On the Complexity of Branching Modular Model Checking (Extended Abstract)

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

[1] The Technion, Department of Computer Science, Haifa 32000, Israel.
Email: ornab@cs.technion.ac.il
[2] Rice University, Department of Computer Science, P.O. Box 1892, Houston,
TX 77251-1892, U.S.A. Email: vardi@cs.rice.edu

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 assumptions and the guarantees are specified by universal branching temporal formulas (i.e., all path quantifiers are universal). Verifying modules with respect to such specifications is called the *branching modular model-checking problem*. We consider both $\forall$CTL and $\forall$CTL*, the universal fragments of CTL and 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 model checking for both $\forall$CTL and $\forall$CTL*. In particular, branching modular model checking is PSPACE-complete for $\forall$CTL and EXPSPACE-complete for $\forall$CTL*.

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]. One of the most significant developments in this area is the discovery of algorithmic methods for verifying temporal logic properties of *finite-state* programs [CES86, LP85, QS81]. This 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, is a model of (satisfies) the propositional temporal logic formula that specifies that behavior. Hence the name *model checking* for the verification methods derived from this viewpoint. Surveys can be found in [CG87, Wol89, CGL93].

We distinguish between two types of temporal logics: linear and branching [Lam80]. In linear temporal logics, each moment in time has a unique possible future, while in branching temporal logics, each moment in time may split into several possible futures. The complexity of model checking for both linear and branching temporal logics is well understood. Suppose we are given a program of size n and a temporal logic formula of size m; for a branching temporal logic such as CTL, model-checking algorithms run in time $O(nm)$ [CES86], while, for a linear temporal logic such as LTL, model-checking algorithms run in time $O(n2^m)$ [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 ones 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]. 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. The advantage of using modular proof rules is that it enables one to apply model checking only to the underlying modules, which have much smaller state spaces.

The state-explosion problem is only one motivation for pursuing modular verification. Modular verification is advocated also for other methodological reasons; a robust verification methodology should provide rules for deducing properties of programs from the properties of their constituent modules. Indeed, efforts to develop modular verification frameworks were undertaken in the mid 1980s; [Pnu85] is a good survey.

A key observation, see [Jon83, Lam83], 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 and is studied in depth in [Var95].

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 also satisfies ψ. We call this *branching modular model checking*. Furthermore, it is argued there, as well as in [DDGJ89, Jos89, GL94, 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 $\forall$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 $\forall$CTL and in the more expressive $\forall$CTL*. We start by examining the most fundamental questions about these logics: *satisfiability*, *validity*, and *implication* (since $\forall$CTL and $\forall$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 $\forall$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 $\forall$CTL* formulas. While maximal models for $\forall$CTL involve an exponential blow-up, maximal models for $\forall$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 $\forall$CTL and $\forall$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 $\forall$CTL.

We turn on to the branching modular model-checking problem. It turns out that the same two fundamental techniques of maximal models and space-efficient fair model checking also yield a solution to the branching modular model-checking problem. We prove that the problem is PSPACE-complete for $\forall$CTL and EXPSPACE-complete for $\forall$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.

2 Preliminaries

2.1 Universal Branching Temporal Logics

We assume that the reader is familiar with the syntax and the semantics of the linear temporal logic LTL, as well as with these of the branching temporal logics CTL* and CTL. The formal definition of these logics can be found in [Eme90]. We define here the universal fragments $\forall$CTL* and $\forall$CTL of CTL* and CTL.

The logic $\forall CTL^\star$ combines both branching-time and linear-time operators. A path quantifier A ("for all computations") can prefix an assertion composed of an arbitrary combination of the linear-time operators X ("next time") and U ("until"). There are two types of formulas in $\forall$CTL*: *state formulas*, whose satisfaction is related to a specific state, and *path formulas*, whose satisfaction is related to a specific computation. Formally, let AP be a set of atomic proposition names. A $\forall$CTL* state formula is either:

- $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$, where φ_1 and φ_2 are $\forall$CTL* state formulas.
- $A\psi_1$, where ψ_1 is an $\forall$CTL* path formula.

