on Logic in Computer Science* (LICS'95), 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), August 1995, Springer-Verlag, Lecture Notes in Computer Science 962, pp. 408–422.

** Supported in part by ONR YIP award N00014-95-1-0520, by NSF CAREER award CCR-9501708, by NSF grant CCR-9504469, by AFOSR contract F49620-93-1-0056, by ARO MURI grant DAAH-04-96-1-0341, by ARPA grant NAG2-892, and by SRC contract 95-DC-324.036.

*** Supported in part by NSF grants CCR-9628400 and CCR-9700061, and by a grant from the Intel Corporation.

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. Surveys can be found in [CG87, Wol89, CGL93].

We distinguish between two types of temporal logics: linear and branching [Lam80]. In linear temporal logics, each moment in time has a unique possible future, while in branching temporal logics, each moment in time may split into several possible futures. 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 $O(nm)$ [CES86], while, for linear temporal logic such as LTL, model-checking algorithms run in time $n2^{O(m)}$ [LP85]. Since model checking with respect to a linear temporal logic formula is PSPACE-complete [SC85], 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 is 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 one in the area of computer-aided verification and is the subject of active research (cf. [BCM+90]).

Modular verification is one possible way to address the state-explosion problem, cf. [CLM89, ASSS94]. 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 [Pnu85b].

A key observation, see [Lam83, Jon83, MC81], 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 [Pnu85b], 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 [JT95]). 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 [CES86, EL85b, EL87] and made explicit in [Jos87a, Jos87b, Jos89, DDGJ89], 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 [GL94], where assumptions are taken to apply to the computation tree of the system within which the module is interacting. Accordingly, assumptions in [GL94] are also expressed in branching temporal logic. There, a module M satisfies an assume-guarantee pair $\langle \varphi, \psi \rangle$ iff 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 [DDGJ89, Jos89, GL91, DGG93], 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 [GL94] 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*. Our key result is that modular model checking can be reduced to standard model checking. At the same time, we show that there is a significant penalty in computational complexity. The fundamental technique used here is the *maximal-model* technique introduced in [GL94]. It is shown there that with every ∀CTL formula φ one can associate a *maximal model* M_φ (called the *tableau* of φ in [GL94]) 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* [VS85, EJ88] 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 maximal-model technique yield optimal algorithms for modular model checking. We prove that the problem is PSPACE-complete for ∀CTL and is EXPSPACE-complete for ∀CTL*. We then show that the linear-branching model-checking problem is a special case of the branching modular model-checking problem, but that the EXPSPACE-hardness result apply already to assumptions in linear temporal logic. We also show that the increase in complexity is solely 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.

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* $\pi = \sigma_0, \sigma_1, \ldots$, 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 j's 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 $\pi \models p$ iff $p \in \sigma_0$ and $\pi \models \neg p$ iff $p \notin \sigma_0$.

- $\pi \models \psi \vee \varphi$ iff $\pi \models \psi$ or $\pi \models \varphi$.
- $\pi \models \psi \wedge \varphi$ iff $\pi \models \psi$ and $\pi \models \varphi$.
- $\pi \models X\psi$ iff $\pi^1 \models \psi$.
- $\pi \models \psi U \varphi$ iff there exists $k \geq 0$ such that $\pi^k \models \varphi$ and $\pi^i \models \psi$ for all $0 \leq i < k$.
- $\pi \models \psi \tilde{U} \varphi$ iff 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 size of a formula φ by $|\varphi|$ and we use the following abbreviations in writing formulas:

- $\rightarrow$ and $\leftrightarrow$, interpreted in the usual way.
- $F\psi = \mathbf{true} U \psi$ ("eventually").
- $G\psi = \neg F \neg \psi$ ("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.
- $E\psi$ or $A\psi$, where ψ is a CTL* path formula.

