On the Complexity of Modular Model Checking

Moshe Y. Vardi
Rice University*

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 environment with which the module is interacting. This is called the *assume-guarantee* paradigm. Even when one specifies the guaranteed behavior of the module in a branching temporal logic, the assumption in the assume-guarantee pair concerns the interaction of the environment with the module along each computation, and is therefore often naturally expressed in linear temporal logic. In this paper we consider assume-guarantee specifications in which the assumption is given by an LTL formula and the guarantee is given by a CTL formula. Verifying modules with respect to such specifications is called the *linear-branching model-checking problem*. We apply automata-theoretic techniques to obtain a model-checking algorithm whose running time is linear in the size of the module and the size of the CTL guarantee, but doubly exponential in the size of the LTL assumption. We also show that the high complexity in the size of the LTL specification is inherent by proving that the problem is EXPSPACE-complete. The lower bound applies even if the branching temporal guarantee is restricted to be specified in ∀CTL, the universal fragment of 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, Pnu81]. One of the most significant developments in this area is the discovery of algorithmic methods for verifying temporal logic properties of *finite-state* programs [CES86, LP85, QS81]. This derives its significance both from the fact that many synchronization and communication protocols can be modeled as finite-state programs, as well as from the great ease of use of fully algorithmic methods. Finite-state programs can be modeled by transition systems where each state has a bounded description, and hence can be characterized by a fixed number of boolean atomic propositions. This means that a finite-state program can be viewed as a finite *propositional Kripke structure* and its properties can be specified using *propositional* temporal logic. Thus, to verify the correctness of the program with respect to a desired behavior, one only has to check that the program, modeled as a finite Kripke structure, satisfies (is a model of) the propositional temporal logic formula that specifies that behavior. Hence the name *model checking* for the verification methods derived from this viewpoint. Surveys can be found in [CG87, Wol89, CGL93].

We distinguish between two types of temporal logics: linear and branching [Lam80]. In linear temporal logics, each moment in time has a unique possible future, while in branching temporal logics, each moment in time may split into several possible futures. The complexity of model checking for both linear and branching temporal logics is well understood. Suppose we are given a program of size n and a temporal specification of size m. For a branching temporal logic such as CTL, model-checking algorithms run in time that $O(nm)$ [CES86], while, for linear temporal logic such as LTL, model-checking algorithms run in time $O(n2^m)$ [LP85]. The latter bound probably cannot be improved, since model checking with respect to linear temporal specification is PSPACE-complete [SC85]. 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, ASSSV94]. In modular verification, one uses proof rules of the following form:

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

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

*Work partly done at the IBM Almaden Research Center. Address: Department of Computer Science, P.O. Box 1892, Houston, TX 77251-1892, U.S.A., email: vardi@cs.rice.edu, fax: 713-285-5930.

smaller state spaces.

The state-explosion problem is of course 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. Thus, 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 environment with 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 its environment 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 formulas. The meaning of such a pair is that the behavior of the module is guaranteed to satisfy ψ, assuming that the behavior of the environment satisfies φ. As observed in [Pnu85], in this case the assume-guarantee pair can be combined to a single linear temporal specification $\varphi \rightarrow \psi$. Thus, verifying a module with respect to assume-guarantee linear temporal pairs is essentially the same as verifying the module with respect to linear temporal 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. In contrast, the assumption in the assume-guarantee pair concerns the interaction of the environment with the module along each computation, and is therefore often naturally expressed in linear temporal logic. This point is already implicit in [CES86]. As explained there, in many applications we can prove that the module behaves in the desired way only under a *fairness* assumption. This assumption concerns the interaction of the environment with the module along every computation is not expressible in CTL. For this reason, the basic model-checking algorithm for CTL is extended in [CES86] to handle also fairness assumptions (see also [EL85, EL87]).

This point was made explicit by Josko and his collaborators [Jos87a, Jos87b, Jos89, DDGJ89], who have demonstrated by many examples that an assume-guarantee pair for branching temporal verification 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 satisfies such a pair, which we call the *linear-branching* model-checking problem, is more general than either linear or branching model checking and has received little attention in the literature.

Josko has considered a special case of the linear-branching model-checking problem [Jos87a, Jos87b, Jos89]. In his formalism, called MCTL, an assume-guarantee pair $\langle \varphi, \psi \rangle$ consists of a CTL formula ψ and an LTL formula φ of a special form (Josko also defined GMCTL, in which somewhat more general LTL formulas are allowed). He then showed that one can model check such assume-guarantee pairs in time $O(nm2^l)$, where n is the size of the module, m is the size of the CTL guarantee, and l is the size of the LTL assumption. Josko also showed that the PSPACE-hardness lower bound for linear model checking [SC85] applies also to the linear-branching model-checking problem.

In this paper we investigate the linear-branching model-checking problem 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 [VW86, BVW94]. In both cases, one associates with each temporal formula a finite automaton on infinite structures that accepts exactly all the structures that satisfy the formula. For linear temporal logic the relevant structures are infinite computations and the automata used are nondeterministic Büchi automata, while for branching temporal logic the relevant structures are infinite computation trees and the automata used are alternating automata.

It is quite clear that an algorithm for linear-branching model checking has to generalize known algorithms for linear temporal model checking and for branching temporal model checking. To solve the linear-branching model checking problem, we combine linear temporal model checking and branching temporal model checking. The semantics of the assume-guarantee specification is defined in terms of the computation tree of the module. 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. We get an algorithm whose time complexity is $nm2^{2^{O(l)}}$, where n is the size of the module, m is the size of the CTL guarantee, and l is the size of the LTL assumption. A careful analysis then yields an algorithm whose space complexity is $O\big((m(\log n + \log m + 2^{O(l)}))^2\big)$.

