Relating Linear and Branching Model Checking

O. Kupferman
UC Berkeley
EECS Department, Berkeley CA 94720-1770, U.S.A.,
Email: orna@eecs.berkeley.edu, URL: www.eecs.berkeley.edu/~orna

M.Y. Vardi
Rice University
Department of Computer Science, Houston, TX 77005-1892, U.S.A.
Email: vardi@cs.rice.edu, URL: www.cs.rice.edu/~vardi

Abstract
The difference in the complexity of branching and linear model checking has been viewed as an argument in favor of the branching paradigm. In particular, the computational advantage of CTL model checking over LTL model checking makes CTL a popular choice, leading to efficient model-checking tools for this logic. Can we use these tools in order to verify linear properties? In this paper we relate branching and linear model checking. With each LTL formula ψ, we associate a CTL formula ψ_A that is obtained from ψ by preceding each temporal operator by the universal path quantifier A. We first describe a number of attempts to utilize the tight syntactic relation between ψ and ψ_A in order to use CTL model-checking tools in the process of checking the formula ψ. Neither attempt, however, suggests a method that is guaranteed to perform better than usual LTL model checkers. We then claim that, in practice, LTL model checkers perform nicely on formulas with equivalences of CTL. In fact, they often proceed essentially as the ones for 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 [CE81, 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. In temporal-logic *model checking*, we verify the correctness of a finite-state program with respect to a desired behavior by checking whether a labeled transition system that models the program satisfies a temporal logic formula that specifies this behavior (for a survey, see [CGL93]).

Two possible views regarding the nature of time induce two types of temporal logics [Lam80]. In *linear* temporal logics, time is treated as if each moment in time has a unique possible future. Thus, linear temporal logic formulas are interpreted over linear sequences and we regard them as describing a behavior of a single computation of a program. In *branching* temporal logics, each moment in time may split into various possible futures. Accordingly, the structures over which branching temporal logic formulas are interpreted can be viewed as infinite computation trees, each describing the behavior of the possible computations of a nondeterministic program.

In the linear temporal logic LTL, formulas are composed from the set of atomic propositions using the usual Boolean connectives as well as the temporal operators G ("always"), F ("eventually"), X ("next"), and U ("until"). For example, the LTL formula $\psi = G(req \rightarrow F grant)$ refers to the atomic propositions req and $grant$ and is true in a computation iff every state in the computation in which request is valid is followed by some state in the future in which grant is valid. The branching temporal logic CTL* augments LTL by the path quantifiers E ("there exists a computation") and A ("for all computations"). For example, the CTL* formula $\varphi = AG(req \rightarrow EF grant)$ is true in a computation tree iff in all its computations, every state in which request is valid branches to at least one computation in which grant is eventually valid. The branching temporal logic CTL is a fragement of CTL* in which every temporal operator is preceded by a path quantifier. For example, while φ above is a CTL formula, the CTL* formula $AG(req \rightarrow F grant)$ is not a CTL formula. Finally, the branching temporal logic $\forall$CTL is a fragement of CTL in which only universal path quantification is allowed. Formally, one introduces a positive normal form for CTL formulas so that $\forall$CTL is the syntactic fragement of CTL that does not employ the path quantifier E. For an LTL formula ψ, we denote by ψ_A the $\forall$CTL formula obtained from ψ by preceding each temporal operator by the path quantifier A. For example, ψ_A for ψ above is $AG(req \rightarrow AF grant)$.

Given a transition system M and a linear temporal logic formula ψ, the model-checking problem for M and ψ is to decide whether ψ holds in all the computations of M. When ψ is a branching temporal logic formula, the problem is to decide whether ψ holds in the computation tree of M. The complexity of model checking for both linear and branching temporal logics is well understood: suppose we are given a transition system of size n and a temporal logic formula of size m. For the branching temporal logic CTL, model-checking algorithms run in time $O(nm)$ [CES86], while, for the linear temporal logic LTL, model-checking algorithms run in time $n2^{O(m)}$ [LP85]. Since LTL model checking 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. In particular, the computational advantage of CTL model checking over LTL model checking makes CTL a popular choice, leading to efficient model-checking tools for this logic. Can we use these tools in order to verify linear properties? A straightforward relation between LTL and CTL model checking follows from the fact that LTL model checking can be reduced to the language-containment problem [VW86], which itself can be reduced to searching for fair paths, and hence to CTL model checking [VW86, CDK93]. Such an approach, however, involves the definition of a new transition system whose size is exponential in the length of the LTL formula. As such, it does not enjoy the computational advantage of CTL. The real challenge is to relate CTL and LTL model checking in a practical way, one that would enable us to use CTL model-checking tools in order to perform efficient model checking on some fragement of LTL.