A CTL* path formula is either:

- A CTL* state formula.
- $\psi \vee \varphi$, $\psi \wedge \varphi$, $X\psi$, $\psi U \varphi$, 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 an *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 total transition relation $R \subseteq W \times W$, a set $W_0 \subseteq W$ of initial states, a labeling function $L : W \to 2^{AP}$, and a Rabin fairness condition α; that is, α defines a subset of W^ω (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; that is, the number of different R-successors that w has. A *computation* of a fair module is a 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. That is,

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

A computation of M is *fair* iff it satisfies the fairness condition α. Thus, if α is $\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$, then π is fair iff 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, iff π visits G_i infinitely often and visits B_i only finitely often. We say that a fair module is *nonempty* iff 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 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 $w \models p$ iff $p \in L(w)$ and $w \models \neg p$ iff $p \notin L(w)$.
- $w \models \psi \vee \varphi$ iff $w \models \psi$ or $w \models \varphi$.
- $w \models \psi \wedge \varphi$ iff $w \models \psi$ and $w \models \varphi$.
- $w \models E\psi$ iff there exists a fair computation $\pi = w_0, w_1, \ldots$ such that $w_0 = w$ and $\pi \models \psi$.
- $w \models A\psi$ iff 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 iff $w_0 \models \varphi$.

A fair module M satisfies a formula φ, denoted $M \models \varphi$, iff φ holds in *all* initial states of M. The problem of determining whether a given fair module M satisfies a formula φ is the *fair-model-checking* problem. The complexity of fair model checking is very well understood.

Theorem 1.

(1) [SC85, VW86] *The fair-model-checking problem for specification in LTL is PSPACE-complete. Determining whether $M \models \varphi$ for φ in LTL can be done in time $k2^{O(l)}$ and space $O((\log k + l)^2)$, where k is the size of M, and l is the length of φ.*

(2) [CES86, KV95] *The fair-model-checking problem for specification in CTL is PTIME-complete. Determining whether $M \models \varphi$ for φ in CTL can be done in time $O(kl)$ and space $O(l \log^2 k)$, where k is the size of M, and l is the length of φ.*

(3) [EL85a, KV95] *The fair-model-checking problem for specification in CTL* is PSPACE-complete. Determining whether $M \models \varphi$ for φ in CTL* can be done in time $k2^{O(l)}$ and space $O(l(\log k + l)^2)$, where k is the size of M, and l is the length of φ.*

Since modular model-checking with assumption φ and guarantee ψ in LTL reduce to model checking the formula $\varphi \rightarrow \psi$ [Pnu85a], 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.

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 [GL94]. 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 $\langle M', w' \rangle$ iff the following conditions hold:

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

(2) For all s and s', we have that $H(s, s')$ implies the following:

- **(2.1)** $L(s) \cap AP' = L(s')$.
- **(2.2)** 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'* iff for every $w \in W_0$ there exists $w' \in W'_0$ such that $H(w, w')$. 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 [Mil71], where there are no fairness conditions. We sometimes relate module (with no fairness condition) with $\leq$. Then, we assume that all computations are fair, and the relation that follows coincides with the one in [Mil71].

Theorem 2. [GL94]

(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, if $M = \langle AP, W, R, W_0, L \rangle$ and $M' = \langle AP', W', R', W'_0, L' \rangle$, then $M \| M' = \langle AP'', W'', R'', W''_0, L'' \rangle$, where,

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

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 (that we assume to be disjoint), and M has m pairs in its fairness condition, then $M \| M'$ has nn' states and m pairs.

The following properties of compositions are proven in [GL94] for fair Streett modules (modules where the fairness condition is Streett), and we prove them here for modules and fair Rabin modules.

Theorem 3. *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) *$M' \leq M' \| M'$.*

Proof: The proof is very similar to the proof for Streett modules given in [GL94]. We start with (1). Assume that $M' \leq M''$. Let H be a simulation from M' to M''. Let W be the states space of M. It is easy to see that the relation $H' = \{\langle \langle w, w' \rangle, \langle w, w'' \rangle \rangle : H(w', w'')\}$ 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 $M' \| M'$. □

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 $M \leq M_\varphi$ iff $M \models \varphi$.

