An Automata-Theoretic Approach to Modular Model Checking

Orna Kupferman
UC Berkeley*

Moshe Y. Vardi
Rice University†

May 8, 1997

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 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 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.

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

*Address: EECS Department, Berkeley CA 94720-1770, U.S.A. Email: orna@eecs.berkeley.edu

†Address: Department of Computer Science, Houston, TX 77251-1892, U.S.A. Email: vardi@cs.rice.edu

[CES86, LP85, QS81]. 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. Surveys can be found in [CG87, Wol89].

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 method-

ology 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 [Pnu85].

A key observation, see [Lam83, Jon83], 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 [Pnu85], in this case the assume-guarantee pair $\langle \varphi, \psi \rangle$ can be combined to a single linear temporal logic formula $\varphi \rightarrow \psi$. 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, EL85, 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*. 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 inter-reducible as they are for CTL and CTL*).

We use two fundamental techniques to solve these questions. The first technique 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 second technique is the automata-theoretic framework to branching-time model checking introduced in [BVW94]. 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 [BVW94] 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 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 model-checking in its full generality, i.e., we allow the assumption to be specified by an arbitrary LTL formula and the guarantee to be specified by an arbitrary CTL* formula.

We bring to bear on the problem the automata-theoretic techniques that were developed for linear and branching temporal logics [VW86a, BVW94]. The semantics of the assume-guarantee specification is defined in terms of the computation tree of the module. Using the

automata-theoretic framework for linear temporal logics [VW86b], we show how the computation tree, which is infinite, can be collapsed to a finite module, while retaining the relevant information from the computation tree. This amounts to annotating the states of the module with information about the linear temporal assumption. We then apply to the annotated module a combination of CTL* and LTL model-checking algorithms. This algorithm improves the general algorithm for guarantees in universal logics not only by allowing the assumption to be a general CTL* formula, but also by avoiding the construction of maximal models.

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 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** 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** 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.

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* (*modules*, for short). A 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 module is a sequence of states, $\pi = w_0, w_1, \ldots$ such that for every $i \geq 0$, we have that $\langle w_i, w_{i+1} \rangle \in R$. 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 $\alpha = \{\langle G_1, B_1 \rangle, \ldots, \langle G_k, B_k \rangle\}$, then π is fair iff there exists $1 \leq i \leq k$ such that $inf(w) \cap G_i \neq \emptyset$ and $inf(w) \cap B_i = \emptyset$. In other words, iff π visits G_i infinitely often and visits B_i only finitely often. We say that a module is *nonempty* iff there exists a fair computation that starts at an initial state. A *transition system* is a module with no fairness condition. That is, all the computations of a transition system are considered fair. We denote a transition system by $K = \langle AP, W, R, W_0, L \rangle$ and use M_K to denote the transition system obtained from a module M by making all its computations fair.

We use $w \models \varphi$ to indicate that a state formula φ holds at state w (assuming an agreed 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 $p \notin L(w)$.
- $w \models \psi \vee \psi$ iff $w \models \psi$ or $w \models \varphi$.
- $w \models \psi \wedge \psi$ 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 state formula φ iff $w_0 \models \varphi$.

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

2.2 Simulation Relation and Composition of Modules

In the context of modular verification, it is helpful to define an order relation between modules [GL94]. Intuitively, the order captures what it means for a module M' to have "more behaviors" than a 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 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 module M' has more behaviors than the module M. In fact, every possible behavior of M is also a possible behavior of M'. Note that our simulation is different from the classical simulation used by Milner [Mil71], where there are no fairness conditions.

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' and are fair in both of them. In order to define $M \| M'$, we first define it as a *generalized Rabin fair module* M''. In a generalized Rabin fair module, the fairness condition consists of triples of sets of states. A computation π of a module with $\alpha = \{\langle G^1_1, G^2_1, B_1 \rangle, \langle G^1_2, G^2_2, B_2 \rangle, \ldots, \langle G^1_n, G^2_n, B_n \rangle\}$ is fair if there exists a triple $\langle G^1_i, G^2_i, B_i \rangle \in \alpha$ such that $inf(\pi) \cap G^1_i \neq \emptyset$, $inf(\pi) \cap G^2_i \neq \emptyset$, and $inf(\pi) \cap B_i = \emptyset$. Given M and M', we define $M'' = \langle AP'', W'', R'', W''_0, L'', \alpha'' \rangle$, 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 $L''(\langle w, w' \rangle) = L(w) \cup L'(w')$.
- $\alpha'' = \{\langle (G \times W') \cap W'', (W \times G') \cap W'', ((B \times W') \cup (W \times B')) \cap W'' \rangle :$ $\langle G, B \rangle \in \alpha \text{ and } \langle G', B' \rangle \in \alpha'\}$.

We now have to translate $M\|M'$ to a module; thus, to convert the triples in α'' to pairs. For a pair of pairs, $\langle G, B\rangle \in \alpha$ and $\langle G', B'\rangle \in \alpha'$, let $M''(\langle G, B\rangle, \langle G', B'\rangle)$ be a "specialized module" that has exactly these computations that are joint to M and M', visit G and G' infinitely often, and visit B and B' finitely often. Note that these computations are fair in $M\|M'$ by satisfying the triple $\langle (G \times W') \cap W'', (W \times G') \cap W'', ((B \times W') \cup (W \times B')) \cap W''\rangle$. We can easily define $M''(\langle G, B\rangle, \langle G', B'\rangle)$, as in the product of two Büchi automata, by augmenting each state of W'' with a Boolean flag [VW86b]. Formally, $M''(\langle G, B\rangle, \langle G', B'\rangle) = \langle AP'', W'' \times \{0,1\}, M''', W_0'' \times \{0\}, L''', \alpha'''\rangle$ where,

- $\langle\langle\langle s, s'\rangle, l_1\rangle, \langle\langle t, t'\rangle, l_2\rangle\rangle \in R'''$ iff $\langle\langle s, s'\rangle, \langle t, t'\rangle\rangle \in R''$ and one of the following hold.
  - $l_1 = 0, s \notin G$, and $l_2 = 0$,
  - $l_1 = 0, s \in G$, and $l_2 = 1$,
  - $l_1 = 1, s' \notin G'$, and $l_2 = 1$, or
  - $l_1 = 1, s' \in G$, and $l_2 = 0$.
- $L'''(\langle\langle s, s'\rangle, l\rangle) = L''(\langle s, s'\rangle)$.
- $\alpha''' = ((G \times W') \cap W'') \times \{0\}$.

We can now define the module $M\|M'$ as the disjoint union of the specialized modules of all the possible pairs of pairs in α and α'. Since a module may have several initial states, union is trivial. Thus, if M and M' have n and n' states (that we assume to be disjoint), and have m and m' pairs in their fairness conditions, then $M\|M'$ has $2nn'mm'$ states and mm' pairs.

The following properties of simulation and compositions are proven in [GL94] for fair Streett modules (modules where the fairness condition is Streett), and they hold also for fair Rabin modules.