Can these formidable bounds be improved? We show that the exponential space complexity in the size of the assumption is inherent by proving that the problem is EXPSPACE-hard. In fact, we show that the lower bound apply even we restrict the branching temporal guarantee to be specified in $\forall$CTL, the universal fragment of CTL. In $\forall$CTL, one can quantify over computations universally but not existentially. That is, in $\forall$CTL one can state properties of all computations of the program, but one cannot state that certain computations exist. It has been argued by many researchers that this fragment is sufficiently expressive for many verification applications and is advantageous for modular verification (cf. [DDGJ89, Jos89, DGG93, GL94]). Our result shows that even under this restriction, the complexity of the linear-branching model-checking problem in the size of the linear temporal assumption is very high.

2 Preliminaries

2.1 Linear and Branching Temporal Logics

Linear temporal logic (LTL) is a language of assertions about computations. Its formulas are built from atomic propositions AP by means of Boolean connectives and the temporal connectives X ("next time") and U ("until"). In contrast, CTL (Computation Tree Logic) is a language of assertions about computation trees. Its temporal connectives consist of path quantifiers immediately followed by a single linear-temporal operator. The path quantifiers are A ("for all paths") and E ("for some path"). A CTL formula in *positive normal form* is a CTL formula in which negations are applied only to atomic propositions. It can be obtained by pushing negations inward as far as possible, using De Morgan's laws and dualities. For technical convenience, we use the linear-time operator $\tilde{U}$ as a dual of the U operator, and write all CTL formulas in a positive normal form.

The semantics of LTL is defined with respect to *computations.* A computation $\sigma : \omega \rightarrow 2^{AP}$ is an infinite sequence of truth assignments to the atomic propositions. We use the notation $\sigma, i \models \varphi$ to denote that φ holds at time i on the computation σ. The semantics of CTL is defined with respect to *programs.* A *program* $P = \langle W, R, w^0, L\rangle$ consists of 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 $(w, w') \in R$), an initial state w^0, and a labeling $L : W \rightarrow 2^{AP}$ that maps each state to a set of atomic propositions that hold in this state. We use the notation $P, w \models \varphi$ to denote that φ holds at state w of the program P. The formal definition of the relation $\models$ can be found in [Eme90].

2.2 Automata on Infinite Words

For an introduction to the theory of automata on infinite words and trees see [Tho90].

The types of finite automata on infinite words we consider are those defined by Büchi [Büc62]. A (nondeterministic) automaton on words is a tuple $A = \langle\Sigma, S, S_0, \rho, \alpha\rangle$, where Σ is a finite alphabet, S is a finite set of states, $S_0 \subseteq S$ is a set of starting states, $\rho : S \times \Sigma \rightarrow 2^S$ is a (nondeterministic) transition function, and α is an acceptance condition. A Büchi acceptance condition is a set $F \subseteq S$.

A *run* r of A over a infinite word $w = a_0 a_1 \ldots$, is a sequence $s_0, s_1, \ldots$, where $s_0 \in S_0$ and $s_i \in \rho(s_{i-1}, a_{i-1})$, for all $i \geq 1$. The run r satisfies a Büchi condition F if there is some state in F that repeats infinitely often in r, i.e., for some $s \in F$ there are infinitely many i's such that $s_i = s$. The run r is *accepting* if it satisfies the acceptance condition, and the infinite word w is *accepted* by A if there is an accepting run of A over w. The set of infinite words accepted by A is denoted $L_\omega(A)$. The automaton A is said to be *nonempty* if $L_\omega(A) \neq \emptyset$.

Theorem 2.1

1. [EL85, EL87] *The nonemptiness problem for Büchi automata is solvable in linear time.*

2. [VW94] *The nonemptiness problem for Büchi automata is solvable in nondeterministic logarithmic space.*

The following theorem establishes the correspondence between LTL and Büchi automata.

Theorem 2.2 [VW94] *Given an LTL formula φ, one can build a Büchi automaton $A_\varphi = \langle 2^{AP}, S, S_0, \rho, F\rangle$, where $|S| \leq 2^{O(|\varphi|)}$, such that $L_\omega(A_\varphi)$ is exactly the set of computations satisfying the formula φ.*

2.3 Alternating Tree Automata

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 $A = \langle\Sigma, S, S_0, \rho, F\rangle$, where Σ is a finite alphabet, S is a finite set of states, $S_0 \subseteq S$ is a set of starting states, $\rho : S \times \Sigma \rightarrow 2^{S^2}$ is a transition function, and F is an acceptance condition. A *run* r of A over a Σ-labeled infinite binary tree T is an S-labeled infinite binary tree where $r(\epsilon) \in S_0$ (i.e., the root of the tree is labeled by a state in S_0), and if for every node x we have that $\langle r(x0), r(x1)\rangle \in \rho(r(x), T(x))$ (i.e., the labeling of r obeys the transition function).

Here, the transition function ρ maps an automaton state $s \in S$ and an input letter $a \in \Sigma$ 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 s and is reading a node x labeled by a letter a, it proceeds by first choosing a pair $\langle s_1, s_2\rangle \in \rho(s, a)$ and then splitting into two copies. One copy enters the state s_1 and proceeds to the node $x \cdot 0$ (the left successor of x), and the other copy enters the state s_2 and proceeds to the node $x \cdot 1$ (the right successor of x).

For a given set D, let $\mathcal{B}^+(D \times S)$ be the set of positive Boolean formulas over $D \times S$ (i.e., Boolean formulas built from elements in $D \times S$ using $\wedge$ and $\vee$), where we also allow the formulas **true** and **false** and, as usual, $\wedge$ has precedence over $\vee$. We can represent ρ using $\mathcal{B}^+(\{0, 1\} \times S)$. For example, $\rho(s, a) = \{\langle s_1, s_2\rangle, \langle s_3, s_1\rangle\}$ can be written as $\rho(s, a) = (0, s_1) \wedge (1, s_2) \vee (0, s_3) \wedge (1, s_1)$.