Theorem 4. [GL94] *For every $\forall$CTL formula φ, there exists a maximal model M_φ of size $2^{O(|\varphi|)}$.*

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,

$$inf(r) = \{q \in Q \ : \text{ for infinitely many } i \geq 0, \text{ we have } r(i) = q\}.$$

As Q is finite, it is guaranteed that $Inf(r) \neq \emptyset$. The run r *accepts* w iff $Inf(r) \cap F \neq \emptyset$. 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 iff 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 5. [VW94] *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 Branching Modular Model-Checking for ∀CTL

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:

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

Thus, a key step in modular verification is checking that assume-guarantee assertions hold, which we call the *branching modular model-checking problem*. It was shown in [GL94] that the maximal-model technique yields a solution to the modular model-checking problem.

Theorem 6. [GL94] *For all ∀CTL formulas φ and ψ, and for every fair module M, we have that $\langle\varphi\rangle M\langle\psi\rangle$ iff $M\|M_\varphi \models \psi$.*

Thus, modular model checking for ∀CTL is reducible to standard model checking for ∀CTL. Combining this with Theorems 4 and 1, we get the following complexity results.

Theorem 7.

(1) *Determining whether $\langle\varphi\rangle M\langle\psi\rangle$, for φ and ψ in ∀CTL, can be done in time $km2^{O(l)}$ and space $O(m(\log k+l)^2)$, where k is the size of M, l is the length of φ, and m is the length of ψ.*

(2) *Determining whether $\langle\varphi\rangle M\langle\psi\rangle$, for φ in ∀CTL and ψ in ∀CTL*, can be done in time $k2^{O(l+m)}$ and space $O(m(\log k+l+m)^2)$, where k is the size of M, l is the length of φ, and m is the length of ψ.*

A comparison of Theorem 7 with Theorem 1 shows that the complexity of branching modular model checking with assumptions in ∀CTL is higher than the complexity of CTL model checking, but is comparable to the complexity of LTL model checking. How do LTL and ∀CTL compare from the expressiveness point of view? While ∀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 ∀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 (Theorem 1), 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[EH86]. Thus, $\forall$CTL is simply not expressive enough as a specification language for modular model checking. In the next section we will consider using $\forall$CTL* and LTL as specification languages for assumptions in modular model checking.

3 Modular Model-Checking for $\forall$CTL* and LTL

3.1 Maximal Models

We now consider assumptions in $\forall$CTL*, and we wish to construct maximal models for such assumptions. Unfortunately, the tableau-based technique that was used in the proof of Theorem 4 does not seem to extend to $\forall$CTL*. Indeed, while the satisfiability problem for CTL can be solved using tableau-based technique [EH85], the satisfiability problem for CTL* requires sophisticated automata-theoretic techniques [EJ88]. We now show that these automata-theoretic techniques can be used to construct maximal models for $\forall$CTL* formulas.

Theorem 8. *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 $\pi = w_0, w_1, \ldots$ iff 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 5, 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. The resulted automaton, 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 [Saf89], 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* iff 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 $\langle w, q \rangle$; that it, $\langle w_1, q_1 \rangle = \langle w, q \rangle$. 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 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. □

We can now obtain an alternative proof of Theorem 4.

Theorem 9. *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 $2^{O(|\varphi|)}$. □

3.2 The Branching Modular Model-Checking Problem

We now show the maximal-model technique enables us to reduce modular model checking to standard model checking.

Theorem 10. *For all $\forall$CTL* formulas φ and ψ, and for every fair module M, we have that $\langle\varphi\rangle M\langle\psi\rangle$ iff $M\|M_\varphi \models \psi$.*

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. Since $M_\varphi \models \varphi$ and $M\|M_\varphi \leq M_\varphi$, we have, by Theorem 2 (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 3 (1), that $M\|M\|M' \leq M\|M_\varphi$. Thus, by Theorem 3 (2), $M\|M' \leq M\|M_\varphi$ and therefore, by Theorem 2 (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 11 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 11.