An $\forall$CTL* path formula is either:

- **true**, **false**, p, or $\neg p$, for all $p \in AP$.
- A CTL* state formula.
- $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$, $X\psi_1$, $\psi_1 U \psi_2$, or $\psi_1 \tilde{U} \psi_2$, where ψ_1 and ψ_2 are $\forall$CTL* path formulas.

$\forall$CTL* is the set of state formulas generated by the above rules.

Note that unlike the logic CTL*, the only path quantifier allowed in $\forall$CTL* is the universal one. As negation in $\forall$CTL* can be applies only to atomic propositions, assertions of the form $\neg A\psi$, which is equivalent to $E\neg\psi$, are not allowed either. Note also that $\forall$CTL* interprets atomic propositions, as well as the Boolean constants **true** and **false**, with respect to paths. Thus, propositional assertions are always in a scope of a universal path quantifier. The logic $\forall$CTL is a restricted subset of $\forall$CTL* in which the temporal operators must be immediately preceded by the universal path quantifier.

We define the semantics of $\forall$CTL* with respect to *fair 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 $R \subseteq W \times W$ (i.e., for every $w \in W$ there exists $w' \in W$ such that $\langle w, w' \rangle \in R$), a set W^0 of initial states, a labeling function $L : W \rightarrow 2^{AP}$ that maps each state to a set of atomic propositions that hold in this state, and a Rabin fairness condition α (our choice of this type of fairness condition is technically motivated, as will be clarified in the sequel). We use $w \models \varphi$ to denote that the state w satisfies the formula φ. 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 $\langle w_i, w_{i+1} \rangle \in R$. A computation of M is *fair* iff it satisfies the fairness condition α. That is, 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 $w_j \in G_i$ for infinitely many j's and $w_j \in B_i$ for finitely many j's. When defining the semantics of CTL* over modules, path quantifiers range only over fair computations, cf. [CES86]. In particular, the formula Ap, for an atomic proposition p, means that p holds in the first state for all fair paths. Thus, for a state $w \in W$, we have that $w \models Ap$ iff $p \in L(w)$ or there exists no fair path starting at w. A module M satisfies a formula φ if φ holds in *all* initial states of M. We say that a module is *nonempty* iff there exists a fair computation that starts at an initial state.

2.2 Simulation Relation and Composition of Modules

In the context of modular verification, there is a need to define an order relation between modules [GL94]. Intuitively, we want to define what it means for a module M' to have "more behaviors" than a module M.

Let M and M' be two modules with sets $AP' \subseteq AP$ of atomic propositions, 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 path $\pi' = s'_0, s'_1, \ldots$ 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'.

Let M and M' be two modules with sets AP and AP' of atomic propositions. The *composition* of M and M', denoted $M \| M'$, is a module that has exactly these behaviors which are joint to M and M' and are fair in both of them. In order to define $M \| M'$, we define first a structure $M'' = \langle AP'', W'', R'', W^{0''}, L'' \rangle$, which has no fairness condition and which has exactly the joint behaviors of M and M', ignoring their fairness conditions. Thus, $AP'' = AP \cup AP'$, $W'' = \{\langle w, w' \rangle : L(w) \cap AP' = L(w') \cap AP\}$, $W^{0''} = (W^0 \times W^{0'}) \cap W''$, $R'' = \{\langle \langle w, w' \rangle, \langle s, s' \rangle \rangle : \langle w, s \rangle \in R$ and $\langle w', s' \rangle \in R'\}$, and $L''(\langle w, w' \rangle) = L(w) \cup L'(w')$ for $\langle w, w' \rangle \in W''$. In order to get $M \| M'$, we have to constrain M'' so that its behaviors will be fair in both M and M'. Clearly, a computation π in M'' is fair iff there exist a pair $\langle G, B \rangle \in \alpha$ and a pair $\langle G', B' \rangle \in \alpha'$ such that π visits states from $G \times W'$ infinitly often, visits states from $W \times G'$ infinitly often, and visits states from $(B \times W') \cup (W \times B')$ only finitely often. Thus, checking the fairness of π with respect to such pair of pairs can be done, as in the product of two Büchi automata, by augmenting each state of M'' with a Boolean flag. For mm' such pairs of pairs (originating from m pairs in α and m' pairs in α'), checking the fairness of π can be done, by augmenting each state of M'' with a $2mm'$-valued flag, resulting in $M \| M'$ with $2mm'|W''|$ states.