In nondeterministic tree automata, each conjunction in ρ has exactly one element associated with each direction. In alternating automata on binary trees, $\rho(s, a)$ can be an arbitrary formula from $\mathcal{B}^+(\{0, 1\} \times S)$. We can have, for instance, a transition

$$\rho(s, a) = \big((0, s_1) \wedge (0, s_2)\big) \vee \big((0, s_2) \wedge (1, s_2) \wedge (1, s_3)\big).$$

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. More generally, we consider also trees whose nodes can have different branching degrees. Formally, an *alternating tree automaton* is a tuple $A = \langle\Sigma, \mathcal{D}, S, s_0, \rho, F\rangle$

where Σ is the input alphabet, $\mathcal{D} \subset \mathbb{N}$ is a finite set of of possible branching degrees, S is a finite set of states, $s_0 \in S$ is an initial state, F specifies the acceptance condition, and $\rho : S \times \Sigma \times \mathcal{D} \to \mathcal{B}^+(\mathbb{N} \times S)$ is the transition function. We require that for every $k \in \mathcal{D}$ we have $\rho(s, a, k) \in \mathcal{B}^+(\{0, \ldots, k-1\} \times S)$. In other words, a transition depends on the branching degree and specifies a matching Boolean transition.

A run r of an alternating automaton A on a tree T is a tree where the root is labeled by s_0 and every other node is labeled by an element of $\mathbb{N}^* \times S$. Each node of r corresponds to a node of T. A node in r, labeled by (x, s) describes a copy of the automaton that reads the node x of T while in state s. 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. For formal details see [BVW94, MS87].

In [BVW94], Bernholtz et al. introduce *hesitant alternating automata* (HAA). The acceptance condition of an HAA consists of a pair $F = \langle G, B \rangle$ of subsets of S. Also, in an HAA there is a partition of S into disjoint sets S_i, and there is a partial order $\leq$ on the collection of the S_i's such that for every $s \in S_i$ and $s' \in S_j$ for which s' occurs in $\rho(s, a, k)$, for some $a \in \Sigma$ and $k \in \mathcal{D}$ we have $S_j \leq S_i$. Thus, transitions from a state in S_i lead to states in either the same S_i or a lower one. It follows that every infinite path of a run of an HAA ultimately gets "trapped" within some S_i. In addition, each set S_i is classified as either *transient*, *existential*, or *universal*, such that for each set S_i and for all $s \in S_i$, $a \in \Sigma$, and $k \in \mathcal{D}$, the following hold:

1. If S_i is a transient set, then $\rho(s, a, k)$ contains no elements of S_i.

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

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

It follows that every infinite path of a run gets trapped within some either an existential or a universal set S_i. The path then satisfies an acceptance condition $F = \langle G, B \rangle$ if and only if either S_i is an existential set and $inf(\pi) \cap G \neq \emptyset$, or S_i is an universal set and $inf(\pi) \cap B = \emptyset$. The set of infinite trees accepted by A is denoted $L_\omega(A)$. The number of sets in the partition of S is defined as the *alternation depth* of A.

The following theorem establishes the correspondence between CTL and HAA.

Theorem 2.3 [BVW94] *Given a CTL formula ψ and a finite set $\mathcal{D} \subset \mathbb{N}$, we can construct an HAA $A_{\mathcal{D},\psi} = \langle 2^{AP}, \mathcal{D}, S, s_0, \rho, F \rangle$, where $|S| \leq O(|\psi|)$, such that $L_\omega(A_{\mathcal{D},\psi})$ is exactly the set of computation trees, with branching degree in $\mathcal{D}$, satisfying ψ.*

As is shown in [BVW94], CTL model checking can be reduced to the 1-letter nonemptiness problem for HAA (i.e., the nonemptiness problem over 1-letter alphabets).

Theorem 2.4 [BVW94]

1. *The 1-letter nonemptiness problem for HAA is solvable in TIME($O(n)$), where n is the size of the input automaton.*

2. *The 1-letter nonemptiness problem for HAA is solvable in NSPACE($O(m \log n)$), where n is the size of the input automaton and m is the alternation depth of the input automaton.*

3 Verification of Modules

3.1 Modular Specification of Modules

A *module* $M = \langle W, R, w^0, L \rangle$ consists of a set W of states, a total transition relation $R \subseteq W \times W$, an initial state w^0, and a labeling $L : W \to 2^{AP}$ that maps each state to a set of atomic propositions that hold in this state. Note that there is no difference between our definitions of programs and modules; both are transition systems. The difference is in how programs and modules are viewed. A program is viewed as a complete description of a system. In contrast, a module is viewed as a component of a system. Since we are focusing here on the verification of a single module, we can ignore the issue of how modules are composed (for example, this can be done by introducing edge labels in addition to our state labels).

A module M is specified by an assume-guarantee pair $\langle \varphi, \psi \rangle$, where φ is an LTL formula and ψ is a CTL formula. The idea is that M satisfies the specification $\langle \varphi, \psi \rangle$ if ψ is satisfied by the computation tree that consists of all computations of M that satisfy φ. We now formalize this intuition.

We proceed in two steps. First we convert M to a *tree module* M^t. A *partial path* χ in M is a finite sequence $w_0, w_1, \ldots, w_k$, where w_0 is w^0 (the initial state of M), and $(w_i, w_{i+1}) \in R$ for $0 \leq i < k$. We define $L(\chi)$ 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 $(\chi, \chi') \in R^t$, where $\chi, \chi' \in ppath(M)$, if χ' R-extends χ. The tree module M^t is now defined as $M^t = (ppath(M), R^t, w^0, L)$. Note that in M^t is indeed a tree; every state has a unique predecessor.

We will shortly define what it means for a state χ in M^t to satisfy a CTL formula θ with respect to φ, denoted $M^t, \chi \models_\varphi \theta$. We then say that M satisfies ψ with respect to φ, denoted $M \models_\varphi \psi$, if $M^t, w^0 \models_\varphi \psi$.