Theorem 2.1 [GL94]

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

(2) *For every M and M', we have $M\|M' \leq M$.*

(3) *For every M, M', and M'', if $M \leq M'$ then $M\|M'' \leq M'\|M''$.*

(4) *For every M, we have $M \leq M\|M$.*

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

A 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 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 module M, we have that $M \leq M_\varphi$ iff $M \models \varphi$. A construction of maximal models for $\forall$CTL formulas is described in [GL94]. 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 $w \in \Sigma^\omega$, 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 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 module can be viewed as infinite words over the alphabet 2^{AP}. According to this view, each 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 2.2 [VW94] *Given an LTL formula ψ, there exists a Büchi automaton $\mathcal{A}_\psi = \langle 2^{AP}, Q, \delta, Q_0, F \rangle$, of size $2^{O(|\psi|)}$, such that $\mathcal{L}(\mathcal{A}_\psi)$ is exactly the set of computations satisfying ψ.*

2.4 Alternating Tree Automata

An *infinite tree* is a 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 degree of x and is denoted by $d(x)$. We consider here trees in which each node has at least one successor. A *path* ρ of a tree T is a 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 module, we can view a computation tree as a module with infinitely many states. We sometimes refer to satisfaction of temporal logic formulas in a computation tree, meaning their satisfaction in this module. Given a set $\mathcal{D} \subset \mathbb{N}$, a $\mathcal{D}$-tree is a computation tree in which all the nodes have degrees in $\mathcal{D}$.

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 [MS87]. 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 iff 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 conjunction in δ has 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 $\mathcal{A} = \langle \Sigma, \mathcal{D}, Q, \delta, q_0, F \rangle$ as $|Q| + |F| + |\delta|$, where $|Q|$ and $|F|$ are the respective cardinalities of the sets Q and F, and where $|\delta|$ 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 stored 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 $\mathbb{N}^* \times Q$. 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 which case it contains a leaf, or infinite. Each node of T_r corresponds to a node of T. A node in T_r, labeled by (x, q), describes a copy of the automaton that reads the node x of T and visits the state q. Note that many nodes of T_r can correspond to the same node of T; in contrast, in a run of a nondeterministic automaton on $\langle T, V \rangle$ there is a one-to-one correspondence between the nodes of the run and the nodes of the tree. The labels of a node and its successors have to satisfy the transition function. Formally, $\langle T_r, r \rangle$ is a Σ_r-labeled tree where $\Sigma_r = \mathbb{N}^* \times Q$ 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 hold:
   - 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 $\theta = \mathbf{true}$, 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 $\theta = \mathbf{false}$, we can not have a run that takes a transition with $\theta = \mathbf{false}$. A run $\langle T_r, r \rangle$ is

accepting iff all its infinite paths satisfy the acceptance condition. As with nondeterministic automata, an automaton accepts a tree iff 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. Note that an alternating automaton over infinite words is simply an alternating automaton over infinite trees with $\mathcal{D} = \{1\}$.

Example 2.3 We define an alternating Büchi 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 below.

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**	$(0, q_3) \wedge (1, q_3)$	$(0, q_3) \wedge (1, q_3)$

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 [BVW94], we introduced *hesitant alternating automata* (HAA). A HAA 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* [MSS86], 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. 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 hold:

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 either an existential or a 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 number of sets in the partition of Q is defined as the *depth* of $\mathcal{A}$. For a state $q \in Q$, we say that q is an *existential state* iff $q \in Q_i$ for an existential set Q_i. Universal states are defined analogously.

Theorem 2.4 [BVW94]

(1) *Given a CTL formula ψ and $\mathcal{D} \subset I\!N$, 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 $\mathcal{D} \subset I\!N$, 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.*

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 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 $p \in \sigma$.
- $\delta(\varphi_1 \vee \varphi_2,\sigma,k) = \delta(\varphi_1,\sigma,k) \vee \delta(\varphi_2,\sigma,k)$.
- $\delta(\varphi_1 \wedge \varphi_2,\sigma,k) = \delta(\varphi_1,\sigma,k) \wedge \delta(\varphi_2,\sigma,k)$.
- $\delta(EX\varphi,\sigma,k) = \bigvee_{c=0}^{k-1}(c,\varphi)$.
- $\delta(AX\varphi,\sigma,k) = \bigwedge_{c=0}^{k-1}(c,\varphi)$.
- $\delta(E\varphi_1 U\varphi_2,\sigma,k) = \delta(\varphi_2,\sigma,k) \vee (\delta(\varphi_1,\sigma,k) \wedge \bigvee_{c=0}^{k-1}(c, E\varphi_1 U\varphi_2))$.

- $\delta(A\varphi_1 U \varphi_2, \sigma, k) = \delta(\varphi_2, \sigma, k) \vee (\delta(\varphi_1, \sigma, k) \wedge \bigwedge_{c=0}^{k-1}(c, A\varphi_1 U \varphi_2))$.
- $\delta(E\varphi_1 \tilde{U} \varphi_2, \sigma, k) = \delta(\varphi_2, \sigma, k) \wedge (\delta(\varphi_1, \sigma, k) \vee \bigvee_{c=0}^{k-1}(c, E\varphi_1 \tilde{U} \varphi_2))$.
- $\delta(A\varphi_1 \tilde{U} \varphi_2, \sigma, k) = \delta(\varphi_2, \sigma, k) \wedge (\delta(\varphi_1, \sigma, k) \vee \bigwedge_{c=0}^{k-1}(c, A\varphi_1 \tilde{U} \varphi_2))$.

As discussed in [BVW94], 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 iff $K \models \psi$. 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). A LAA is a HAA extended with a Rabin acceptance condition $\beta \subseteq 2^Q \times 2^Q$. For a run of a LAA with $\beta = \{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$, a path π that gets trapped within a set Q_i is accepted by the run iff 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 now show that the space complexity result of [BVW94] for the 1-letter nonemptiness of HAA extends to LAA.

Theorem 3.1 *The 1-letter nonemptiness problem for a LAA of size n and with m sets can be solved in space $O(m \log^2 n)$.*

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 HAAs 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).

(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.

(3.1) 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'.

(3.2) 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 hold:

1. $q' \in G$.
2. $q'' \in G_j$.
3. q' is reachable from q.
4. q'' is $Q_i \setminus B_j$-reachable from q'.
5. q' is $Q_i \setminus B_j$-reachable from q''.

If such q' and q'' are found (possibly $q' = q''$), label q with 'T'.

(3.3) 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.

(4.1) 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'.

(4.2) 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 hold:

1. $q' \in B$.
2. $q'' \in G_j$.
3. q' is reachable from q.
4. q'' is $Q_i \setminus B_j$-reachable from q'.
5. q' is $Q_i \setminus B_j$-reachable from q''.

If such q' and q'' are found, label q with 'F'.

(4.3) 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 is 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 no that $rank(q) = l + 1$ and 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 (3.1) 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 (3.2) are found, it follows that Q_i contains a cycle reachable from q, such that the infinite path obtained by unwinding the cycle satisfies β. Hence, the state q is accepting. Finally, if both Steps (3.1) and (3.2) 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 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 [Lyn77]. Here, we evaluate expressions over Q, using a recursion to trace 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 iff 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 in space $O(\log^2 n)$.

For each two states q' and q'' in Q_i, we have that q'' is immediately reachable from q' iff the following two condition 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 in space $O(\log^2 n)$.

It is known, by [Jon75], that the graph accessibility problem is in NLOGSPACE. Thus, by Savitch's Theorem [Sav70], 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 $O(\log^2 n)$. Hence, as the labeling of $q \in Q_i$ only involves a search for reachable states, we can determine its labeling in space $O(m \log^2 n)$.

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

3.2 Fair Model Checking

When we perform fair mode 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 [KG96], formulas of this syntax have a linear translation to CTL.

Consider a CTL* formula ψ. Let M be a module in which $\top$ is valid. Clearly, M satisfies ψ iff 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 2.4. Let $\langle T_M, V_M^\top \rangle$ be the computation tree obtained

by unwinding M into an infinite tree and adding $\top$ to each of the labels. By Theorem 2.4, $\mathcal{A}_{\mathcal{D},f(\psi)}$ accepts $\langle T_M, V_M^\top \rangle$ iff 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 iff M satisfies ψ, with path quantification ranging over fair paths only.