Theorem 1. [GL94]

(1) *The simulation relation $\leq$ is a preorder.*
(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 $M' \models \varphi$. 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 Alternating Tree Automata

For an introduction to the theory of automata on infinite trees see [Tho90]. *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 $\mathcal{A} = \langle \Sigma, Q, \delta, Q^0, F \rangle$. 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).

Let $\mathcal{B}^+(\{0,1\} \times Q)$ be the set of positive Boolean formulas over $\{0,1\} \times Q$ (i.e., Boolean formulas built from elements in $\{0,1\} \times Q$ using $\wedge$ and $\vee$), where we also allow the formulas **true** and **false** and, as usual, $\wedge$ has precedence over $\vee$. 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, an *alternating tree automaton* is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, Q^0, F\rangle$, where Σ is the input alphabet, Q is a finite set of states, $Q^0 \subseteq Q$ is a set of initial states, F specifies the acceptance condition, and $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(\{0,1\} \times Q)$ is the transition function. If $|\Sigma| = 1$, then we call A a 1-letter automaton. A run r of an alternating automaton $\mathcal{A}$ on a binary tree T is a tree where the root is labeled by $q_0 \in Q^0$ and every other node is labeled by an element of $\{0,1\} \times Q$. Each node of r corresponds to a node of T. Suppose that the path to a node y in r is labeled by $q_0, (c_1, q_1), \ldots, (c_m, q_m)$, then y corresponds to the node $x = c_1 \cdots c_m$ of T. Intuitively, the node y of r describes the automaton reading the node x of T. Note that many nodes of r can correspond to the same node of T; in contrast, in a run of a nondeterministic automaton on T 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. The run is accepting if all its infinite paths satisfy the acceptance condition. We use $\mathcal{L}(\mathcal{A})$ to denote the language of $\mathcal{A}$; i.e., the set of trees that $\mathcal{A}$ accepts. An automaton is *nonempty* if it accepts some tree. For formal details see [MS87]. Generalizing the definition to trees with a fixed branching degree n, we define the transition function as $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(\{0, \ldots n-1\} \times Q)$.

3 An Automata-Theoretic Framework to Fair Model Checking

3.1 Libi Alternating Automata

In [BVW94], we introduced *hesitant alternating automata* (HAA). An HAA that run over trees of a fixed branching degree n 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, $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(\{0, \ldots n-1\} \times Q)$ is a transition function, $Q^0 \subseteq Q$ is a set of initial states, 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 $\mathcal{A}$.

Theorem 2. [BVW94]

(1) *Given a CTL formula ψ and $n \in \mathbb{N}$, there exists an HAA $\mathcal{A}_\psi = \langle 2^{AP}, Q, \delta, Q^0, \alpha\rangle$ of size and depth $O(|\psi|)$ such that $\mathcal{L}(\mathcal{A}_\psi)$ is precisely the set of n-ary tree structures satisfying ψ.*
(2) *Given a CTL* formula ψ and $n \in \mathbb{N}$, there exists an HAA $\mathcal{A}_\psi = \langle 2^{AP}, Q, \delta, Q^0, \alpha\rangle$ of size $2^{O(|\psi|)}$ and depth $O(|\psi|)$ such that $\mathcal{L}(\mathcal{A}_\psi)$ is precisely the set of n-ary tree structures satisfying ψ.*