A straightforward approach is as follows: given a transition system M and an LTL formula ψ to be checked with respect to M, try to translate the CTL* formula $A\psi$ to an equivalent CTL formula φ, and then check M with respect to φ. This approach has two drawbacks. First, the problem of deciding whether $A\psi$ has an equivalent CTL formula φ is still open, and so, of course, is the harder problem of constructing φ in cases it exists. Second, even when such φ exists, it may be substantially longer than ψ, making the whole effort useless. A partial success for this approach is presented in [KG96, Sch97], which identify fragments of CTL* that can be easily translated in to CTL.

In this paper we study a more modest approach: instead looking for some equivalent CTL formula to $A\psi$, we restrict ourselves to the specific candidate ψ_A. This approach is weaker, in the sense that $A\psi$ may have an equivalent CTL formula and still not be equivalent to ψ_A. For example, $\psi = (Xp) \vee (Xq)$ is not equivalent to $\psi_A = (AXp) \vee (AXq)$, yet is equivalent to the CTL formula $AX(p \vee q)$. Nevertheless, this approach does not suffer from the drawbacks mentioned above: we know how to decide whether $A\psi$ and ψ_A are equivalent, the construction of ψ_A from ψ is straightforward, and the length of ψ_A is at most double the length of ψ. In addition, $A\psi$ is always weaker than ψ_A (that is, the implication $\psi_A \rightarrow A\psi$ is valid for all ψ), which would turn out to be useful for some heuristics.

We first study the problem of deciding whether $A\psi$ and ψ_A are equivalent. As $A\psi$ and ψ_A are CTL* formulas, the known 2EXPTIME upper bound for CTL* satisfiability [ES84, EJ88] suggests an obvious 2EXPTIME upper bound for the above equivalence problem. Moreover, as $A\psi$ and ψ_A are $\forall$CTL* formulas, the problem can be solved in EXPSPACE [KV95]. We present here an EXPSPACE algorithm that is much simpler than the one in [KV95], as it avoids Safra's complicated co-determinization construction [Saf89] that is used in the definition of the maximal models described there. Our hopes to use the tight syntactic relation between $A\psi$ and ψ_A in order to improve this bound were not fulfilled. We conjecture that the problem can be solved in polynomial space, which would match our lower bound.

So, the first attempt, of checking the sensitivity of the LTL specification to branching, does not seem very appealing. We then turn to consider the second operand of the model-checking problem, namely the transition system M. We say that a transition system M is *indifferent* iff for all LTL formulas ψ, we have that M satisfies $A\psi$ iff M satisfies ψ_A. Indifference is a helpful property: in order to model check either an LTL or an $\forall$CTL formula in an indifferent transition system, one can use both LTL and CTL model-checking tools. We characterize transition systems that are indifferent and show that the problem of determining whether a transition system is indifferent is NLOGSPACE-complete. Hence, it is easy check a given transition system for having this property. On the other hand, our characterization shows that it is unlikely for a practical transition system to be indifferent. Thus, this second attempt does not seem very appealing as well.

In the context of model checking, however, we do not care about the equivalence of $A\psi$ and ψ_A with respect to all transition systems, neither about the indifference of M with respect to all formulas. Rather, we care only about the equivalence of $A\psi$ and ψ_A in the transition system M we wish to check. This can be checked by solving the model-checking problem $M \models A\psi \leftrightarrow \psi_A$. The model-checking problem $M \models A\psi \leftrightarrow \varphi$ for ψ in LTL and φ in $\forall$CTL can be solved in PSPACE [EL85]. What about the special case where $\varphi = \psi_A$? One could hope that the tight syntactic relation between $A\psi$ and ψ_A would make it easier. We show that the problem of deciding whether $A\psi$ and ψ_A are equivalent with respect to a given transition system M is PSPACE-complete, thus it is not easier than performing LTL model checking of ψ with respect to M.

There is, however, a method to avoid the equivalence check of $A\psi$ and ψ_A and still use the relation between the two formulas in order to benefit from CTL model-checking tools. The idea is very similar to an idea used in the model checker SPIN [Hol97]. In order to use its partial-order reductions, SPIN has to check its specifications for closure under stuttering. SPIN avoids the expensive check but still employs its partial-order reductions. For that, SPIN first closes the given specification under stuttering, then it model checks (with partial-order reductions) the closed specification, and it uses the relation between the closed specification and the original one in order to deduce either the satisfaction of the original specification or its not being closed under stuttering [HK96]. We follow the same lines: instead of checking $A\psi$, verification starts by checking M with respect to ψ_A. If M satisfies ψ_A, we deduce that M satisfies $A\psi$ as well. If M does not satisfy ψ_A, we use the counterexample that the model checker provides in order to determine whether it is the case that M satisfies $A\psi$ or whether it is the case that $A\psi$ and ψ_A are not equivalent. Thus, sometimes we do end up performing LTL model checking, yet the transition system to be checked then is the counterexample, which is typically much smaller than the original transition system.