$\mathcal{A}_{\mathcal{D},f(\psi)} = \langle 2^{AP \cup \{\top\}}, \mathcal{D}, Q_\psi, \delta_\psi, Q^0_\psi, \alpha_\psi \rangle$, 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 $\mathcal{A}_{M,\psi} = \langle \{a\}, W \times Q_\psi, \delta, W_0 \times Q^0_\psi, \alpha, \beta \rangle$, 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 3.2

(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: (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 $|\beta| = |\beta_M| * |Q_\psi|$.

To prove (2), we show that $\mathcal{L}(\mathcal{A}_{M,\psi})$ is nonempty if and only if there exists a *β-fair* accepting run of $\mathcal{A}_{\mathcal{D},f(\psi)}$ on $\langle T_M, V_M^\top \rangle$. That is, a run in which all the paths that get trapped in existential sets satisfy β_M and paths do not satisfy β_M may get trapped in any universal sets. To see that this is sufficient, note that all the existential formulas in $f(\psi)$ are of the form $E(G\top \wedge \xi)$. Therefore, as taking the product with M we assume that $\top$ holds everywhere, once $\mathcal{A}_{\mathcal{D},f(\psi)}$ enters the existential set associated with $E(G\top \wedge \xi)$, the path of $\langle T_M, V_M^\top \rangle$ that is traced for satisfaction of $(G\top \wedge \xi)$ gets trapped in this set, making sure that it is fair. Dually, all the universal formulas in $f(\psi)$ are of the form $A(F\neg\top \vee \xi)$. Therefore, as taking the product with M we assume that $\top$ holds everywhere, once $\mathcal{A}_{\mathcal{D},f(\psi)}$ enters the universal set associated with $A(F\neg\top \vee \xi)$, paths of $\langle T_M, V_M^\top \rangle$ that are not fair and do not satisfy $(G\top \wedge \xi)$ get trapped in this set, and are accepted.

Given a β-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-fair accepting run of $\mathcal{A}_{\mathcal{D},f(\psi)}$ over $\langle T_M, V_M \rangle$. Assume first there exists a β-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-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-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 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 2.4 and 3.1, together with Proposition 3.2, imply the theorem below.

Theorem 3.3

(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 module.*

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

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. For a specification language that is closed under negation, these problems are inter-reducible. 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* module M such that φ holds in M? As a 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 modules? (It is easy to see that φ holds in all modules iff φ holds in all nonempty modules.) The implication problem is defined as follows: given two formulas φ and ψ, does φ imply ψ? Namely, does ψ hold in every module in which φ holds? In the lemma below, we assert that the satisfiability and the validity problems are special cases of the implication problem.

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 $\varphi \rightarrow \psi$ is valid iff $\varphi \wedge \neg\psi$ 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 iff ψ holds in all 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 [GL94] 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 iff $M_\varphi \models \psi$.*

To complete the reduction of implication to fair model checking, we have to describe the construction of maximal models.

Theorem 4.3 *For every $\forall CTL^\star$ 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. That is, $\mathcal{A}_{\forall(\varphi)}$ nondeterministically guesses a location i and a formula $A\xi$ and then proceeds as the Büchi ω-automaton of $\neg\xi$. By [VW94], 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 branching subformulas of ξ, this causes no difficulty, as we can regard the branching 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

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

where the initial set W_0 includes all states $w \in c(\varphi)$ for which $\varphi \in w$, 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 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 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 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 $A\xi' \in w$, and in particular for $A\xi$, the computation $w_1, w_2, \ldots$ satisfies ξ.

Consider now a 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 of state formulas in $c(\varphi)$ that are true in w. The simulation H is defined as follows:

- 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 $M \models \varphi$, 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, 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 $H(w, \langle f(w), q\rangle)$ and for all $i \geq 1$ we have $H(w_i, \langle f(w_i), q_i\rangle)$. So, it only 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 $q_i, q_{i+1}, \ldots$. Thus, $q_i, q_{i+1}, \ldots$ is accepted by $\overline{\mathcal{A}_{\forall(\varphi)}}$ with q_i as an initial state and therefore, by the definition of β', the computation π' is fair. □

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

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

A different construction for maximal models of ∀CTL formulas is given in [GL94].

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.

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 validity problem of $\varphi \rightarrow \psi$ is reducible to model checking of ψ in M_φ. Theorems 4.3 and 4.4 provide us with a double exponential bound for maximal models of CTL* and an exponential bound for maximal models of CTL. Together with Theorem 3.3, this yields 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 $O(m \log^2(m2^{O(n)})) = O(m(n^2 + \log^2 m))$. 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 $O(m(m + 2^{O(n)})^2) = O(m^2(m + 2^{O(n)}))$.

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 [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 module satisfying ξ_A. For the second direction, assume that ξ_A is satisfiable in some module M. Consider a computation π of M. We can view π as a 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 iff 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 $\exists$CTL formula θ such that 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 $\varphi \rightarrow \psi$ is not valid. We call the LTL formula ξ the *assumption* and the $\exists$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 $|x| = n$. 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 c. We use the node u_n to encode the content of cell c. 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 c. 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 c is labeled is encoded at the node u_n of the block $u_1, \ldots, u_n$ that corresponds to c. Why then do we need a block of length n to encode a single letter? The block also encodes the location of the cell c 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 saves us 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 c. 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.
   - $G(((b \vee d) \wedge \neg c) \rightarrow (X(\neg d) \wedge X^n c))$.
   - $G((\neg(b \vee d) \wedge \neg c) \rightarrow (X(\neg d) \wedge X^n \neg c))$.
   - $G(((b \vee d) \wedge c) \rightarrow (X d \wedge X^n \neg c))$.

- $G((\neg(b \vee d) \wedge c) \rightarrow (Xd \wedge X^n c))$.

An atomic proposition l marks the last node of a configuration, that is, l holds in a node u_n of a block $u_1, \ldots, u_n$ iff 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.

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

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_{n+1} to be σ'_1 and σ_{1-1} 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. [1],

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

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

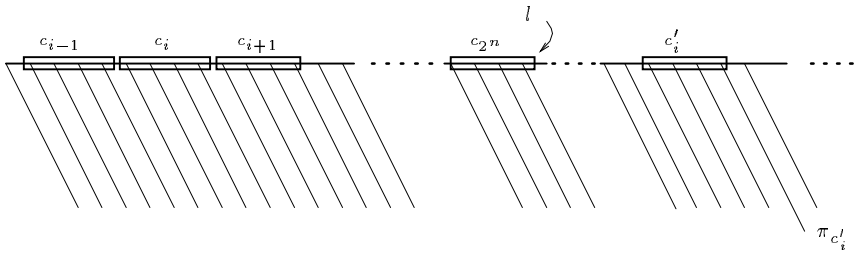
requires the computation to reach an accepting configuration.

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

[1] 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 $i = s(n)$ and $(q, \gamma_i) \rightarrow (q', \gamma'_i, R)$ are not possible.

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 [VS85], is by marking one of the two positions with a "pebble", as explained below.

Consider the bold computation in the figure below. Each of its nodes branches into a pale suffix. Let us call the bold computation a *real* computation and call all the computations that branch to a pale 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 pale 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 c_{i-1}, c_i, c_{i+1} along the real computation, and the block c'_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 c'_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 c_{i-1}, c_i, and c_{i+1}, respectively. Let $\pi_{c'_i}$ be the auxiliary computation that leaves the real computation at the last position of the block c'_i. We use this auxiliary computation in order to check that σ'_i indeed equals $next(\sigma_{i-1}, \sigma_i, \sigma_{i+1})$. This is how we use $\pi_{c'_i}$. 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 block where the computation becomes auxiliary is $next(\sigma_{i-1}, \sigma_i, \sigma_{i+1})$.