It remains to define satisfaction in M^t. We follow the framework of generalized temporal semantics in [Eme83]. A *path* π in M^t is an infinite sequence $\chi_0, \chi_1, \ldots$ where $(\chi_i, \chi_{i+1}) \in R^t$ for all $i \geq 0$. We say that the path π is *anchored* if χ_0 is w^0. With each path $\pi = \chi_0, \chi_1, \ldots$, we associate a computation $L(\pi) = L(\chi_0), L(\chi_1), \ldots$. We say that π is a φ-path if $L(\pi), 0 \models \varphi$.

- $M, \chi \models_\varphi p$ for $p \in AP$ if $p \in L(\chi)$.
- $M, \chi \models_\varphi \neg\psi'$ if $M, \chi \not\models_\varphi \psi'$.
- $M, \chi \models_\varphi \psi_1 \wedge \psi_2$ if $M, \chi \models_\varphi \psi_1$ and $M, \chi \models_\varphi \psi_2$.
- $M, \chi \models_\varphi EX\psi'$ if there exists an anchored φ-path $\pi = \chi_0, \ldots, \chi_i, \chi_{i+1}, \ldots$ of M^t such that $\chi_i = \chi$ and $M, \chi_{i+1} \models_\varphi \psi'$.
- $M, \chi \models_\varphi AX\psi'$ if for every anchored φ-path $\pi = \chi_0, \ldots, \chi_i, \chi_{i+1}, \ldots$ of M^t such that $\chi_i = \chi$ we have $M, \chi_{i+1} \models_\varphi \psi'$.
- $M, \chi \models_\varphi E\psi_1 U \psi_2$ if there exists a φ-path $\pi = \chi_0, \ldots, \chi_i, \chi_{i+1}, \ldots$ of M^t such that $\chi_i = \chi$ and for some $k \geq 0$ we have that $M, \chi_{i+k} \models_\varphi \psi_2$ and $M, \chi_{i+j} \models_\varphi \psi_1$ for $0 \leq j < k$.
- $M, \chi \models_\varphi A\psi_1 U \psi_2$ if for every φ-path $\pi = \chi_0, \ldots, \chi_i, \chi_{i+1}, \ldots$ of M^t such that $\chi_i = \chi$ there is some $k \geq 0$ such that $M, \chi_{i+k} \models_\varphi \psi_2$ and $M, \chi_{i+j} \models_\varphi \psi_1$ for $0 \leq j < k$.
- $M, \chi \models_\varphi E\psi_1 \tilde{U} \psi_2$ if there exists a φ-path $\pi = \chi_0, \ldots, \chi_i, \chi_{i+1}, \ldots$ of M^t such that $\chi_i = \chi$ and for all $j \geq 0$ such that $M, \chi_j \not\models_\varphi \psi_2$, there exists $0 \leq k < j$ such that $M, \chi_k \models_\varphi \psi_1$.
- $M, \chi \models_\varphi A\psi_1 \tilde{U} \psi_2$ if for every φ-path $\pi = \chi_0, \ldots, \chi_i, \chi_{i+1}, \ldots$ of M^t such that $\chi_i = \chi$ we have that for all $j \geq 0$ such that $M, \chi_j \not\models_\varphi \psi_2$, there exists $0 \leq k < j$ such that $M, \chi_k \models_\varphi \psi_1$.

Note that this definition is essentially the standard definition of the semantics of CTL, except that we define the truth of formulas on the nodes of the computation tree M^t of M and we relativize the quantifiers to the φ-paths of M^t.[1]

The *linear-branching model-checking problem* is to decide, given a finite module M, an LTL formula φ, and a CTL formula ψ, whether $M \models_\varphi \psi$.

3.2 An Upper Bound

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 that 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.

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

Theorem 3.1 *There is an algorithm that decides whether a module M satisfies a CTL formula ψ with respect to an LTL formula φ in time $nm2^{2^{O(l)}}$, where n is the size of M, m is the size of ψ, and l is the size of φ.*

Proof sketch: Let $M = \langle W, R, w^0, L\rangle$. Not all paths in M are φ-paths. Thus, when the model-checking algorithm tries to satisfy a CTL formula $E\theta$ 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 M^t (defined in Section 3.1), 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 $A_\varphi = \langle 2^{AP}, S, S_0, \rho, F\rangle$, where $|S| \leq 2^{O(|\varphi|)}$, such that $L_\omega(A_\varphi)$ is exactly the set of computations satisfying the formula φ. We then apply to A_φ 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 A_φ with M. That is, we define a module $M_\varphi = \langle W_\varphi, R_\varphi, w^0_\varphi, L_\varphi\rangle$ as follows:

- $W_\varphi = W \times 2^S$,
- $w^0_\varphi = \langle w^0, S_0\rangle$,
- $L_\varphi(\langle w, X\rangle) = \langle L(w), X\rangle$,
- $(\langle u, X\rangle, \langle v, Y\rangle \in R_\varphi)$ if $(u, v) \in R$ 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 module*.

We now define the semantics of CTL formulas on φ-annotated modules. In the following definition, A^X_φ denotes the automaton $\langle 2^{AP}, S, X, \rho, F\rangle$, i.e., it is A_φ started from the state set $X \subseteq S$.

- $M_\varphi, \langle u, X\rangle \models_\varphi p$ for $p \in AP$ if $p \in L(u)$.
- $M_\varphi, \langle u, X\rangle \models_\varphi \neg\psi'$ if $M_\varphi, \langle u, X\rangle \not\models_\varphi \psi'$.
- $M_\varphi, \langle u, X\rangle \models_\varphi \psi_1 \wedge \psi_2$ if $M_\varphi, \langle u, X\rangle \models_\varphi \psi_1$ and $M_\varphi, \langle u, X\rangle \models_\varphi \psi_2$.