So far, we considered several ways for using ψ_A in the process of ψ model checking. Unfortunately, neither ways suggests a method that is guaranteed to perform better than usual LTL model checkers. Our motivation, recall, is the computational advantage of CTL over LTL. This advantage, however, refers to worst-case complexity. In the second part of the paper we claim that, in practice, standard LTL model checkers perform nicely on most formulas ψ for which $A\psi$ and ψ_A are equivalent. In fact, model checking of ψ often proceeds essentially as model checking of ψ_A. Thus, in practice, there is no need to relate linear and branching model checking and the question whether $A\psi$ is equivalent to ψ_A is not that important. In order to prove our claim, we compare the behavior of the bottom-up CTL model-checking algorithm of [CES86] with the behavior of the automata-based LTL model-checking algorithm of [VW86]. We show that most LTL formulas ψ for which $A\psi$ is equivalent to ψ_A induce a special type of automata. Solving the nonemptiness problem of these automata involves essentially the same steps required for solving the model-checking problem for ψ_A. Intuitively, both model checkers proceed according to the semantics of the formula and are insensitive to the syntax in which it is given.

2 PRELIMINARIES

2.1 Temporal logics

The logic *CTL** is a branching temporal logic. Formulas of CTL* are defined with respect to a set AP of atomic propositions and describe the evolution of these propositions. We present here a positive normal form for CTL* formulas where negation may be applied only to atomic propositions. In this form, a path quantifier, E ("for some path") or A ("for all paths"), can prefix an assertion built from the set of atomic proposition using the usual Boolean connectives and the temporal operators X ("next time"), U ("until"), and $\tilde{U}$ ("duality of until"). There are two types of formulas in CTL*: *state formulas*, whose satisfaction is related to a specific state, and *path formulas*, whose satisfaction is related to a specific path. Formally, let AP be a set of atomic proposition names. A CTL* state formula is either:

- **true**, **false**, p or $\neg p$, for $p \in AP$.
- $\psi \vee \varphi$ or $\psi \wedge \varphi$ where ψ and φ are CTL* state formulas.
- $E\psi$ or $A\psi$, where ψ is a CTL* path formula.

A CTL* path formula is either:

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

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

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

The logic *LTL* is a linear temporal logic. Given a set AP, an LTL formula is defined as follows:

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

Thus, each LTL formula ψ corresponds to a CTL* formula $A\psi$ in which ψ is a path formula with no path quantifiers. Hence, LTL can be viewed as a fragment of CTL*. For an LTL formula ψ, we denote by ψ_A the $\forall$CTL formula obtained from ψ by preceding each temporal operator by the universal path quantifier A. For example, $(Xp \vee Xq)_A = AXp \vee AXq$. The formula ψ_E is defined similarly for the existential path quantifier E. We denote the size of a formula φ by $|\varphi|$ and we use the following abbreviations in writing formulas: $\rightarrow$ and $\leftrightarrow$, interpreted in the usual way, $F\psi = \mathbf{true}U\psi$ ("eventually"), and $G\psi = \neg F \neg \psi$ ("always").

We define the semantics of CTL* and its sublanguages with respect to *fair Rabin modules* (*modules*, for short). A module $M = \langle AP, W, W_0, R, L, \alpha \rangle$ consists of a set AP of atomic propositions, a set W of states, a set $W_0 \subseteq W$ of initial states, a total transition relation $R \subseteq W \times W$ (i.e., for every state $w \in W$, there exists at least w' with $R(w, w')$), a labeling function $L : W \rightarrow 2^{AP}$, and a Rabin fairness condition $\alpha \subseteq 2^W \times 2^W$, which defines a subset of W^ω. The module M is of *branching degree 1* iff for every state $w \in W$, there exists a single w' with $R(w, w')$. A *computation* of a module is an infinite sequence of states, $\pi = w_0, w_1, \ldots$ such that for every $i \geq 0$, we have that $R(w_i, w_{i+1})$. When $w_0 \in W_0$, we say that π is an *initialized computation*. For a computation π, let $Inf(\pi)$ denote the set of states that repeat infinitely often in π. That is,