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 following conjuncts.

(1) $r \wedge FG\neg r$.

(2) b.

(3)

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

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

(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 iff T accepts x. □

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 3.3 (with n and m denoting the lengths of φ and ψ, respectively) is $O(m(m+\log 2^{O(n)})^2)) = O(m^3+mn^2)$. 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 $\forall CTL^\star$ are PSPACE-complete.*

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

(3) *The validity problem for $\forall CTL^\star$ 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 iff the LTL formula obtained from φ by eliminating its path quantifiers is satisfiable.

Since $\forall$CTL subsumes propositional logic, hardness in NP for the validity problem of $\forall$CTL is immediate. To prove membership in 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 module with at most $|\varphi|$ states and $|\varphi|$ transitions. 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 modules that satisfy φ. The definition of S_φ proceeds by induction on the structure of φ. Each module in S_φ has a single initial state. We use $S_{\varphi_1} \bowtie S_{\varphi_2}$ to denote the set of all modules obtained by taking a module M_1 from S_{φ_1} and a module M_2 from S_{φ_2}, such that M_1 and M_2 agree on the labeling of their initial states, and unifying their initial states to a single initial state. We use $S_{\varphi_1} \rightarrow S_{\varphi_2}$ to denote the set of all modules obtained by taking a module M_1 from S_{φ_1}, a 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 modules obtained by taking a module from S_{φ_2} and adding a self loop to its initial state.

- $S_{\mathbf{true}}$ is the set of all one-state modules over AP.
- $S_{\mathbf{false}} = \emptyset$.
- S_p for $p \in AP$ is the set of all one-state modules over AP in which p holds.
- $S_{\neg p}$ for $\neg p \in AP$ is the set of all one-state modules over AP in which p does not hold.
- $S_{\varphi_1 \vee \varphi_2} = S_{\varphi_1} \cup S_{\varphi_2}$.
- $S_{\varphi_1 \wedge \varphi_2} = S_{\varphi_1} \bowtie S_{\varphi_2}$.
- $S_{EX\varphi_1} = S_{true} \rightarrow S_{\varphi_1}$.
- $S_{E\varphi_1 U \varphi_2} = S_{\varphi_2} \cup (S_{\varphi_1} \rightarrow S_{\varphi_2})$.

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

Typically, the 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 module that satisfy $E\varphi_1 U\varphi_2$ must satisfy either φ_1 or φ_2, the set of initial states of modules in $S_{E\varphi_1 U\varphi_2}$ is equal to the set of initial states of modules in S_{φ_1} or in S_{φ_2}. In addition, 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 (rather than the less-economical $E\varphi_1 U\varphi_2$) in a successor state. Following the same idea, since the initial state of each module that satisfies $E\varphi_1\tilde{U}\varphi_2$ must satisfy φ_2, the set of initial states of modules in $S_{E\varphi_1\tilde{U}\varphi_2}$ is equal to the set of initial states of modules in S_{φ_2}. In addition, all the 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 module in S_φ 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 module M with a single initial state that satisfies φ, there exists a 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 $\mathbf{true}, \mathbf{false}, p, \neg p$, or $\varphi_1 \vee \varphi_2$. For the other forms, we proceed as follows.

- Let $\varphi = \varphi_1 \wedge \varphi_2$. By the induction hypothesis, for every module M with a single initial state that satisfies φ_1, there exists a module $M' \in S_{\varphi_1}$ such that M and M' agree on the labeling of their initial states, and similarly for φ_2. When φ is satisfiable, the restrictions that φ_1 and φ_2 impose on the initial state do not contradict. Thus, there are modules $M_1 \in S_{\varphi_1}$ and $M_2 \in S_{\varphi_2}$ such that M_1 and M_2 agree on the labeling of their initial states. Hence, $S_{\varphi_1} \bowtie S_{\varphi_2}$ is not empty, and (2) holds. Also, as $S_{\varphi_1} \bowtie S_{\varphi_2}$ contains all the possible combinations of modules from S_{φ_1} and S_{φ_2} that agree on the labeling of their initial states, (1) holds as well.

- Let $\varphi = EX\varphi_1$. Since $S_{\mathbf{true}}$ is the set of all one-state modules over AP, (1) is immediate. Also, as satisfaction of φ implies the satisfaction of φ_1, (2) follows easily from the induction hypothesis.

- Let $\varphi = E\varphi_1 U\varphi_2$. As satisfaction of φ implies the satisfaction 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 satisfaction of φ implies the satisfaction 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.

It is left to prove PSPACE completeness for the validity problem of $\forall$CTL*. Since a $\forall$CTL* formula ψ is valid iff 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, 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 iff the $\forall$CTL* formula $A\xi$ is valid. $\square$

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 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 called the *branching modular model-checking problem*. We now show that the techniques developed in the previous sections, maximal models and space-efficient fair model checking, yield also a solution to the branching modular model-checking problem.

Theorem 5.1 *For all $\forall$CTL* formulas φ and ψ, and for every 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 2.1 (5), 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 2.1 (3), that $M \| M \| M' \leq M \| M_\varphi$. Thus, $M \| M' \leq M \| M_\varphi$ and therefore, by Theorem 2.1 (5), $M \| M' \models \psi$. Hence, $\langle\varphi\rangle M \langle\psi\rangle$. $\square$

It follows that branching modular model checking can be reduced to fair model checking. In Theorem 5.2 below we apply the reduction to the logics CTL and CTL*. We also show that the upper bounds that follow are tight.

Theorem 5.2

(1) *The branching modular model-checking problem for $\forall$CTL is PSPACE-complete. 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* is EXPSPACE-complete. Precisely, determining whether $\langle\varphi\rangle M\langle\psi\rangle$, for φ and ψ in $\forall$CTL*, 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 5.1, 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 3.3, 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 module. By Theorem 4.4, the size of M_φ, for φ in $\forall$CTL, is $2^{O(l)}$. Hence, the size of the module $M\|M_\varphi$ is $k2^{O(l)}$. It follows that the the problem of determining whether $\langle\varphi\rangle M\langle\psi\rangle$ can be solved in space $O(m\log^2 mk2^{O(l)}) = O(m(l+\log mk)^2)$. Theorem 3.3 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 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 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 module M, the 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 $\varphi \rightarrow \psi$ is valid iff $\langle\varphi\rangle M_{AP}\langle\psi\rangle$. 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 *PSPACE*-hardness proof in [SC85] uses temporal formulas with an unbounded number of primitive propositions. By using a Turing machine M that accepts a *PSPACE*-complete language, it is possible to bound the number of primitive proposition used to the size of the working alphabet of M. Since it is possible to encode the truth values of m primitive proposition in one state by the truth values of a single primitive proposition along $\log m$ states, it follows that satisfiability of temporal formulas with a single primitive proposition is also *PSPACE*-hard.

Since the satisfiability problem for LTL is PSPACE-hard already for formulas with a fixed number of atomic propositions [SC85], so does the implication problem for ∀CTL. 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. Indeed, the complexities given in Theorem 5.2 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].

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*. Since both the assumption and the fairness condition of a given module M now refer to its computations, we assume that M is a transition system with no fairness condition; that is, all its computations are fair. This does not weaken our results: by adding atomic propositions that mark the states of M, we can specify a fairness condition for M as part of the assumption (paying the involved increase in complexity).