- $M_\varphi, \langle u, X\rangle \models_\varphi EX\psi'$ if there exists a path $\pi = \langle u_0, X_0\rangle, \ldots$ in M_φ such that $u_0 = u$, $X_0 = X$, $L(\pi)$ is accepted by A_φ^X, and $M_\varphi, \langle u_1, X_1\rangle \models_\varphi \psi'$.
- $M_\varphi, \langle u, X\rangle \models_\varphi AX\psi'$ if there every path $\pi = \langle u_0, X_0\rangle, \ldots$ in M_φ such that $u_0 = u$, $X_0 = X$, and $L(\pi)$ is accepted by A_φ^X, we have that $M_\varphi, \langle u_1, X_1\rangle \models_\varphi \psi'$.
- $M_\varphi, \langle u, X\rangle \models_\varphi E\psi_1 U\psi_2$ if there exists a path $\pi = \langle u_0, X_0\rangle, \ldots$ in M_φ such that $u_0 = u$, $X_0 = X$, $L(\pi)$ is accepted by A_φ^X, and for some $k \geq 0$ we have that $M, \langle u_k, X_k\rangle \models_\varphi \psi_2$ and $M, \langle u_j, X_j\rangle \models_\varphi \psi_1$ for $0 \leq j < k$.
- $M_\varphi, \langle u, X\rangle \models_\varphi A\psi_1 U\psi_2$ if for every path $\pi = \langle u_0, X_0\rangle, \ldots$ in M_φ such that $u_0 = u$, $X_0 = X$, and $L(\pi)$ is accepted by A_φ^X, there is some $k \geq 0$ such that $M, \langle u_k, X_k\rangle \models_\varphi \psi_2$ and $M, \langle u_j, X_j\rangle \models_\varphi \psi_1$ for $0 \leq j < k$.
- $M_\varphi, \langle u, X\rangle \models_\varphi E\psi_1 \tilde{U}\psi_2$ if there exists a path $\pi = \langle u_0, X_0\rangle, \ldots$ in M_φ such that $u_0 = u$, $X_0 = X$, $L(\pi)$ is accepted by A_φ^X, and for all $i \geq 0$ such that $M, \langle u_i, X_i\rangle \not\models \psi_2$, there exists $0 \leq j < i$ such that $M, \langle u_j, X_j\rangle \models \psi_1$.
- $M_\varphi, \langle u, X\rangle \models_\varphi A\psi_1 \tilde{U}\psi_2$ if for every path $\pi = \langle u_0, X_0\rangle, \ldots$ in M_φ such that $u_0 = u$, $X_0 = X$, and $L(\pi)$ is accepted by A_φ^X, we have that for all $i \geq 0$ such that $M, \langle u_i, X_i\rangle \not\models \psi_2$, there exists $0 \leq j < i$ such that $M, \langle u_j, X_j\rangle \models \psi_1$.

We now claim that M_φ indeed retains all the relevant information from M^t.

Claim. $M \models_\varphi \psi$ if and only if $M_\varphi, w_\varphi^0 \models_\varphi \psi$.

We now prove the analogue of Theorem 2.3 for φ-annotated modules. That is, we show that given an LTL formula φ, a CTL formula ψ, and a finite set $\mathcal{D} \subset \mathbb{N}$, we can construct an HAA $A_{\mathcal{D},\varphi,\psi}$ with state set Q, where $|Q| \leq |\psi| \cdot 2^{O(|\varphi|)}$, such that $L_\omega(A_{\mathcal{D},\varphi,\psi})$ is exactly the set of φ-annotated computation trees, with branching degree in $\mathcal{D}$, satisfying ψ. We will take $\mathcal{D}$ to be the set of branching degrees in M.

The *closure* $cl(\psi)$ of a CTL formula φ is the set of all subformulas of φ (including φ itself). Clearly, for every φ, the size of $cl(\varphi)$ is at most $|\varphi|$. Let $E\tilde{U}(\psi)$ all subformula of ψ of the form $E\varphi_1 \tilde{U}\varphi_2$, and let $AU(\psi)$ all subformula of ψ of the form $A\varphi_1 U\varphi_2$. Recall that S is the state set of the automaton A_φ. 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.

$A_{\mathcal{D},\varphi,\psi}$ is the HAA

$$\langle 2^{AP} \times 2^{2^S}, \mathcal{D}, Q, \psi, \delta, \langle G, B\rangle\rangle,$$

where the alphabet is $2^{AP} \times 2^{2^S}$, $Q = cl(\psi) \cup S \cup \overline{S} \cup (E\tilde{U}(\psi) \times S) \cup (AU(\psi) \times \overline{S})$. Intuitively, the automaton is

- in state $\psi' \in cl(\psi)$ when it tries to satisfy ψ',
- in state $s \in S$ when it tries to find a path that is accepted by $A_\varphi^{\{s\}}$,
- in state $\overline{s} \in \overline{S}$ when it tries to show that there is no path accepted by $A_\varphi^{\{s\}}$,
- in state $\langle E\psi_1 \tilde{U}\psi_2, s\rangle$ when it tries to find a path that satisfies $\psi_1 \tilde{U}\psi_2$ and is accepted by $A_\varphi^{\{s\}}$, and
- in state $\langle A\psi_1 U\psi_2, \overline{s}\rangle$ when it tries to show that every path either satisfies $\psi_1 U\psi_2$ or is not accepted by $A_\varphi^{\{s\}}$.

Note that $|Q| \leq |\psi| \cdot 2^{O(|\varphi|)}$. For the acceptance condition we have $G = F \cup (E\tilde{U}(\psi) \times F)$ and $B = \overline{F} \cup (AU(\psi) \times \overline{F})$.

It remains to define the transition function δ. Consider first states in S:

- $\delta(s, \langle a, X\rangle, k) = \bigvee_{c=0}^{k-1}(c, \rho(s, a))$.

Consider now states in $\overline{S}$:

- $\delta(\overline{s}, \langle a, X\rangle, k) = \bigwedge_{c=0}^{k-1}(c, \overline{\rho(s, a)})$.