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

A computation of M is *fair* iff it satisfies the fairness condition. Thus, if the fairness condition is $\alpha = \{\langle G_1, B_1 \rangle, \ldots, \langle G_k, B_k \rangle\}$, then π is fair iff there exists $1 \leq i \leq k$ for which $Inf(\pi) \cap G_i \neq \emptyset$ and $Inf(\pi) \cap B_i = \emptyset$. In other words, iff π visits G_i infinitely often and visits B_i only finitely often. Each computation $\pi = w_0, w_1, \ldots$ of M induces a *trace* $L(\pi) = L(w_0), L(w_1), \ldots$ in $(2^{AP})^\omega$. We sometimes refer to the *language*, $\mathcal{L}(M)$, of a module M, meaning the set of traces induced by M's initialized fair computations. We say that M is *nonempty* if $\mathcal{L}(M) \neq \emptyset$. For a module M and a state $w \in W$, we use M^w to denote the module obtained from M by taking the set of initial states to be the singleton $\{w\}$. For simplicity, we assume that each initial state $w_0 \in W_0$ is the first state of at least one initialized fair computation. Thus, M^{w_0} is nonempty.

Assuming an agreed module M, we use $w \models \varphi$ to indicate that a state formula φ holds at state w, and we use $\pi \models \varphi$ to indicate that a path formula φ holds at path π. The formal definition of the relation $\models$ can be found in [Eme90]. A module M satisfies a formula ψ iff ψ holds in all the initial states of M. The *model-checking problem* is to decide, given M and ψ, whether $M \models \psi$.

2.2 Simulation relation and maximal models

In [GL94], Grumberg and Long define an order relation between modules that captures what it means for a module M' to have "more behaviors" than a module M. The definition in [GL94] extend previous definitions [Mil71], which relate modules with no fairness conditions. Let $M = \langle AP, W, R, W_0, L, \alpha \rangle$ and $M' = \langle AP, W', R', W_0', L', \alpha' \rangle$ be two modules and let w and w' be states in W and W', respectively. A relation $H \subseteq W \times W'$ is a *simulation relation* from $\langle M, w \rangle$ to $\langle M', w' \rangle$ iff the following conditions hold:

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

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

(2.1) $L(s) \cap AP' = L(s')$.

(2.2) For every fair computation $\pi = s_0, s_1, \ldots$ in M, with $s_0 = s$, there exists a fair computation $\pi' = s'_0, s'_1, \ldots$ in M', with $s'_0 = s'$, such that for all $i \geq 0$, we have $H(s_i, s'_i)$.

A simulation relation H is a *simulation from M to M'* iff for every $w \in W_0$, there exists $w' \in W'_0$ such that $H(w, w')$. If there exists a simulation from M to M', we say that M' *simulates* M and we write $M \leq M'$. Intuitively, it means that the module M' has more behaviors than the module M. In fact, every possible behavior of M is also a possible behavior of M'. The following theorem is proved in [GL94] for modules with a Streett fairness condition. The proof is independent of the type of the fairness condition being used and it holds also for fair Rabin modules.

Theorem 2.1 *For every M and M' such that $M \leq M'$, and for every $\forall CTL^\star$ formula φ, we have that $M' \models \varphi$ 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_φ is a maximal model of φ if $M_\varphi \models \varphi$ and for every module M we have that $M \leq M_\varphi$ if $M \models \varphi$. Note that by the preceding theorem, if $M \leq M_\varphi$, then $M \models \varphi$. Thus, M_φ is a maximal model for φ if for every module M, we have that $M \leq M_\varphi$ iff $M \models \varphi$.

Let φ and ψ be two $\forall$CTL* formulas. 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:

Theorem 2.2 [GL94] *Let φ and ψ be $\forall CTL^\star$ formulas, and let M_φ be a maximal model of φ. Then φ implies ψ iff $M_\varphi \models \psi$.*

2.3 Büchi word automata

A *Büchi automaton over infinite words* is $\mathcal{A} = \langle \Sigma, Q, Q_0, \delta, F \rangle$, where Σ is the input alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function, and $F \subseteq Q$ is an acceptance condition. Intuitively, $\delta(q, \sigma)$ is the set of states that $\mathcal{A}$ can move into when it is in state q and it reads the letter σ. Given an input infinite word $\rho = \sigma_0, \sigma_1, \sigma_2, \ldots$ in Σ^ω, a *run* of $\mathcal{A}$ on ρ is an infinite sequence $r = q_0, q_1, q_2, \ldots$ in Q^ω where $q_0 \in Q_0$ (i.e., the run starts in one of the initial states) and for every $i \geq 0$, we have $q_{i+1} \in \delta(q_i, \sigma_i)$ (i.e., the run obeys the transition function). Each run r induces a set *Inf*(r) of states that r visits infinitely often. As Q is finite, it is guaranteed that *Inf*$(r) \neq \emptyset$. The run r *accepts* ρ iff *Inf*$(r) \cap F \neq \emptyset$. Note that $\mathcal{A}$ can have many runs on ρ. An automaton $\mathcal{A}$ accepts an input word ρ iff there exists a run r of $\mathcal{A}$ on ρ such that r accepts ρ. The *language* of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of infinite words that $\mathcal{A}$ accepts.