We now formalize this intuition. Consider a transition system $M = \langle AP, W, R, w_0, L\rangle$. Note that, for technical convenience, we assume that M has a single initial state. A *partial path* ζ in M is a finite prefix, $w_0, w_1, \ldots, w_k$, of a path in M. We define $L(\zeta)$ to be $L(w_k)$, i.e., the label of a partial path is the label of its last state. We say that the partial path $w_0, \ldots, w_k, w_{k+1}$ *R-extends* the partial path $w_0, \ldots, w_k$. We denote the set of partial paths of M by $ppath(M)$. The transition relation R now induces a total relation R^t on partial paths in a natural way; we say that $\langle\zeta, \zeta'\rangle \in R^t$, where $\zeta, \zeta' \in ppath(M)$, if ζ' R-extends ζ.

The *tree module* of M is $M^t = \langle AP, ppath(M), R^t, w_0, L\rangle$. Note that M^t is indeed a tree; every state has a unique predecessor. A *path* π in M^t is an infinite sequence $\zeta_0, \zeta_1, \ldots$ where for all $i \geq 0$ we have $\langle\zeta_i, \zeta_{i+1}\rangle \in R^t$. We say that the path π is *anchored* if ζ_0 is w_0. With each

path $\pi = \zeta_0, \zeta_1, \ldots$, we associate a computation $L(\pi) = L(\zeta_0), L(\zeta_1), \ldots$. For an LTL formula φ and a path π, we say that π is a φ-path if $L(\pi) \models \varphi$.

We say that a state ζ in M^t satisfies a CTL* formula ψ with respect to φ, denoted $M^t, \zeta \models_\varphi \psi$, if M^t satisfies ψ with path quantification ranging only over computations that are anchored φ-paths. Similarly, $M^t, \pi \models_\varphi \xi$, for a path π of M^t and a formula ξ if the path π satisfies ξ with path quantification (which may appear in state subformulas of ξ) ranging only over computations that are anchored φ-paths.

Thus, for example,

- $M^t, \zeta \models_\varphi E\xi$ if there exists an anchored φ-path $\pi = \zeta_0, \ldots, \zeta_i, \zeta_{i+1}, \ldots$ of M^t such that $\zeta_i = \zeta$ and $M^t, \pi^i \models_\varphi \xi$.
- $M^t, \zeta \models_\varphi A\xi$ if for every anchored φ-path $\pi = \zeta_0, \ldots, \zeta_i, \zeta_{i+1}, \ldots$ of M^t such that $\zeta_i = \zeta$ we have $M^t, \pi^i \models_\varphi \xi$.

We then say that M satisfies ψ with respect to φ, denoted $M \models_\varphi \psi$, if $M^t, w_0 \models_\varphi \psi$. 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 [2]. Now, given a finite module M, an LTL formula φ, and a CTL* formula ψ, the assume-guarantee specification $[\varphi]M\langle\psi\rangle$ holds whenever $M \models_\varphi \psi$.

We now show that the branching modular framework is more general than the linear-branching modular framework.

Theorem 5.3 *For every LTL formula φ, 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: To prove the claim, we need to generalize the notion of module. An *extended module* is a pair $\langle M, P\rangle$, where M is a module and P is a set of computations of M that start from the initial state. Formulas of $\forall$CTL* can be interpreted over an extended module $\langle M, P\rangle$ by letting path quantifiers range over suffixes of computations in P. We say that a module M simulates an extended module $\langle M', P\rangle$ iff M simulates the module obtained from M' by adding a fairness condition according to which a path is fair iff it belongs to P. Recall that the truth of $[\varphi]M\langle\psi\rangle$ is determined by interpreting ψ over the extended module $\langle M^t, P_\varphi\rangle$, where P_φ is the set of anchored φ-paths of M^t.

Given φ, M, and ψ, assume first that $[\varphi]M\langle\psi\rangle$ holds. Let M' be such that $M\|M' \models A\varphi$. Then, all fair computations in $M\|M'$ satisfy φ. Therefore, $M\|M' \leq \langle M^t, P_\varphi\rangle$, and $M\|M' \models \psi$. Thus, $\langle A\varphi\rangle M\langle\psi\rangle$.

Assume now that $\langle A\varphi\rangle M\langle\psi\rangle$ holds. Since $M_{A\varphi} \models A\varphi$, it follows that $M\|M_{A\varphi} \models \psi$. For every module M, we have that $\langle M^t, P_\varphi\rangle \leq M$ and $\langle M^t, P_\varphi\rangle \leq M_{A\varphi}$. Since $\langle M^t, P_\varphi\rangle \leq M$, we

[2] We note that the formal definitions in [Jos87a, Jos87b, Jos89] apply only to restricted linear temporal assumptions and involve a complicated syntactic construction.

have that $\langle M^t, P_\varphi\rangle \leq \langle M^t, P_\varphi\rangle \| M$ too. Since $\langle M^t, P_\varphi\rangle \leq M_{A\varphi}$, we have that $\langle M^t, P_\varphi\rangle \| M \leq M_{A\varphi} \| M$ too. Thus, together, we have that $\langle M^t, P_\varphi\rangle \leq M \| M_{A\varphi}$. It follows that $\langle M^t, P_\varphi\rangle \models \psi$, which means that $[\varphi]M\langle\psi\rangle$ holds. $\square$

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 5.2, 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 use of Safra's co-determinization construction, which we used for defining maximal models in Section 4.1.

It is quite clear that an algorithm for linear-branching model checking has to generalize known algorithms for linear temporal model checking and branching temporal model checking. To see this, note that if the linear temporal assumption φ is the LTL formula **true**, then $M \models_\varphi \psi$ iff $M \models \psi$. Also, if the branching temporal guarantee ψ is the CTL formula A**false**, then $M \models_\varphi \psi$ iff $M \models \neg\varphi$. We now show how linear temporal model checking and branching temporal model checking can be combined to yield a linear-branching model-checking algorithm. For that, we first need some definitions and notations.

Let $M = \langle AP, W, R, w_0, L\rangle$. Not all paths in M are φ-paths. Hence, when the model-checking algorithm tries to satisfy a CTL* formula $E\xi$ in a state w, it has to make sure that the chosen path from w is a φ-path. But this depends not only on w but also on the partial path that led from w_0 to w. Thus, the model-checking algorithm should be applied not to M, but to the tree module M^t, in which the states are the partial paths of M. Unfortunately, M^t is an infinite module. The solution is to collapse M^t to a finite module. Instead of remembering the partial paths in their entirety, we only need to remember how the partial paths look from the "point of view" of the LTL formula φ.

Per Theorem 2.2, we construct a Büchi automaton $\mathcal{A}_\varphi = \langle 2^{AP}, S, \rho, S_0, F\rangle$ such that $\mathcal{L}(\mathcal{A}_\varphi)$ is exactly the set of computations satisfying the formula φ. We then apply to $\mathcal{A}_\varphi$ the classical *subset construction* of [RS59], that is, we extend ρ to a mapping from $2^S \times 2^{AP}$ to 2^S as follows.

$$\rho(X, a) = \{t \in S \ : \ t \in \rho(s, a) \text{ for some } s \in X\}.$$

We now combine the subset transition diagram of $\mathcal{A}_\varphi$ with M. That is, we define a transition system $M_\varphi = \langle AP, W_\varphi, R_\varphi, w^0_\varphi, L_\varphi\rangle$ as follows.

- $W_\varphi = W \times 2^S$,
- $w^0_\varphi = \langle w_0, S_0\rangle$,

- $L_\varphi(\langle w, X\rangle) = L(w)$,
- $\langle\langle u, X\rangle, \langle v, Y\rangle\rangle \in R_\varphi$ if $\langle u, v\rangle \in R$ and $Y = \rho(X, L(u))$.