We briefly describe here the construction of HAA for $\forall$CTL(which is, essentially, a special case of the construction of HAA for CTL). Given ψ, the state set Q is the set of all the subformulas of ψ augmented with the state A**false**. The acceptance condition α is $\langle\emptyset, B\rangle$, where B consists of all the formulas of the form $A\varphi_1 U \varphi_2$ in Q and the state A**false**. The set of initial states in the singleton $\{\psi\}$. It remains to define the transition function δ. For all $\sigma \in 2^{AP}$, we define:

- $\delta(Ap, \sigma) =$ **true** if $p \in \sigma$. - $\delta(Ap, \sigma) = A$**false** if $p \notin \sigma$.
- $\delta(A\neg p, \sigma) =$ **true** if $p \notin \sigma$. - $\delta(A\neg p, \sigma) = A$**false** if $p \in \sigma$.
- $\delta(A\mathbf{false}, \sigma) = \bigwedge_{c=0}^{n-1}(c, A\mathbf{false})$.
- $\delta(\varphi_1 \wedge \varphi_2, \sigma) = \delta(\varphi_1, \sigma) \wedge \delta(\varphi_2, \sigma)$.
- $\delta(\varphi_1 \vee \varphi_2, \sigma) = \delta(\varphi_1, \sigma) \vee \delta(\varphi_2, \sigma)$.
- $\delta(AX\varphi_2, \sigma) = \bigwedge_{c=0}^{n-1}(c, \varphi_2)$.
- $\delta(A\varphi_1 U \varphi_2, \sigma) = \delta(\varphi_2, \sigma) \vee (\delta(\varphi_1, \sigma) \wedge \bigwedge_{c=0}^{n-1}(c, A\varphi_1 U \varphi_2))$.
- $\delta(A\varphi_1 \tilde{U} \varphi_2, \sigma) = \delta(\varphi_2, \sigma) \wedge (\delta(\varphi_1, \sigma) \vee \bigwedge_{c=0}^{n-1}(c, A\varphi_1 \tilde{U} \varphi_2))$.

It is easy to see that the state A**false** constitutes a universal set. The task of this state will get clearer later.

In order to perform fair model checking, we introduce here *Libi alternating automata* (LAA). An LAA is an HAA with an acceptance condition $\alpha \in 2^Q \times 2^Q$ extended with a Rabin acceptance condition $\beta \subseteq 2^Q \times 2^Q$. For a run of an LAA with $\beta = \{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$, a path π which 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 $inf(\pi) \cap B_j = \emptyset$.

We now show that the space complexity result of [BVW94] for the 1-letter nonemptiness of HAA extends to LAA.

Theorem 3. *The 1-letter nonemptiness problem for LAA of size n and depth m can be solved in space polynomial in $m \log n$.*

Proof. Consider an LAA with $\alpha = \langle B, G \rangle$ and $\beta = \{\langle G_1, B_1 \rangle, \ldots, \langle G_k, B_k \rangle\}$. The property of LAA we use is that, from a state in Q_i, it is possible to search for another reachable state in the same Q_i nondeterministically using space $O(\log n)$. For 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 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 known 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. A key step in our algorithm is the evaluation of Boolean expressions. It is known that this can be done using space that is logarithmic in the size of the expression [Lyn77]. Here, we evaluate expressions over Q, using a recursion of depth at most m to trace their Boolean value.

We call a state q' of Q_i *provably true* if, when the evaluation 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. Also, for a set $S \subseteq Q$ and two states q and q', we say that q' is S*-reachable* from q, if q' is reachable from q using states from S only (in particular, q and q' are members of S).

The following recursive procedure labels the states of the automaton with 'T' (accepts) or 'F' (does not accept). The language of the LAA is nonempty iff an initial state is labeled 'T'.

(1) One starts at an initial state.

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

(3) At a state q of an existential Q_i, one proceeds as follows.