Theorem 2.3 [VW94] *Given an LTL formula ψ, there exists a Büchi automaton $\mathcal{A}_\psi$ of size $2^{O(|\psi|)}$, such that $\mathcal{L}(\mathcal{A}_\psi)$ is exactly the set of computations satisfying ψ.*

3 RELATING ψ AND ψ_A

In this section we describe a number of attempts to utilize the tight syntactic relation between ψ and ψ_A in order to relate linear and branching model checking.

In other words, we are given a module M and an LTL formula ψ, and we try to make use of ψ_A and to benefit from CTL model-checking tools in the process of deciding whether M satisfies ψ. A natural thing to start with is to check the equivalence of $A\psi$ and ψ_A. For an LTL formula ψ, we say that ψ is *branchable* iff $A\psi$ and ψ_A are equivalent. Clearly, if ψ is branchable, then checking whether M satisfies ψ can be reduced to checking whether M satisfies ψ_A. We first claim that modules with no branches can not distinguish between $A\psi$ and ψ_A.

Lemma 3.1 *For a module M of branching degree 1 and an LTL formula ψ, we have that $M \models A\psi$ iff $M \models \psi_A$.*

Proof. Let $M = \langle AP, W, W_0, R, L, \alpha \rangle$. Since M is of branching degree 1, it contains a single initialized fair computation π. We can associate with each state $w \in W$ a suffix π^w of π that starts in w. It is easy to prove, by induction on the structure of ψ, that for every state w, we have that $\pi^w \models \psi$ iff $w \models \psi_A$. It follows that $M \models A\psi$ iff $M \models \psi_A$. $\square$

We now claim that $A\psi$ is weaker than ψ_A:

Lemma 3.2 *For every LTL formula ψ, the implication $\psi_A \rightarrow A\psi$ is valid.*

Proof. Consider a module M that satisfies ψ_A. We show that all the initialized fair computations of M satisfy ψ. Then, by the semantics of $\forall$CTL*, we have that M satisfies $A\psi$. Let π be an initialized fair computation of M. The computation π can be viewed as a module of branching degree 1. Since π is simulated by M, it satisfies ψ_A. Hence, by Lemma 3.1 that π satisfies ψ as well. $\square$

By Lemma 3.2, deciding whether ψ is branchable can be reduced to checking the implication $A\psi \rightarrow \psi_A$. This implication is not valid for all LTL formulas ψ. For example, the formula $AFGp$ does not imply the formula $AFAGp$. We now solve this implication problem. As $A\psi$ and ψ_A are CTL* formulas, the known 2EXPTIME upper bound for CTL* satisfiability [ES84, EJ88] suggests an obvious 2EXPTIME upper bound for the problem. Moreover, as $A\psi$ and ψ_A are $\forall$CTL* formulas, the problem can be solved in EXPSPACE [KV95]. Can we hope for a better bound? This at first seems unlikely: the implication problem $\psi \rightarrow \varphi$ is EXPSPACE-hard already for ψ in LTL and φ in $\forall$CTL [KV95]. Here, however, we handle the special case where $\varphi = \psi_A$. Hopefully, the tight syntactic relation between ψ and ψ_A would enable a more efficient check. We conjecture that the problem can be solved in polynomial space, which matches our lower bound. The algorithm we present below requires exponential space, and in the worst case shows no improvement over the known EXPSPACE upper bound. Yet, it is much simpler than the algorithm in [KV95], as it avoids Safra's complicated co-determinization construction that is used in the definition of the maximal models described there.

Theorem 3.3 *For an LTL formula ψ, checking $A\psi \rightarrow \psi_A$ is in EXPSPACE and is PSPACE-hard.*

Proof. We start with the upper bound. By Theorem 2.2, checking the implication $A\psi \rightarrow \psi_A$ can be reduced to model checking of ψ_A in the maximal model $M_{A\psi}$ of $A\psi$. To complete the reduction of implication to model checking, we have to describe the construction of maximal models for LTL formulas. A construction of maximal models for $\forall$CTL* formulas is described in [KV95]. The restriction to LTL formulas enables us to come with a much simpler construction.