Consider now states in $cl(\psi)$. In the following definition, the dual $\overline{\theta}$ of a formula θ in $\mathcal{B}^+(\mathbb{N} \times (S \cup \overline{S}))$ is obtained from θ by switching $\vee$ and $\wedge$, and switching s and $\overline{s}$; e.g., $\overline{\overline{s_1} \vee (s_2 \wedge s_3)}$ is $s \wedge (\overline{s_2} \vee \overline{s_3})$.

- $\delta(p, \langle a, X\rangle, k) = \textbf{true}$ if $p \in a$.
- $\delta(p, \langle a, X\rangle, k) = \textbf{false}$ if $p \notin a$.
- $\delta(\neg p, \langle a, X\rangle, k) = \textbf{true}$ if $p \notin a$.
- $\delta(\neg p, \langle a, X\rangle, k) = \textbf{false}$ if $p \in a$.
- $\delta(\varphi_1 \wedge \varphi_2, \langle a, X\rangle, k) =$ $\delta(\varphi_1, \langle a, X\rangle, k) \wedge \delta(\varphi_2, \langle a, X\rangle, k)$.
- $\delta(\varphi_1 \vee \varphi_2, \langle a, X\rangle, k) =$ $\delta(\varphi_1, \langle a, X\rangle, k) \vee \delta(\varphi_2, \langle a, X\rangle, k)$.
- $\delta(EX\varphi_2, \langle a, X\rangle, k) =$ $\bigvee_{c=0}^{k-1}\big((c, \varphi_2) \wedge \bigvee_{s \in \rho(X,a)}(c, s)\big)$.
- $\delta(AX\varphi_2, \langle a, X\rangle, k) =$ $\bigwedge_{c=0}^{k-1}\big((c, \varphi_2) \vee \bigwedge_{s \in \rho(X,a)}(c, \overline{s})\big)$.
- $\delta(E\varphi_1 U\varphi_2, \langle a, X\rangle, k) =$ $\big(\delta(\varphi_2, \langle a, X\rangle, k) \wedge \bigvee_{s \in X} \delta(s, \langle a, X\rangle)\big)$ $\vee \big(\delta(\varphi_1, \langle a, X\rangle, k) \wedge \bigvee_{c=0}^{k-1}(c, E\varphi_1 U\varphi_2)\big)$.
- $\delta(A\varphi_1 U\varphi_2, a, k) =$ $\bigwedge_{s \in X} \delta(\langle A\varphi_1 U\varphi_2, \overline{s}\rangle, \langle a, X\rangle)$.
- $\delta(E\varphi_1 \tilde{U}\varphi_2, \langle a, X\rangle, k) =$ $\bigvee_{s \in X} \delta(\langle E\varphi_1 \tilde{U}\varphi_2, s\rangle, \langle a, X\rangle)$.

- $\delta(A\varphi_1\tilde{U}\varphi_2, \langle a, X\rangle, k) =$
  $\bigwedge_{s\in X} \overline{\delta(s, \langle a, X\rangle)} \vee \big(\delta(\varphi_2, a, k) \wedge$
  $(\delta(\varphi_1, \langle a, X\rangle, k) \vee \bigwedge_{c=0}^{k-1}(c, A\varphi_1\tilde{U}\varphi_2)\big).$

Finally, consider states in $E\tilde{U}(\psi) \times S \cup AU(\psi) \times \overline{S}$:

- $\delta(\langle E\varphi_1\tilde{U}\varphi_2, s\rangle, \langle a, X\rangle, k) =$
  $\delta(\varphi_2, a, k) \wedge$
  $\big((\delta(\varphi_1, \langle a, X\rangle, k) \wedge \delta(s, \langle a, X\rangle, k)) \vee$
  $\bigvee_{c=0}^{k-1} \bigvee_{s'\in\rho(s,a)}(c, \langle E\varphi_1\tilde{U}\varphi_2, s'\rangle)\big).$
- $\delta(\langle A\varphi_1 U\varphi_2, \overline{s}\rangle, a, k) =$
  $\delta(\varphi_2, \langle a, X\rangle, k) \vee$
  $\big(\delta(\varphi_1, \langle a, X\rangle, k) \vee \delta(\overline{s}, \langle a, X\rangle, k) \vee$
  $\bigwedge_{c=0}^{k-1} \bigwedge_{s'\in\rho(s,a}(c, \langle A\varphi_1 U\varphi_2, \overline{s'}\rangle)\big).$

For example, the clause for a formula $E\varphi_1 U\varphi_2$ says that the formula holds in a node x labeled by $\langle a, X\rangle$ if either φ_2 holds in x and there is a path accepted by A_φ^X that starts at x or φ_1 holds at x and $E\varphi_1 U\varphi_2$ holds in a successor of x. The acceptance condition guarantees that satisfaction of φ_2 cannot be postponed for ever. As another example, the clause for the a formula $E\varphi_1\tilde{U}\varphi_2$ says that the formula holds in a node x if $\varphi_1\tilde{U}\varphi_2$ is satisfied along a path that is accepted by A_φ^X.

To show that $A_{\mathcal{D},\varphi,\psi}$ is an HAA, we define a partition of Q into disjoint sets and a partial order over the sets. Each subformula $\theta \in cl(\psi)$ constitutes a (singleton) set $\{\theta\}$ in the partition, with $\{\theta_1\} < \{\theta_2\}$ if θ_1 is a proper subformula of θ_2. Also, for formulas $\theta_1 \in E\tilde{U}(\psi)$ and $\theta_2 \in AU(\psi)$, the sets $\{\langle\theta_1, s\rangle : s \in S\}$ and $\{\langle\theta_2, \overline{s}\rangle : s \in S\}$ are sets of the partition, with $\{\langle\theta_1, s\rangle : s \in S\} < \{\theta_1\}$, $\{\langle\theta_2, \overline{s}\rangle : s \in S\} < \{\theta_2\}$, $\{\theta\} < \{\langle\theta_1, s\rangle : s \in S\}$ if θ is a proper subformula of θ_1, and $\{\theta\} < \{\langle\theta_2, \overline{s}\rangle : s \in S\}$ if θ is a proper subformula of θ_2. Finally, each of the set S and $\overline{S}$ is a minimal set of the partition.