Note that the definition of M_φ does not depend on M being finite. We call M_φ a *φ-annotated transition system*. Each φ-annotated transition system induces a $(2^{AP} \times 2^S)$-labeled tree obtained by unwinding it. When the tree has branching degrees in $\mathcal{D}$, for some set $\mathcal{D} \subseteq \mathbb{N}$, we say that it is a φ-annotated $\mathcal{D}$-tree.

We now define the semantics of CTL* formulas on φ-annotated transition systems. In the following definition, $\mathcal{A}_\varphi^X$ denotes the automaton $\langle 2^{AP}, S, \rho, X, F\rangle$, i.e., it is $\mathcal{A}_\varphi$ starting from the state set $X \subseteq S$. As with CTL*, we use $M_\varphi, \langle w, X\rangle \models_\varphi \psi$ to denote that the state $\langle w, X\rangle$ of M_φ satisfies the state formula ψ and use $M_\varphi, \pi \models_\varphi \xi$ to denote that the path $\pi = \langle w_1, X_1\rangle, \langle w_2, X_2\rangle, \ldots$ of M_φ satisfies the path formula ξ. For path formulas, the relation $\models$ is the same as for CTL* interpreted over modules. In particular, we have

- $M_\varphi, \pi \models \psi$, for a state formula ψ, iff $M_\varphi, \langle w_1, X_1\rangle \models \psi$.

For state formulas, we have the following.

- For all $\langle w, X\rangle$ we have that $M_\varphi, \langle w, X\rangle \models \mathbf{true}$ and $M_\varphi, \langle w, X\rangle \not\models \mathbf{false}$.
- For an atomic proposition $p \in AP$, we have $M_\varphi, \langle w, X\rangle \models p$ iff $p \in L_\varphi(\langle w, X\rangle)$ and $w \models \neg p$ iff $p \notin L_\varphi(\langle w, X\rangle)$.
- $M_\varphi, \langle w, X\rangle \models_\varphi \psi_1 \wedge \psi_2$ if $M_\varphi, \langle w, X\rangle \models_\varphi \psi_1$ and $M_\varphi, \langle w, X\rangle \models_\varphi \psi_2$.
- $M_\varphi, \langle w, X\rangle \models_\varphi \psi_1 \vee \psi_2$ if $M_\varphi, \langle w, X\rangle \models_\varphi \psi_1$ or $M_\varphi, \langle w, X\rangle \models_\varphi \psi_2$.
- $M_\varphi, \langle w, X\rangle \models_\varphi E\xi$ if there exists a path $\pi = \langle w_1, X_1\rangle, \langle w_2, X_2\rangle \ldots$ in M_φ such that $w_1 = w$, $X_1 = X$, $L(\pi)$ is accepted by $\mathcal{A}_\varphi^X$, and $M_\varphi, \pi \models_\varphi \xi$.
- $M_\varphi, \langle w, X\rangle \models_\varphi A\xi$ if for every path $\pi = \langle w_1, X_1\rangle, \langle w_2, X_2\rangle \ldots$ in M_φ such that $w_1 = w$, $X_1 = X$ and $L(\pi)$ is accepted by $\mathcal{A}_\varphi^X$, we have $M_\varphi, \pi \models_\varphi \xi$.

We claim that M_φ indeed retains all the relevant information from M^t. Thus, $M \models_\varphi \psi$ if and only if $M_\varphi, w_\varphi^0 \models_\varphi \psi$. Consider a partial path $\zeta = w_0, w_1, \cdots, w_n \in \mathit{ppath}(M)$. Let ψ be a CTL* formula. It is easy to prove, by induction on the structure of ψ, that $M^t, \zeta \models_\varphi \psi$ iff $M_\varphi, \langle w_n, X\rangle \models_\varphi \psi$, where X is the set of states in $\mathcal{A}_\varphi$ reachable from S_0 be reading $L(w_0) \cdot L(w_1) \cdots L(w_{n-1})$. Indeed, each φ-path in M^t that starts at ζ corresponds to a path that starts at $\langle w_n, X\rangle$ and is accepted by $\mathcal{A}_\varphi^X$. Taking $\zeta = w_0$, we get that $M^t, w_0 \models_\varphi \psi$ iff $M_\varphi, \langle w_0, S_0\rangle \models_\varphi \psi$. Since, by the definition, $M \models_\varphi \psi$ iff $M^t, w_0 \models_\varphi \psi$, and since $w_\varphi^0 = \langle w_0, S_0\rangle$, we are done.

In order to prove the claim, it is not hard to prove, by induction on the structure of ψ, that for every CTL* formula ψ and every $\zeta = w_0, w_1, \cdots, w_n \in ppath(M)$, we have $M^t, \zeta \models_\varphi \psi$ iff $M_\varphi, \langle w_n, X\rangle \models_\varphi \psi$, where X is the set of states in $\mathcal{A}_\varphi$ reachable from S_0 be reading $L(w_0) \cdot L(w_1) \cdots L(w_n)$.

We now prove the analogue of Theorem 2.4 for φ-annotated transition systems.

Theorem 5.4

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

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

Proof: Given an LTL formula φ, let $\mathcal{A}_\varphi = \langle 2^{AP}, S, \rho, S_0, F\rangle$ be its corresponding Büchi word automaton as described in Theorem 2.2. Thus, $\mathcal{L}(\mathcal{A}_\varphi)$ is exactly the set of computations satisfying the formula φ. Given a set $\mathcal{D} \subset \mathbb{N}$ and a CTL* formula ψ, let $\mathcal{A}_{\mathcal{D},\psi} = \langle 2^{AP}, Q', \delta', Q'_0, \langle G', B'\rangle\rangle$ be their corresponding HAA as described in Theorem 2.4. Thus, $\mathcal{L}(\mathcal{A}_{\mathcal{D},\psi})$ is exactly the set of $\mathcal{D}$-trees satisfying ψ. Let us mention some of the properties of $\mathcal{A}_{\mathcal{D},\psi}$ (for full details see [BVW94]). Each state of $\mathcal{A}_{\mathcal{D},\psi}$ may be of one of the following three types:

1. A transient state. These states belong to the transient sets of $\mathcal{A}_{\mathcal{D},\psi}$. Typically, 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 $\mathcal{A}_{\mathcal{D},\psi}$. Typically, 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 ξ. Accordingly, each state q in the set is associated with a set $\{\xi_1, \ldots, \xi_n\}$ of subformulas of ξ. When $\mathcal{A}_{\mathcal{D},\psi}$ is in state q it tries to find a path that satisfies the formulas $\xi_1, \ldots, \xi_n$.

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

Note that each set in the partition of $\mathcal{A}_{\mathcal{D},\psi}$ is associated with a state subformula of ψ and vice versa. The automaton $\mathcal{A}_{\mathcal{D},\psi}$ has a single initial state, associated with the formula ψ. When $\mathcal{A}_{\mathcal{D},\psi}$ is in an existential or a universal set and it visits a state associated with a set $\{\xi_1, \ldots, \xi_n\}$ of path formulas, it traces the automaton $\mathcal{A}_\xi^{\{\xi_1,\ldots,\xi_n\}}$ along some branch (or all branches). Since

the ξ_i's may contain state subformulas that are not propositions (and hence their truth does not follow immediately from the current letter in the input), $\mathcal{A}_{\mathcal{D},\psi}$ sends copies to states associated with such state subformulas. Consider a state q of $\mathcal{A}_{\mathcal{D},\psi}$. Assume that q belongs to the set Q in the partition. The automaton moves from the set Q to a set Q', smaller in the partial order, either when q is a transient state, in which case $\mathcal{A}_{\mathcal{D},\psi}$ moves to **true**, **false**, or to sets associated with inner subformulas of ψ, or when q is an existential or a universal state, in which case q is associated with some formulas $\xi_1, \ldots, \xi_n$, and $\mathcal{A}_{\mathcal{D},\psi}$ moves to sets associated with state subformula of the ξ_i's.