Given an LTL formula ψ, let $\mathcal{A}_\psi = \langle 2^{AP}, Q, Q_0, \delta, F\rangle$ be the Büchi automaton that corresponds to ψ (see Theorem 2.3). Following the construction in [VW94], each state s of $\mathcal{A}_\psi$ is a set of subformulas of ψ. When in state s, the automaton $\mathcal{A}_\psi$ accepts exactly all the computations that satisfy all the formulas in s. In particular, for every state s, the set $\delta(s,\sigma)$ is not empty iff $s \cap AP = \sigma$. Defining the maximal model M_ψ, we apply to $\mathcal{A}_\psi$ the classical *subset construction* of [RS59], that is, we extend δ to a mapping from $2^Q \times 2^{AP}$ to 2^Q where $\delta(S,a) = \{q \in Q : q \in \delta(s,a)$ for some $s \in S\}$. We also extends δ to words in $(2^{AP})^*$, where $\delta(S,\varepsilon) = S$ and for $w \in (2^{AP})^*$ and $a \in 2^{AP}$, we have $\delta(S, w\cdot a) = \delta(\delta(S,w),a)$. We define $M_\psi = \langle AP, W, W_0, R, L, \alpha\rangle$ as follows.

- $W \subseteq 2^Q \times Q \times \{0,1\}$ is such that $\langle S,s,b\rangle \in W$ iff $s \in S$. Intuitively, the 2^Q-element S follows the subset construction. Since the Q-element s is always a member of S, then whenever we move from the state $\langle S,s,b\rangle$ to a state $\langle S',s',b'\rangle$, it may be the case that the update of the Q-element is consistent with δ (i.e., s' is a successor of s in $\mathcal{A}_\psi$), and it may also be the case that the update of the Q-element is not consistent with δ (i.e., s' is a successor of some other state in S). The Boolean flag b distinguishes between the two cases.
- $W_0 = \{\langle Q_0, q_0, 0\rangle : q_0 \in Q_0\}$.
- $L(\langle S,s,b\rangle) = s \cap AP$. Thus, the label of each state is the set of atomic propositions that hold in its Q-element.
- $R(\langle S,s,b\rangle, \langle S',s',b'\rangle)$ if $\delta(S, L(\langle S,s,b\rangle)) = S'$, $s' \in S'$, and one of the following holds
  - $s' \in \delta(s, L(\langle S,s,b\rangle))$ and $b' = 0$, or
  - $b' = 1$.
- $\alpha = \{\langle 2^Q \times F \times \{0,1\}, 2^Q \times Q \times \{1\}\rangle\}$. That is, a computation of M_ψ is fair iff its projection on the Q-element visits F infinitely often and its projection on the Boolean flag visits 1 only finitely often. Accordingly, a computation is fair if it has a suffix in which the Q-element describes an accepting run of $\mathcal{A}_\psi$.

We say that a computation $\langle S_0,s_0,b_0\rangle, \langle S_1,s_1,b_1\rangle, \langle S_2,s_2,b_2\rangle, \ldots$ of M_ψ is *stable* iff $b_i = 0$ for all $i \geq 0$. Note that a computation is stable only if its Q-element describes a run of $\mathcal{A}_\psi$.

We now prove that $M_\psi \models A\psi$ and that for all modules M, we have $M \models A\psi$ only if $M \leq M_\psi$. We first prove that $M_\psi \models A\psi$. Recall that each state s of $\mathcal{A}_\psi$ is a set of subformulas of ψ. When in state s, the automaton $\mathcal{A}_\psi$ accepts exactly all the computations that satisfy all the formulas in s. Therefore, for every state $\langle S,s,b\rangle$ in W and for every formula $\varphi \in s$, all the stable fair computations that leave $\langle S,s,b\rangle$ satisfy φ. Indeed, such computations describe an accepting run of $\mathcal{A}_\psi$ from the state s. Since each fair computation of M_ψ eventually becomes stable, we can associate with each initialized fair computation $\pi = \langle S_0,s_0,b_0\rangle, \langle S_1,s_1,b_1\rangle, \langle S_2,s_2,b_2\rangle, \langle S_2,s_3,b_3\rangle, \ldots$, an index k such that for all $j \geq k$, we have $b_j = 0$. Since as long as a computation of M_ψ is not stable, its 2^Q-element maintains a set of states in Q that $\mathcal{A}_\psi$ may be in, it is easy to prove, by induction on k, that π is accepted by $\mathcal{A}_\psi$.