- **(3.1)** One searches 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, the state q is labeled 'T'.
- **(3.2)** If no such state exists, one guesses $1 \leq j \leq k$ and searches 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 $q' = q''$), then q is labeled 'T'.
- **(3.3)** if none of the first two cases apply, q is labeled 'F'.

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

- **(4.1)** One searches for a reachable state q' of the same Q_i that is provably false. If such a state q' is found, the state q is labeled 'F'.
- **(4.2)** If no such state exists, one guesses $1 \leq j \leq k$ and searches 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, then q is labeled 'F'.
- **(4.3)** if none of the first two cases apply, q is labeled 'T'.

The procedure is recursive, but since each recursive call takes us to a lower Q_i, the depth of the recursion is bounded by m. Since each invocation of the procedure can be executed nondeterministically using space $O(\log n)$, the whole procedure can thus be executed using space $O(m \log n)$. The claim then follows by Savitch Theorem [Sav70].

3.2 Fair Model Checking

It is well known that for nondeterministic tree automata the nonemptiness problem can be reduced to the 1-letter nonemptiness problem [Rab69]. As discussed in [BVW94], this is not the case for alternating tree automata, where the 1-letter nonemptiness problem is easier than the general nonemptiness problem. The key idea underlying the automata-theoretic approach to branching temporal model checking is that by taking the product of an alternating automaton with a module we get an alternating automaton over a 1-letter alphabet.

Given a branching temporal logic formula ψ, let $\mathcal{A}_\psi = \langle \Sigma, Q_\psi, \delta_\psi, \psi, \alpha_\psi \rangle$ be an HAA that accepts precisely all the n-ary tree models of ψ. Consider a module $M = \langle \Sigma, W, R, W^0, L, \beta_M \rangle$. For $w \in W$, let $succ_R(w) = \langle w_0, \ldots, w_{n-1} \rangle$ be an ordered list of w's R-successors (it is technically convenient to assume that each state has n successors and that the states of W are ordered). The product automaton of $\mathcal{A}_\psi$ and M is the LAA $\mathcal{A}_{M,\psi} = \langle \{a\}, W \times Q_\psi, \delta, W^0 \times \{\psi\}, \alpha, \beta \rangle$, where δ, α, and β are defined as follows:

- Let $q \in Q_\psi, w \in W, succ_R(w) = \langle w_0, \ldots, w_{n-1} \rangle$, and $\delta_\psi(q, L(w)) = \theta$. 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 $(c, \langle w_c, \eta \rangle)$.
- For $\alpha_\psi = \langle G, B \rangle$, we have that $\alpha = \langle W \times G, W \times B \rangle$.
- For $\beta_M = \{\langle G_1, B_1 \rangle, \ldots, \langle G_k, B_k \rangle\}$, we have that $\beta = \{\langle G_1 \times Q_\psi, B_1 \times Q_\psi \rangle, \ldots, \langle G_k \times Q_\psi, B_k \times Q_\psi \rangle\}$.

Proposition 4.

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

Let $Q_1 \leq \ldots \leq Q_m$ be the partition of the states of $\mathcal{A}_\psi$ 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 $\mathcal{A}_\psi$.

In conclusion, given an HAA $\mathcal{A}_\psi$ that accepts precisely all the n-ary tree models of the formula ψ, model checking of a module M with a fixed branching degree n with respect to ψ is reducible to checking the 1-letter nonemptiness of an LAA of size $O(|M| * |\mathcal{A}_\psi|)$ and of the same depth as $\mathcal{A}_\psi$. Thus, Theorem 3 implies the theorem below.

Theorem 5. *Given an HAA $\mathcal{A}_\psi$ of size n and depth m, and a module M of size l, model checking of M with respect to ψ can be done using space polynomial in $m \log(nl)$.*

Theorem 5, together with the HAA described in Theorem 2, imply that the fair model-checking problem for both CTL and CTL* is solvable in space polynomial in $|\psi| \log |M|$. 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 problem for a specification language are *satisfiability*, i.e., whether the specification is not trivially false, *validity*, i.e., whether the specification is trivially true, and *implication*, i.e., whether one specification logically implies 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 $\forall$CTL* formula, 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 ψ hold in every module in which φ holds? We now show that the satisfiability and the validity problems are special cases of the implication problem.