(1) *The branching modular model-checking problem for $\forall$CTL is PSPACE-complete.*
(2) *The branching modular model-checking problem for $\forall$CTL* is EXPSPACE-complete.*
(3) *Determining whether $\langle\varphi\rangle M\langle\psi\rangle$, for φ and ψ in $\forall$CTL*, can be done in time $k2^{O(m)+2^{O(l)}}$ 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.*

(4) *Determining whether $\langle\varphi\rangle M\langle\psi\rangle$, for φ in $\forall CTL^\star$ and ψ in $\forall CTL$, can be done in time $km2^{2^{O(l)}}$ in space $O(m(\log k + 2^{O(l)})^2)$, 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 10, the problem of determining whether $\langle\varphi\rangle M\langle\psi\rangle$ is reducible to model checking of ψ in $M\|M_\varphi$. The upper bounds bounds then follow from Theorems 1 and 8.

We now turn to the lower bounds. For both bounds, we do a reduction from the implication problem. 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? For a set AP of atomic propositions, let M_{AP} be the maximal 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 $L(w) = w$. Let φ and ψ be $\forall$CTL* formulas over a set AP of atomic propositions. 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 $M\|M_{AP} \models \varphi$ iff $M \models \varphi$. Thus, the implication $\varphi \rightarrow \psi$ holds iff $\langle\varphi\rangle M_{AP}\langle\psi\rangle$. The complexity of the reduction depends on the size of M_{AP}. We will show that for both $\forall$CTL and $\forall$CTL*, the size of M_{AP} is fixed.

To prove the PSPACE lower bound for the implication problem for $\forall$CTL, we prove a PSPACE lower bound for its satisfiability problem. The result then follows since the formula φ is satisfiable if and only if φ does not imply A**false**. We prove hardness in PSPACE for $\forall$CTL satisfiability by a reduction from LTL satisfiability, proved to be PSPACE-hard in [SC85]. 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 iff 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. The satisfiability problem for LTL is PSPACE-hard already for formulas with a fixed number of atomic propositions. The PSPACE-hardness proof in [SC85] 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} in our reduction is fixed.

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 iff φ does not imply ψ. In fact we prove a stronger lower bound. Given T and x, we construct an LTL formula ξ and an $\exists$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 iff the formula $A\xi \wedge \theta$ is satisfiable. Thus, taking $\varphi = A\xi$ and $\psi = \neg\theta$, we have that T accepts x iff the implication $\varphi \rightarrow \psi$ does not hold. Thus, the branching modular model-checking problem is EXPSPACE-complete even for an assumption of the form $A\xi$, where ξ is an LTL formula, a fixed fair module, and an $\forall$CTL guarantee. For details see [KV95]. □

Note that the crucial factor in the complexity of the branching modular model checking problem is the assumption part of the specification. Indeed, the lower bounds given in Theorem 11 remains 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 [Jos87b, Jos89, GL94], see also [AL93]. We will come back to this point in Section 4.

3.3 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.

The idea is to use assume-guarantee assertions in modular proof rules of the following form [Jos87a, Jos87b, Jos89]:

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

where $br(\varphi)$ is a branching version (it is an $\forall$CTL formula) of the LTL formula φ; see above references for details. Verifying assertions of the form $[\varphi]M\langle\psi\rangle$ 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 iff $L(\pi) \in P$. 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

- $R^t(\zeta, \zeta')$ iff there exists $w_{k+1} \in W$ such that $\zeta' = w_0, \ldots, w_k, w_{k+1}$ and $R(w_k, w_{k+1})$. 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* iff 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 iff $\{w_0\} \models \psi$. We say that $\langle M, P\rangle \models \psi$ iff $\langle M^t, P\rangle \models \psi$. Now, $[\varphi]M\langle\psi\rangle$ holds iff $\langle M, P_\varphi\rangle \models \psi$, where P_φ is the set of all that computations that satisfy φ.[3].