We now have to check that $A_{\mathcal{D},\varphi,\psi}$ accepts M_φ^t. As is shown in [BVW94], this can be done by taking the product of $A_{\mathcal{D},\varphi,\psi}$ with M_φ. The result is an HAA $A_{M,\varphi,\psi}$ over a 1-letter alphabet, where the size of $A_{M,\varphi,\psi}$ $nm2^{2^{O(l)}}$. $A_{M,\varphi,\psi}$ is nonempty precisely when $A_{\mathcal{D},\varphi,\psi}$ accepts M_φ^t. The claim then follows, since the 1-letter emptiness for HAAs is solvable in linear running time (Theorem 2.4). $\square$

The algorithm of Theorem 3.1 not only has doubly exponential running time, it also may use doubly exponential amount of space. This makes this algorithm even less practical than the doubly exponential running time would indicate. It turns out, however, that the space complexity of the algorithm need not be that high.

Theorem 3.2 *There is an algorithm that decides whether a module M satisfies a CTL formula ψ with respect to an LTL formula φ using space $O\big((m(\log n + \log m + 2^{O(l)}))^2\big)$, where n is the size of M, m is the size of ψ, and l is the size of φ.*

Proof sketch: By Theorem 2.4, the 1-letter nonemptiness problem for bounded-alternation HAA is solvable in NSPACE($\alpha \log \beta$), where β is the size of the input automaton and α is the alternation depth of the input automaton.

In the proof of Theorem we reduced the linear-branching model checking problem to 1-letter nonemptiness of the HAA $A_{M,\varphi,\psi}$, whose size is $nm2^{2^{O(l)}}$. It is easy to see that the alternation depth of $A_{M,\varphi,\psi}$ is $O(|psi|)$. Thus, we can check nonemptiness in NSPACE($O(m(\log n + \log m + 2^{O(l)}))$). The claimed bound then follows by the Savitch Theorem [Sav70]. $\square$

Remark. The algorithms above can be extended to handle branching temporal guarantees in CTL*. In this case we get time complexity of $n2^{O(m)}2^{2^{O(l)}}$ and space complexity of $O\big((m(\log n + m + 2^{O(l)}))^2\big)$.

3.3 A Lower Bound

The upper bound given in Theorem 3.2 is exponential in the length of the linear temporal assumption. Can we do better? We now show that the exponential space complexity is inherent by proving that the problem is EXPSPACE-hard even if the module and the CTL guarantee are of bounded size.

Theorem 3.3 *The linear-branching model-checking problem is EXPSPACE-complete.*

Proof sketch: The upper bound is given in Theorem 3.2.

The lower bound is proven by reduction from the problem whether an exponential-space machine N accepts an input word x. By taking N to be a machine that accepts an EXPSPACE-complete language, we can fix N and vary only x. Our encoding is somewhat similar to the encoding used in [VS85] to prove that satisfiability of CTL* is 2EXPTIME-complete. There is a basic difficulty, however, in adapting the encoding of [VS85]. The formula constructed there uses $O(|x|)$ atomic propositions, and essentially all $2^{O(|x|)}$ truth assignments to these propositions occur in the intended model of this formula. In our case, the intended model is a subtree of the computation tree of the given module, which means that the number of truth assignments that occur in the intended model is at most linear in the size of the input.

In our reduction here, the set AP of atomic propositions depends only on the machine N and is independent of $|x|$. The module $M = \langle W, R, w^0, L\rangle$ has an almost trivial structure. We take $W = 2^{AP}$, $R = 2^{AP} \times 2^{AP}$, and $L(X) = X$. That is, the state set consist of all truth assignments to AP with the obvious labeling function and all transitions are possible. One of the truth assignments is chosen as the initial state w^0. Thus, for a fixed machine N, the module M is fixed.

The computation tree of M obviously contains all possible sequences of truth assignments to AP starting from w^0. For a proposition Q and a node u of the computation tree, we let $Q(u)$ denote the truth value of Q at u (0 or 1). Let $n = |x|$. We divide every such sequence to blocks of length n. Every such block corresponds to a single tape cell of the machine N. Consider a block $u_1, \ldots, u_n$, which corresponds to a cell c. We use an atomic proposition B to mark the end of the block; that is, B should fail on $u_1, \ldots, u_{n-1}$ and hold on u_n. This will be enforced by the linear temporal assumption φ. I.e., φ contains a conjunct

$$\neg B \wedge X(\neg B \wedge X(\neg B \ldots \wedge XB)) \wedge G(B \Leftrightarrow X^n B)$$

We have atomic propositions $S_1, \ldots, S_d$. The bit-vector $S_1(u_n), \ldots, S_d(u_n)$ encodes the symbol written on the cell c. (The number d depends on the size of the working alphabet of N.)

Since the symbol at c is encoded at the node u_n, why do we need a block of length n to encode a single cell? The block also encodes the location of the cell c on the tape. That location is a number between 0 and $2^n - 1$. We have an atomic proposition C, called *counter*, and we let $C(u_1), \ldots, C(u_n)$ encode the location of c.

Thus, a sequence of 2^n such blocks corresponds to a configuration of N. The value of the counters along this sequence should go from 0 to $2^n - 1$, and then start again from 0. This will be enforced by the linear temporal assumption φ. (To keep the size of φ be $O(n)$, we need also an atomic proposition D that acts as a "carry" bit.) An atomic proposition F marks the last node of a configuration, that is, F holds in a node u_n of a block $u_1, \ldots, u_n$ iff C holds on all nodes in the block.