Infinite paths in a run of $\mathcal{A}_{\mathcal{D},\psi}$ get trapped in an existential or a universal set. Recall that a path that gets trapped in an existential set associated with the subformula $E\xi$ of ψ corresponds to an infinite run of $\mathcal{A}_\xi$. Therefore, the set G' is the union of the Büchi sets of all the word automata of formulas ξ for which $E\xi$ is a state subformula of ψ. Dually, B' is the union of the Büchi sets of all the word automata of formulas ξ for which $A\xi$ is a state subformula of ψ.

We define $\mathcal{A}_{\mathcal{D},\varphi,\psi} = \langle \Sigma, \mathcal{D}, Q, \delta, Q'_0, \alpha \rangle$, where

- $\Sigma = 2^{AP} \times 2^S$. That is, the inputs to $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ are φ-annotated transition systems, unwound into infinite trees.
- In order to define the set Q of states we first need some notations. With each state $s \in S$ we associate a dual state $\overline{s}$. Let $\overline{S} = \{\overline{s} \ : \ s \in S\}$ be the set of dual states. The dual $\overline{\theta}$ of a formula θ in $\mathcal{B}^+(S \cup \overline{S})$ is obtained from θ by switching **true** and **false**, switching $\vee$ and $\wedge$, and switching s and $\overline{s}$; e.g., $\overline{\overline{s_1} \vee (s_2 \wedge s_3)}$ is $s_1 \wedge (\overline{s_2} \vee \overline{s_3})$.

  Let $e(\psi), u(\psi) \subseteq Q'$ be the sets of existential and universal states of $\mathcal{A}_{\mathcal{D},\psi}$, respectively. Then,

  $$Q = S \cup \overline{S} \cup Q' \cup (e(\psi) \times S) \cup (u(\psi) \times \overline{S}).$$

  Intuitively, $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ is

  - in state $s \in S$ when it tries to find a path that is accepted by $\mathcal{A}_\varphi^{\{s\}}$,
  - in state $\overline{s} \in \overline{S}$ when it tries to show that there is no path accepted by $\mathcal{A}_\varphi^{\{s\}}$,
  - in state $q \in Q$ when it tries to satisfy q (that is, when q is a transient state, $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ tries to satisfy the formula with which is q is associated; when q is an existential state associated with a set $\{\xi_1, \ldots, \xi_n\}$ of path formulas, $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ tries to find a path that satisfies the formulas $\xi_1, \ldots, \xi_n$; when q is a universal state associated with a set $\{\xi_1, \ldots, \xi_n\}$ of path formulas, $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ tries to prove that all paths satisfy the formulas $\xi_1, \ldots, \xi_n$),
  - in state $\langle q, s \rangle$ for $q \in e(\psi)$ when it tries to find a path that satisfies the path formulas associated with q and is accepted by $\mathcal{A}_\varphi^{\{s\}}$, and
  - in state $\langle q, \overline{s} \rangle$ for $q \in u(\psi)$ when it tries to show that every path either satisfies that path formulas associated with q or is not accepted by $\mathcal{A}_\varphi^{\{s\}}$.

- For the acceptance condition, we have $G = F \cup (G' \times F)$ and $B = \overline{F} \cup (B' \times \overline{F})$.
- It remains to define the transition function δ. Consider first states in S and $\overline{S}$. We define
  - $\delta(s, \langle a, X \rangle, k) = \bigvee_{c=0}^{k-1} (c, \rho(s, a))$.
  - $\delta(\overline{s}, \langle a, X \rangle, k) = \bigwedge_{c=0}^{k-1} (c, \overline{\rho(s, a)})$.

  That is, when in state s, the automaton $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ traces the word automaton $\mathcal{A}_\varphi$ along some branch. The states in S constitute an existential set. Since $G \cap S = F$, the automaton $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ with initial state s indeed accepts all trees with a path accepted by $\mathcal{A}_\varphi^{\{s\}}$. Dually, in a state $\overline{s}$ the automaton $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ traces the automaton dual to $\mathcal{A}_\varphi$ along all branches. Since $B \cap \overline{S} = \overline{F}$, the automaton $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ with initial state $\overline{s}$ indeed accepts all trees in which no path is accepted by $\mathcal{A}_\varphi^{\{s\}}$.

  For the states in $Q' \cup (e(\psi) \times S) \cup (u(\psi) \times \overline{S})$, we define δ according to δ'. Essentially, we obtain δ from δ' by sending additional copies that take care of satisfaction of φ along the computations of the tree. Recall that the automaton $\mathcal{A}_{\mathcal{D},\psi}$ moves to a new set when it starts to trace an inner subformula. This inner formula may be of the form $E\xi$ or $A\xi$, and $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ needs to check which of the paths in the input subtree are suffixes of φ-paths (and hence, should be taken into an account in the existential and universal path quantification). In order to check whether a path in the input subtree is a suffix of a φ-path, $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ reads from its input also a subset $X \subseteq S$, which maintains the set of states that $\mathcal{A}_\varphi$ may visit after reading the prefix of the path. This subset enables $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ to determine, by checking only the suffix of a certain path, whether the path is a φ-path.

  Consider a transition $\theta = \delta'(q, \sigma, k)$. Some atoms (c, q') in θ are of a special *status* as follows:

  - An (c, q') is *existentially fresh* (ef) in $\delta'(q, \sigma, k)$ if q belongs to an existential set Q, q' belongs to a set strictly smaller than Q, and (c, q') is not conjunctively related to an atom (c', q'') for $q'' \in Q$.
  - An (c, q') is *universally fresh* (uf) in $\delta'(q, \sigma, k)$ if q belongs to a universal set Q, q' belongs to a set strictly smaller than Q, and (c, q') is not disjunctively related to an atom (c', q'') for $q'' \in Q$.
  - An atom (c, q') *stays existential* (se) in $\delta'(q, \sigma, k)$ if q and q' belong to the same existential set.
  - An atom (c, q') *stays universal* (su) in $\delta'(q, \sigma, k)$ if q and q' belong to the same universal set.

  For example, if the states q and q_2 belong to the same set Q, the states q_1, q_3, and q_4 belong to sets smaller than Q, and $\theta = \delta(q, \sigma, k)$ is $(0, q_1) \wedge (1, q_2) \vee (0, q_3) \wedge (1, q_4)$, then

  - If Q is an existential set, then q_2 has status se, and q_3 and q_4 have status ef in θ.

– If Q is a universal set, then q_1 has status uf, and q_2 has status su in θ.

Consider first a state $q \in Q'$. For every a, X, and k, we obtain $\delta(q, \langle a, X\rangle, k)$ from $\theta = \delta'(q, a, k)$ by replacing an atom (c, q') with status $y \in \{ef, uf, se, su\}$ in θ by

– $(c, q') \wedge \bigvee_{t \in \rho(X,a)}(c, t)$, if $y = ef$,
– $(c, q') \vee \bigwedge_{t \in \rho(X,a)}(c, \bar{t})$, if $y = uf$,
– $\bigvee_{t \in \rho(X,a)}(c, \langle q', t\rangle)$, if $y = se$, and
– $\bigwedge_{t \in \rho(X,a)}(c, \langle q', \bar{t}\rangle)$, if $y = su$.