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 12. *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 $\langle M, P\rangle \models \psi$ iff $M' \models \psi$. 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

- $R'(\langle w, q\rangle, \langle w', q'\rangle)$ iff $R(w, w')$ and $\delta(q, L(w)) = q'$.

[3] We note that the formal definitions of $[\varphi]M\langle\psi\rangle$ in [Jos87a, Jos87b, Jos89] apply only to restricted linear temporal assumptions and involve a complicated syntactic construction.

- $L'(\langle w, q\rangle) = L(w)$.
- $\alpha = \{\langle W \times G, W \times B\rangle : \langle G, B\rangle \in F\}$.

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 $\langle M^t, P\rangle$ iff $\langle \mathit{last}(\zeta), \delta(q_0, \sigma(\zeta))\rangle \models \psi$ in M'. In particular, $\{w_0\} \models \psi$ in $\langle M^t, P\rangle$ iff $\langle w_0, q_0\rangle \models \psi$ 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 $M \| M$.

□

To solve the the linear-branching model-checking problem, we show that the branching modular framework is more general than the linear-branching modular framework. Thus, the algorithms discussed in Section 3.2 are applicable also here.

Theorem 13. *For every LTL formula φ, fair module M, and a $\forall$CTL* formula ψ, we have that $\langle A\varphi\rangle M\langle\psi\rangle$ iff $[\varphi]M\langle\psi\rangle$.*

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 12. 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 (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 12, $M' \leq M'\|M$, 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 ∀CTL formula $AFAGp$ is not equivalent to any formula of the form $A\xi$, where ξ 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 11, that the EXPSPACE lower bound holds already for assumptions of the form $A\xi$ for an LTL formula ξ).

4 Concluding Remarks

The results of the paper indicate that modular model-checking for general universal or linear assumptions is rather intractable. Our results provide an *a posteriori* justification for Josko's restriction on the linear temporal assumption [Jos87a, Jos87b, Jos89]. Essentially, for a restricted linear temporal assumption φ, one can get a more economical automata-theoretic construction of the maximal model associated with the CTL* formula $A\varphi$ (exponential rather than doubly exponential). We note that it is argued in [LP85] 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 [CLM89] 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 environment 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.

Acknowledgment: We thank Martin Abadi, Orna Grumberg, and Pierre Wolper for helpful suggestions and discussions.

References

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

[ASSS94] A. Aziz, T.R. Shiple, V. Singhal, and A.L. Sangiovanni-Vincentelli. Formula-dependent equivalence for compositional CTL model checking. In *Proc. 6th Conf. on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 324–337, Stanford, CA, June 1994. Springer-Verlag.

[BCM+90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc. 5th Symposium on Logic in Computer Science*, pages 428–439, Philadelphia, June 1990.

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

[CG87] E.M. Clarke and O. Grumberg. Research on automatic verification of finite-state concurrent systems. In *Annual Review of Computer Science*, volume 2, pages 269–290, 1987.

[CGL93] E.M. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Decade of Concurrency – Reflections and Perspectives (Proceedings of REX School)*, volume 803 of *Lecture Notes in Computer Science*, pages 124–175. Springer-Verlag, 1993.

[CLM89] E.M. Clarke, D.E. Long, and K.L. McMillan. Compositional model checking. In R. Parikh, editor, *Proc. 4th IEEE Symposium on Logic in Computer Science*, pages 353–362. IEEE Computer Society Press, 1989.

[DDGJ89] W. Damm, G. Döhmen, V. Gerstner, and B. Josko. Modular verification of Petri nets: the temporal logic approach. In *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness (Proceedings of REX Workshop)*, volume 430 of *Lecture Notes in Computer Science*, pages 180–207, Mook, The Netherlands, May/June 1989. Springer-Verlag.