The difficult part in the reduction is in guaranteeing that the sequence of configurations indeed forms a legal computation of N. To enforce this, we have to compare tape locations in two successive configurations. If these configurations are $c_0, \ldots, c_i, \ldots, c_{2^n-1}$ and $d_0, \ldots, d_i, \ldots, d_{2^n-1}$, then we need to relate d_i to c_{i-1}, c_i, c_{i+1}. To be able to do such a comparison, it is not sufficient to consider one path in the module. While one "real" path represents the computation of N, we need to introduce many auxiliary paths, as in [VS85]. An atomic proposition I will hold on the real path and fail on the auxiliary paths. The existence of one real path and many auxiliary paths is enforced by the branching temporal guarantee ψ:

$$EG(I \wedge EXEG\neg I).$$

Thus, ψ requires that there be a real path along which I holds and every point of which can be extended to an auxiliary path on which I fails. The formula φ will force the first $n2^n$ nodes of the real path to represent the initial configuration of N with input x.

The reason for having auxiliary paths is that we can "mark" a position on such path in LTL. The LTL formula $I \wedge XG\neg I$, holds on an auxiliary path at the point where the real path becomes an auxiliary path. Thus, using the auxiliary paths we can "point" to any node on the real path. The formula φ can now compare the value of the counter at this node to the value of the counter in another node. If these values are the same and the nodes are in successive configurations (i.e., there's only one true occurrence of F between them), then these nodes represent corresponding tape cells and φ can enforce a desired relationship between them. □

3.4 ∀CTL

Theorem 3.3 indicates that linear-branching model checking in its full generality is rather intractable. What is the implication of this result on modular verification?

In modular verification, one uses assertions of the form $\langle\varphi\rangle M\langle\psi\rangle$, where φ is an LTL formula and ψ is a CTL formula, to assert that ψ holds in the computation tree that consists of all computations of the program that satisfy the linear temporal formula φ, i.e., $M \models_\varphi \psi$. Assume-guarantee assertions are used to verify properties of compositions $M_1\|M_2$ of modules. The computation tree of $M_1\|M_2$, however, may not contains all the computations in the computations trees of M_1 and M_2; some computations might be eliminated by the composition. Thus, in order to perform modular verification, one has to restrict attention to properties that have the *upward preservation property*, i.e., once they are satisfied in a module, they are satisfied also in every system that contains this module.

For this reason, it is argued in [DDGJ89, Jos89, DGG93, GL94], that in the context of modular verification in it advantageous to use only *universal* branching temporal logic, i.e., branching temporal logic without existential path quantifiers. Thus, 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 upward preservartion property.

Under this restriction [Jos87a, Jos87b, Jos89], assume-guarantee assertions are used in modular proof rules of the following form:

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

where $br(\varphi)$ is a branching version (it is an ∀CTL formula) of the LTL formula φ; see above references for details.

We now observe that the exponential space complexity in the size of the linear temporal assumption of the linear-branching model checking problem holds even if we restrict the branching temporal guarantee to be specified in ∀CTL, the universal fragment of CTL. In ∀CTL, every A quantifier is in the scope of an even number of negations and every E quantifiers is under an odd number of negations. Put otherwise, a ∀CTL formula in positive-normal form contains only A quantifiers. It is known that ∀CTL can express properties that are not expressible in LTL; for example, the ∀CTL formula $AFAGp$ is not expressible in LTL [CD88].

It is not hard to see that ∀CTL is easier to reason about than CTL. For example, while the satisfiability problem for CTL is EXPTIME-complete [FL79], it can be shown that the satisfiability problem for ∀CTL is PSPACE-complete [BV95]. Nevertheless, Theorem 3.3 shows that even for branching temporal guarantees in ∀CTL, the complexity of linear-branching model checking is EXPSPACE-hard. Clearly, $M \models_\varphi \psi$ iff $M \not\models_\varphi \neg\psi$. Since the reduction in the proof of Theorem 3.3 uses a purely existential temporal guarantee, the lower bound applies also to ∀CTL.

4 Concluding Remarks

The results of the previous section indicate that linear-branching model-checking for general 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 Section 3 are worst-case bounds. In practice, the automaton A_φ need not be exponential in the size of φ, and the subset construction in the proof of Theorem 3.1 need not yield an exponential blowup in the size of A_φ; heuristics can be employed to avoid unnecessary states in A_φ and unnecessary states in M_φ. If the size of the linear assumption φ is not too large and the doubly exponential blowup is avoided, then our algorithm 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]. In the full paper we will provide an automata-theoretic explanation for Josko's complexity results. 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 M_φ. In particular, under this restriction we can show a space bound of $O\big((m(\log n+\log m+l))^2\big)$ (i.e., polynomial rather than exponential in the size of 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 consider modular verification (furthermore, it is shown in [BV95] that modular model checking is computationally hard even when both assumptions and guarantees are given in ∀CTL). This suggests that the tradeoff between CTL and LTL is not a simple tradeoff between complexity and expressiveness.

Acknowledgements

I am grateul to Martin Abadi, Orna Kupferman, and Pierre Wolper for their helpful comments on a previous 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.

[ASSSV94] 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.

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method and Philos. Sci. 1960*, pages 1–12, Stanford, 1962. Stanford University Press.

[BV95] O. Bernholtz and M.Y. Vardi. On the complexity of branching modular model checking. March 1995.

[BVW94] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *Computer Aided Verification, Proc. 6th Int. Workshop*, Stanford, California, June 1994. Lecture Notes in Computer Science, Springer-Verlag. 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*, pages 428–437. Lecture Notes in Computer Science, 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, January 1986.

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

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

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

[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.

[Eme83] E.A. Emerson. Alternative semantics for temporal logics. *Theoretical Computer Science*, 26:121–130, 1983.

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

[FL79] M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and Systems Sciences*, 18:194–211, 1979.

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

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

[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.

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

[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 Int'l 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.

[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.

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

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

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

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

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