We now prove that for all modules M, we have $M \models A\psi$ only if $M \leq M_\psi$. Consider a module $M = \langle AP, W_M, W_{M0}, R_M, L_M, \alpha_M \rangle$ and assume that $M \models A\psi$. We show a simulation H from M to M_ψ. We first need some notation. For a state $w \in W_M$, let $\Pi'(w) \subseteq W_M^*$ be the set of prefixes of initialized computations in M that lead to a state from which w is reachable in a single transition. Thus, $\Pi'(w) = \{t_0, \ldots, t_{n-1} : t_0 \in W_{M0}$, for all $0 < i < n, R_M(t_{i-1}, t_i)$, and $R_M(t_{n-1}, w)\}$. We now define, for $w \notin W_{M0}$, the set $\Pi(w) = \Pi'(w)$, and for $w \in W_{M0}$, the set $\Pi(w) = \Pi'(w) \cup \{\varepsilon\}$. Now, let $S(w) \subseteq 2^Q$ be the set of sets of states for which $S \in S(w)$ iff there exists a computation $\rho \in \Pi(w)$ such that $\mathcal{A}_\psi$ may visit any of the states in S after traversing ρ. That is, $S(w) = \{S \subseteq Q :$ there exists $\rho \in \Pi(w)$ such that $S = \delta(Q_0, L_M(\rho))\}$. We can now define the simulation relation H. For a state $w \in W_M$ and a state $\langle S, s, b\rangle \in W$, we have $H(w, \langle S, s, b\rangle)$ iff $L_M(w) = s \cap AP$ and $S \in S(w)$. We prove that H is indeed a simulation. Consider an initial state $w_0 \in W_{M0}$. Recall that for every state s of $\mathcal{A}_\psi$, the set $\delta(s, \sigma)$ is not empty iff $s \cap AP = \sigma$. Since $w_0 \models A\psi$, there exists $q_0 \in Q_0$ such that $q_0 \cap AP = L(w_0)$. Hence, as $Q_0 \in S(w_0)$, the state w_0 is related by H to some initial state of M_ψ. For the second condition, let w and $w' = \langle S, s, b\rangle$ be such that $H(w, w')$. By the definition of H, we have that $L_M(w) = s \cap AP$. Hence, the states w and w' agree on their labels. Let $\pi = w_0, w_1, \ldots$ be a fair computation in M with $w_0 = w$. Since $S \in S(w)$, the set $\Pi(w)$ contains a (possibly empty) prefix $\rho = t_0, t_1, \ldots, t_{n-1}$ such that $S = \delta(Q_0, L_M(\rho))$. The computation $\rho \cdot \pi$ is an initialized fair computation in M. Hence, since $M \models A\psi$, the computation $\rho \cdot \pi$ satisfies ψ, and therefore, $L_M(\rho \cdot \pi)$ is accepted by $\mathcal{A}_\psi$. Let $s_0, s_1, \ldots$ be an accepting run of $\mathcal{A}_\psi$ on $L_M(\rho \cdot \pi)$. By the construction of $\mathcal{A}_\psi$, for every $i \geq 0$, we have that $s_{n+i} \cap AP = L_M(w_i)$. For $i \geq 0$, let S_i be the set of states that $\mathcal{A}_\psi$ may be in after reading the prefix of length i of $L_M(\rho \cdot \pi)$; i.e. $S_i = \delta(Q_0, L_M(\rho \cdot \pi)[1 \ldots i])$. Consider the sequence $\pi' = \langle S_n, s, b\rangle, \langle S_{n+1}, s_{n+1}, b_1\rangle, \langle S_{n+2}, s_{n+2}, b_2\rangle, \langle S_{n+3}, s_{n+3}, b_3\rangle, \ldots$, where $b_1 = 1$ and for all $i \geq 2$ we have $b_i = 0$. Note that $S_n = S$, thus $\langle S_n, s, b\rangle = w'$. We show that π' is a fair computation of M_ψ. First, since for all $i \geq 0$, we have that $s_{n+i} \in S_{n+i}$, all the triples in the sequence are states of M_ψ. Also, since for all $i \geq 0$, we have $\delta(S_{n+i}, w_i) = S_{n+i+1}$, and the for all $i \geq 1$, we have $s_{n+i+1} \in \delta(s_{n+1}, w_i)$, the sequence π' is a computation of M_ψ. Finally, since the sequence $s_0, s_1, \ldots$ is accepted by $\mathcal{A}_\psi$ and visits F infinitely often and since π' has a stable suffix, π' satisfies α and is therefore fair. Now, since, by the definition of H, we have that $H(w_i, \langle S_{n+i}, s_{n+i}, b_i\rangle)$ holds for all $i \geq 1$, we are done.

Since the size of $\mathcal{A}_\psi$ is exponential in ψ, the size of M_ψ is $2^{2^{O(|\psi|)}}$. The model-checking problem for CTL with respect to fair Rabin modules with a single pair can be solved in space that is polynomial in the length of the formula and is poly-logarithmic in the size of the module [KV95]. Moreover, the algorithm described in [KV95] proceeds on-the-fly, without keeping the whole module in memory at once. Therefore, the EXPSPACE upper bound follows.