[DGG93] D. Dams, O. Grumberg, and R. Gerth. Generation of reduced models for checking fragments of CTL. In *Proc. 5th Conf. on Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 479–490. Springer-Verlag, June 1993.

[EH85] E.A. Emerson and J.Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30:1–24, 1985.

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

[EJ88] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th IEEE Symposium on Foundations of Computer Science*, pages 368–377, White Plains, October 1988.

[EL85a] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. In *Proc. 20th ACM Symposium on Principles of Programming Languages*, pages 84–96, New Orleans, January 1985.

[EL85b] E.A. Emerson and C.-L. Lei. Temporal model checking under generalized fairness constraints. In *Proc. 18th Hawaii International Conference on System Sciences*, North Holywood, 1985. Western Periodicals Company.

[EL87] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8:275–306, 1987.

[GL91] O. Grumberg and D.E. Long. Model checking and modular verification. In *Proc. 2nd Conferance on Concurrency Theory*, volume 527 of *Lecture Notes in Computer Science*, pages 250–265. Springer-Verlag, 1991.

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

[Jon83] C.B. Jones. Specification and design of (parallel) programs. In R.E.A. Mason, editor, *Information Processing 83: Proc. IFIP 9th World Congress*, pages 321–332. IFIP, North-Holland, 1983.

[Jos87a] B. Josko. MCTL – an extension of CTL for modular verification of concurrent systems. In *Temporal Logic in Specification, Proceedings*, volume 398 of *Lecture Notes in Computer Science*, pages 165–187, Altrincham,UK, April 1987. Springer-Verlag.

[Jos87b] B. Josko. Model checking of CTL formulae under liveness assumptions. In *Proc. 14th Colloq. on Automata, Programming, and Languages (ICALP)*, volume 267 of *Lecture Notes in Computer Science*, pages 280–289. Springer-Verlag, July 1987.

[Jos89] B. Josko. Verifying the correctness of AADL modules using model chekcing. In *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness (Proceedings of REX Workshop)*, volume 430 of *Lecture Notes in Computer Science*, pages 386–400, Mook, The Netherlands, May/June 1989. Springer-Verlag.

[JT95] B. Jonsson and Y.-K. Tsay. Assumption/guarantee specifications in linear-time temporal logic. In P.D. Mosses, M. Nielsen, and M.I. Schwartzbach, editors, *TAPSOFT '95: Theory and Practice of Software Development*, volume 915 of *Lecture Notes in Computer Science*, pages 262–276, Aarhus, Denmark, May 1995. Springer-Verlag.

[KV95] O. Kupferman and M.Y. Vardi. On the complexity of branching modular model checking. In *Proc. 6th Conferance on Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 408–422, Philadelphia, August 1995. Springer-Verlag.

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

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

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symposium on Principles of Programming Languages*, pages 97–107, New Orleans, January 1985.

[MC81] B. Misra and K.M. Chandy. Proofs of networks of processes. *IEEE Trans. on Software Engineering*, 7:417–426, 1981.

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

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

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

[Pnu85a] A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In *Proc. Advanced School on Current Trends in Concurrency*, pages 510–584, Berlin, 1985. Volume 224, LNCS, Springer-Verlag.

[Pnu85b] A. Pnueli. In transition from global to modular temporal reasoning about programs. In K. Apt, editor, *Logics and Models of Concurrent Systems*, volume F-13 of *NATO Advanced Summer Institutes*, pages 123–144. Springer-Verlag, 1985.

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

[Saf89] S. Safra. *Complexity of automata on infinite objects.* PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1989.

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

[VS85] M.Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc 17th ACM Symp. on Theory of Computing*, pages 240–251, 1985.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, June 1986.

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

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

This article was processed using the LaTeX macro package with LLNCS style