Lemma 6. *Let φ be a formula in $\forall$CTL*. 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 A***true*** implies φ.*

4.1 Maximal Models and Implication

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). The following theorem is proved in [GL94] for $\forall$CTL, but the proof applies also to $\forall$CTL*.

Theorem 7. *Let φ and ψ be $\forall$CTL* 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 8. *For every $\forall$CTL* formula φ, there exists a maximal model M_φ of size $2^{2^{O(|\varphi|)}}$.*

Proof. For a $\forall$CTL* formula φ, let $sf(\varphi)$ denote the set of state subformulas of φ. Given φ, let $\forall(\varphi) \subseteq sf(\varphi)$ denote the set of all the state subformulas of φ of the form $A\xi$. Let $\mathcal{A}_{\forall(\varphi)}$ be a Büchi ω-automaton over $\Sigma = 2^{sf(\varphi)}$ such that $\mathcal{A}_{\forall(\varphi)}$ accepts an infinite word $\pi = w_0, w_1, \ldots$ iff there exists a suffix $w_i, w_{i+1}, \ldots$ of π and a formula $A\xi \in \forall(\varphi)$ such that $A\xi \in w_i$ and $w_i, w_{i+1}, \ldots$ does not satisfy ξ. 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 $\bar{\mathcal{A}}_{\forall(\varphi)}$, is a deterministic Rabin automaton which 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 $\bar{\mathcal{A}}_{\forall(\varphi)}$ 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 $L(w) = w \cap AP$. So far M is more general than any model of φ, but it is not necessarily a model of φ. To make it a maximal model, we take the product of M with $\bar{\mathcal{A}}_{\forall(\varphi)}$ as follows. Let $\bar{\mathcal{A}}_{\forall(\varphi)} = \langle \Sigma, Q, \delta, q^0, \beta \rangle$, 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 and β' are defined as follows.

– $R = \{\langle \langle w, q \rangle, \langle w', q' \rangle \rangle : \delta(q, w) = q'\}$.
– $\beta' = \{\langle c(\varphi) \times G_1, c(\varphi) \times B_1 \rangle, \ldots, \langle c(\varphi) \times G_k, c(\varphi) \times B_k \rangle\}$.

Theorem 9. *For every ∀CTL formula φ, there exists M_φ of size $2^{O(|\varphi|)}$.*

Proof. Exactly as for ∀CTL⋆. Here, however, $\bar{\mathcal{A}}_{\forall(\varphi)}$ is of size $2^{O(|\varphi|)}$.

4.2 Complexity

We are now ready to combine the maximal-model technique with the space-efficient fair-model-checking algorithm to solve the implication problem.

Theorem 10. *Given two universal branching temporal logic formulas φ and ψ, the implication problem $\varphi \to \psi$ can be solved in space polynomial in $m \log(nl)$ where m and n are the depth and the size, respectively, of an HAA corresponding to ψ, and l is the size of a maximal model for φ.*

Proof. By Theorem 7, the validity problem of $\varphi \to \psi$ is reducible to model checking of ψ in M_φ. The result follows from Theorem 5.

The complexity of our method with respect to two logics L_1 and L_2 depends on the ability to associate with formulas of L_2 HAA, in the size of these HAA, their depth, as well as in the size of the maximal models for formulas in the logic L_1. In Theorem 11 below, we apply the method to the logics CTL and CTL⋆. We also show that the upper bounds that follow are tight.

Theorem 11.