Intuitively, existentially fresh atoms are atoms in which $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ leaves an existential set associated, say, with $E\xi$, and starts tracing a state subformula of ξ. Thus, $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ leaves the set $E\xi$ before making sure that ξ is satisfied along a φ-path. Therefore, $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ "activates" new copies that check for satisfaction of φ. On the other hand, atoms that stay existential are atoms in which $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ stays in the existential set and it can trace the automata $\mathcal{A}_\xi$ and $\mathcal{A}_\varphi$ together.

Consider now a state $\langle q, s\rangle \in (e(\psi) \times S) \cup (u(\psi) \times \overline{S})$. For every a, X, and k, we obtain $\delta(q, \langle a, X\rangle, k)$ from $\theta = \delta'(q, a, k)$ by replacing an atom (c, q') with status y in θ by

– $(c, q') \wedge \bigvee_{t \in \rho(X,a)}(c, t)$, if $y = ef$,
– $(c, q') \vee \bigwedge_{t \in \rho(X,a)}(c, \bar{t})$, if $y = uf$,
– $\bigvee_{t \in \rho(s,a)}(c, \langle q', t\rangle)$, if $y = se$, and
– $\bigwedge_{t \in \rho(s,a)}(c, \langle q', \bar{t}\rangle)$, if $y = su$.

The difference between states in Q' and these in $(e(\psi) \times S) \cup (u(\psi) \times \overline{S})$ is that in the latter $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ has already started to trace the automaton $\mathcal{A}_\varphi$. Therefore, it consults the set $X \subseteq S$ in the input only when it moves to fresh atoms. When it moves to atoms that stay existential or universal, it continues to trace $\mathcal{A}_\varphi$ from its current state.

We now show that $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ is a HAA. Let $Q'_1 \leq \ldots \leq Q'_n$ be the partition of the states of $\mathcal{A}_{\mathcal{D},\psi}$ into sets and the partial order over them (for simplicity, we extend it here to a total order). For an existential or universal set Q'_i let

$$Q_i = \begin{cases} Q'_i \times S & \text{If } Q'_i \text{ is an existential set,} \\ Q'_i \times \bar{S} & \text{If } Q'_i \text{ is a universal set.} \end{cases}$$

The partition of the states of $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ into sets is then S (which is an existential set), $\overline{S}$ (which is a universal set), the sets Q'_i of $\mathcal{A}_{\mathcal{D},\psi}$ (which have the same classification as in $\mathcal{A}_{\mathcal{D},\psi}$), and the sets Q_i above (which are generated from existential and universal sets of $\mathcal{A}_{\mathcal{D},\psi}$ and

have the same classification as their generators). The order between the sets is $S \leq \overline{S} \leq Q_1 \leq Q'_1 \leq \ldots \leq Q_n \leq Q'_n$.

We now consider the size and depth of $\mathcal{A}_{\mathcal{D},\varphi,\psi}$. Clearly, $|Q| \leq 2 \cdot |S| + |Q'| + 2 \cdot |Q'| \cdot |S|$. By Theorem 2.2, we have that $|S| = 2^{O(|\varphi|)}$. By Theorem 2.4, we have that $|Q'| = O(|\psi|)$ for ψ in CTL and $|Q'| = 2^{O(|\psi|)}$ for ψ in CTL*. Hence, $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ has $O(|\psi| \cdot 2^{O(|\varphi|)})$ states for ψ in CTL and has $2^{O(|\psi|+|\varphi|)}$ states for ψ in CTL*. Since the depth of $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ as most doubles the depth of $\mathcal{A}_{\mathcal{D},\psi}$, then, according to Theorem 2.4, the depth of $\mathcal{A}_{\mathcal{D},\psi}$ is $O(|\psi|)$ for both CTL and CTL*. □

Using $\mathcal{A}_{\mathcal{D},\varphi,\psi}$, the linear-branching model-checking problem can be reduced, as the usual branching model-checking problem, to the nonemptiness problem for a 1-letter word HAA. Formally, we have the following.

Theorem 5.5 *The linear-branching model-checking problem is EXPSPACE-complete for guarantees in both CTL and CTL*.*

Proof: Proving the EXPSPACE lower bound in Theorem 5.2, we use an assumption of the form $A\xi$ for an LTL formula ξ and a guarantee in ∀CTL. Therefore, the lower bound proved there is valid also here.

To get the upper bound, we have to check whether $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ accepts M^t_φ. As shown in [BVW94], this can be done by taking the product of $\mathcal{A}_{\mathcal{D},\varphi,\psi}$ with M_φ. The result is a 1-letter word HAA $\mathcal{A}_{M,\varphi,\psi}$. The size of $\mathcal{A}_{M,\varphi,\psi}$ is $|M_\varphi| \cdot |\mathcal{A}_{\mathcal{D},\varphi,\psi}|$. As M_φ has $2^{2^{O(|S|)}}$ states, the size of $\mathcal{A}_{M,\varphi,\psi}$ is double exponential for both CTL and CTL*. The result then follows from the complexity of the nonemptiness problem for 1-letter word HAA [BVW94] (see also the analogue discussion for automata-based fair model checking using LAA in Section 3.2 here).

A careful analysis of the complexity yield an algorithm that uses space $O((m(\log n + \log m + 2^{O(l)}))^2)$, where n is the size of M, m is the size of ψ, and l is the size of φ, for ψ in CTL, and uses space $O((m(\log n + m + 2^{O(l)}))^2)$, 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 can 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. Indeed, it is argued in [AL93] that assumption formulas should be small, simpler than guarantee formulas.

Our result provide an *a posteriori* justification for Josko's restriction on the linear temporal assumption [Jos87a, Jos87b, Jos89]. 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 [SZ93]) for which we can obtain such a complexity bound. 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.

Our results also provide an *a posteriori* 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.

Regardless of how one interprets our complexity results, we believe that they should renew the discussion on the relative merits of linear vs. 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 $\forall$CTL). This suggests that the tradeoff between CTL and LTL is not a simple tradeoff between complexity and expressiveness.

Acknowledgment: We thank Martin Abadi, Orna Grumberg, Pierre Wolper, and Libi 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 *Proceedings of the 5th Symposium on Logic in Computer Science*, pages 428–439, Philadelphia, June 1990.

[BVW94] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In D. L. Dill, editor, *Computer Aided Verification, Proc. 6th Int. Conference*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155, Stanford, June 1994. Springer-Verlag, Berlin.

[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.

[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.

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

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

[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.

[Jon75] N.D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68–75, 1975.

[Jon83] C.B. Jones. Specification and design of (parallel) programs. In R.E.A. Mason, editor, *Information Processing 83: Proceedings of the 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.

[KG96] O. Kupferman and O. Grumberg. Buy one, get one free! *Journal of Logic and Computation*, 6(4):523–539, 1996.

[Lam80] L. Lamport. Sometimes is sometimes "not never" - on the temporal logic of programs. In *Proceedings of the 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 *Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages*, pages 97–107, New Orleans, January 1985.

[Lyn77] N. Lynch. Log space recognition and translation of parenthesis languages. *J. ACM*, 24:583–590, 1977.

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

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

[MSS86] D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th Int. Colloquium on Automata, Languages and Programming*. Springer-Verlag, 1986.

[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.

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

[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.

[RS59] M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. of Research and Development*, 3:115–125, 1959.

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

[Sav70] W.J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *J. on Computer and System Sciences*, 4:177–192, 1970.

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

[SZ93] A.P. Sistla and L.D. Zuck. Reasoning in a restricted temporal logic. *Information and Computation*, 102(2):167–195, 1993.

[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.

[VW86a] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the First Symposium on Logic in Computer Science*, pages 322–331, Cambridge, June 1986.

[VW86b] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Science*, 32(2):182–221, April 1986.

[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.