(1) *The implication problem for ∀CTL is PSPACE-complete.*
(2) *The implication problem for ∀CTL⋆ is EXPSPACE-complete.*

Proof. Theorems 8 and 9 provide us with a doubly-exponential bound for maximal models of ∀CTL⋆ and an exponential bound for maximal models of ∀CTL. Theorem 2 provides us with sizes and depths for HAA of CTL and CTL⋆. Together with Theorem 10, this yields the upper bounds. The PSPACE lower bound for the implication problem for ∀CTL, follows from the PSPACE lower bound for the satisfiability problem for ∀CTL. The later is proved by a reduction from LTL satisfiability, proved to be PSPACE-hard in [SC85], and the result then follows from Lemma 6. The EXPSPACE lower bound for the implication problem for ∀CTL⋆ is proved similarly to the EXPSPACE lower bound proved in [Var95] for linear-branching modular model checking.

Note that the implication problem $\varphi \to \psi$ for φ in $\forall$CTL and ψ in $\forall$CTL* is still in PSPACE.

We now consider the satisfiability and validity problems, which, as observed earlier, are special cases of the implication problem.

Theorem 12.

(1) *The satisfiability problems for $\forall$CTL and $\forall$CTL* are PSPACE-complete.*
(2) *The validity problem for $\forall$CTL is NP-complete.*
(3) *The validity problem for $\forall$CTL* is PSPACE-complete.*

Proof. The results for the satisfiability follow from the PSPACE-completeness of the satisfiability problem for LTL. Since $\forall$CTL subsumes propositional logic, hardness in NP for the validity problem for $\forall$CTL is immediate. Membership in NP follows from a linear-size model property for $\exists$CTL. It remains to prove PSPACE-completeness for the validity problem for $\forall$CTL*. Since a $\forall$CTL* formula ψ is valid iff the implication $A\mathbf{true} \to \psi$ is valid, and since the implication problem $\varphi \to \psi$ 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.

5 Modular Model Checking

5.1 Branching Modular Model Checking

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, i.e., maximal models and space-efficient fair model checking, yield also a solution to the branching modular model-checking problem.

Theorem 13. *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 $M\|M_\varphi \le M_\varphi$, we have 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, $M\|M' \le M_\varphi$, which implies that $M\|M\|M' \le M\|M_\varphi$. Thus, $M\|M' \le M\|M_\varphi$ and therefore $M\|M' \models \psi$. Hence, $\langle\varphi\rangle M\langle\psi\rangle$.

Theorem 14. *Given a module M and two universal branching temporal logic formulas φ and ψ, the problem of determining whether $\langle\varphi\rangle M\langle\psi\rangle$ is solvable in space polynomial in $m\log(nl * |M|)$ where m and n are the depth and the size, respectively, of an HAA corresponding to ψ, and l is the size of a maximal model for φ.*

Proof. By Theorem 13, the problem of determining whether $\langle\varphi\rangle M\langle\psi\rangle$ is reducible to model checking of ψ in $M\|M_\varphi$. The result then follows from Theorem 5.

In Theorem 15 below we apply the method to the logics CTL and CTL*. We also show that the upper bounds that follow are tight.

Theorem 15.

(1) *The branching modular model-checking problem for $\forall$CTL is PSPACE-complete.*
(2) *The branching modular model-checking problem for $\forall$CTL* is EXPSPACE-complete.*

Proof. The upper bounds follow from Theorem 14. The lower bound for $\forall$CTL* follows from [Var95] and the relation between the branching modular framework and the linear-branching modular framework (to be discussed in the next section). To get the lower bound for $\forall$CTL, we reduce the implication problem for $\forall$CTL to branching modular model-checking of a fixed-size module.

In the full paper we provide a more detailed analysis of the complexity of branching modular model checking. 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. 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*. This problem is studied in [Var95]. We now show that the branching modular framework is more general than the linear-branching modular framework.

Theorem 16. *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 some 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. The notion of simulation can be easily extended to extended modules. As detailed in [Var95], the truth of $[\varphi]M\langle\psi\rangle$ is determined by interpreting ψ over the extended module $\langle M_t, P_\varphi\rangle$, where M_t is the computation tree of M and P_φ is the set of computations of M_t that satisfy φ.

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. Since $M\|M' \leq \langle M_t, P_\varphi\rangle$, it follows that $M\|M' \models \psi$. 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 $\langle M_t, P_\varphi\rangle \leq M_{A\varphi}$. This can be shown to imply 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.

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 [CD88]. Thus, the branching modular framework is strictly more expressive than the linear-branching modular framework, with no increase in worst-case computational complexity (it is shown in [Var95] that linear-branching modular model checking is EXPSPACE-complete). We note, however, that the algorithm in [Var95] for linear-branching modular model checking is simpler than the algorithm here, as it avoids the use of Safra's co-determinization construction.

Acknowledgment: We thank Libi for many helpful suggestions and discussions and Orna Grumberg for useful comments on an earlier draft of this paper.

References

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

[BCM+90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc. 5th LICS*, pp. 428–439, 1990.

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

[CD88] E.M. Clarke and I.A. Draghicescu. Expressibility results for linear-time and branching-time logics. In *Proc. Workshop on Linear Time, Branching Time, and Partial Order in Logics and Models for Concurrency*, pp. 428–437. LNCS, Springer-Verlag, 1988.

[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, 1986.

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

[CGL93] E.M. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, eds., *Decade of Concurrency – Reflections and Perspectives (Proc. of REX School)*, LNCS, pp. 124–175. Springer-Verlag, 1993. g,

[CLM89] E.M. Clarke, D.E. Long, and K.L. McMillan. Compositional model checking. In *Proc. 4th LICS*, pp. 353–362, 1989.

[DDGJ89] W. Damm, G. Döhmen, V. Gerstner, and B. Josko. Modular verification of Petri nets: the temporal logic approach. In *Proc. REX Workshop*, LNCS 430, pp. 180–207, 1989.

[DGG93] D. Dams, O. Grumberg, and R. Gerth. Generation of reduced models for checking fragments of CTL. In *Proc. 5th CAV*, LNCS 697, pp. 479–490, 1993.

[EJ88] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. of the 29th FOCS*, 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*, 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.

[Eme90] E.A. Emerson. Temporal and modal logic. *Handbook of theoretical computer science*, pp. 997–1072, 1990.

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

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

[Jos87a] B. Josko. MCTL – an extension of CTL for modular verification of concurrent systems. In *Temporal Logic in Specification*, LNCS 398, pp. 165–187, 1987.

[Jos87b] B. Josko. Model checking of CTL formulae under liveness assumptions. In *Proc. 14th ICALP*, LNCS 267, pp. 280–289, 1987.

[Jos89] B. Josko. Verifying the correctness of AADL modules using model chekcing. In *Proc. REX workshop*, LNCS 430, pp. 386–400, 1989.

[Lam80] L. Lamport. Sometimes is sometimes "not never" - on the temporal logic of programs. In *Proc. 7th POPL*, pp. 174–185, 1980.

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

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 20th POPL*, pp. 97–107, 1985.

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

[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 ICALP*, 1986.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th FOCS*, pp. 46–57, 1977.

[Pnu85] A. Pnueli. In transition from global to modular temporal reasoning about programs. In *Logics and Models of Concurrent Systems*, vol. F-13 of *NATO Advanced Summer Institutes*, pp. 123–144, 1985.

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

[Rab69] M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.

[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.

[Tho90] W. Thomas. Automata on infinite objects. *Handbook of theoretical computer science*, pp. 165–191, 1990.

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

[VS85] M.Y. Vardi and L. Stockmeyer. Improved upper and lower bounds for modal logics of programs. In *Proc 17th STOC*, pp. 240–251, 1985.

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

[Wol89] P. Wolper. On the relation of programs and computations to models of temporal logic. In *Proc. Temporal Logic in Specification*, LNCS 398, pp. 75–123, 1989.