For the lower bound, we do a reduction from LTL satisfiability, proved to be hard for PSPACE in [SC85]. Consider the LTL formula $\theta = (Xp) \vee (Xq) \vee X((\neg p) \wedge (\neg q))$. It is easy to see that while θ is valid, the $\forall$CTL formula $\theta_A = (AXp) \vee (AXq) \vee AX((\neg p) \wedge (\neg q))$ is not valid. Given an LTL formula φ (we assume that the set of φ's atomic propositions does not contain p and q), let $\psi = \varphi \wedge \theta$. While ψ is equivalent to φ, it is not necessarily true that ψ_A is equivalent to φ_A. We prove that $A\psi \rightarrow \psi_A$ iff φ is not satisfiable.

Assume first that φ is not satisfiable. Then, ψ is not satisfiable as well, making $A\psi \rightarrow \psi_A$ valid. Assume now that φ is satisfiable. Let π be a computation that satisfies φ. Let π_p be the computation obtained from π by labeling its second state with p. Similarly, let π_q be the computation obtained from π by labeling its second state with q. Since π_p and π_q agree on the labeling of their first states, we can arrange them in a module M with a single initial state that branches into two infinite computations. While M satisfies $A\varphi$, and hence also $A\psi$, it does not satisfy θ_A, and hence it does not satisfy ψ_A either. Thus, M does not satisfy $A\psi \rightarrow \psi_A$, and the implication is not valid. $\square$

In practice, the algorithm in Theorem 3.3 requires time that is double exponential in the length of ψ. A naive check of a module M with respect to ψ requires time that is polynomial in the size of M and only exponential in the length of ψ. So, though ψ is usually much smaller than M, checking ψ for being branchable may be more expensive than checking M with respect to ψ, making our first attempt not very appealing. The model-checking problem has two operands: the specification ψ and the module M. Our first attempt concerns the sensitivity of the specification to branching. We now turn to discuss the second operand of the model-checking problem, namely the module M. We say that a module M is *indifferent* iff for every LTL formula ψ, we have $M \models A\psi \rightarrow \psi_A$. Indifference is a helpful property: in order to model check either an LTL or an $\forall$CTL formula in a module that is indifferent, one can use both LTL and CTL model-checking tools. In the theorem below we characterize modules that are indifferent.

Theorem 3.4 *Given a module $M = \langle AP, W, W_0, R, L, \alpha\rangle$, the following are equivalent:*

(1) *M is indifferent.*
(2) *For every initial state $w_0^i \in W_0$, the language of $M_i = \langle AP, W, \{w_0^i\}, R, L, \alpha\rangle$ is a singleton.*

Proof. For the direction (1) $\rightarrow$ (2), assume, by way of contradiction, that there exists a state $w_0^i \in W_0$ such that $\mathcal{L}(M_i)$ is neither empty nor a singleton. Then, there exists $k \geq 0$ such that all the traces in $\mathcal{L}(M_i)$ agree on their prefixes of length k, yet there are at least two traces in $\mathcal{L}(M_i)$ that do not agree on their $k+1$'s letter. Consider the LTL formula $\psi = X^k \bigvee_{\sigma \in 2^{AP}} X\sigma$. While $M \models A\psi$, we have that $M_i \not\models \psi_A$ and thus $M \not\models \psi_A$. It follows that M is not indifferent.

For the direction (2) $\rightarrow$ (1), consider an LTL formula ψ. Let $M_{\psi_A} = \langle AP, W', W'^0, R', L', \alpha'\rangle$ be a maximal model of the formula ψ_A. Assume that $M \models A\psi$. We first prove that $\mathcal{L}(M) \subseteq \mathcal{L}(M_{\psi_A})$. Consider an initialized fair computation π of M. We can regard π as a module of branching degree 1. Since π satisfies ψ then, by Lemma 3.1, it satisfies ψ_A as well. Hence, $\pi \leq M_{\psi_A}$. Since simulation implies trace-containment [GL94], it follows that the trace induced by π is a member of $\mathcal{L}(M_{\psi_A})$. So, $\mathcal{L}(M) \subseteq \mathcal{L}(M_{\psi_A})$. Therefore, for every state $w_0^i \in W_0$, we have $\mathcal{L}(M_i) \subseteq \mathcal{L}(M_{\psi_A})$. Consider a module M_i for which $\mathcal{L}(M_i) \neq \emptyset$. Let $H_i \subseteq W \times W'$ be such that $H_i(w, w')$ iff $L(w) = L'(w')$ and $\mathcal{L}(M_i^w) \subseteq \mathcal{L}(M_{\psi_A}^{w'})$. We prove that the relation H_i is a simulation from M_i to M_{ψ_A}. We first show that w_0^i is related by H_i to some initial state of M_ψ. Recall that $\mathcal{L}(M_i) \subseteq \mathcal{L}(M_{\psi_A})$. Since $\mathcal{L}(M_i)$ is a singleton, there exists an initialized fair computation $\rho = w_0, w_1, \ldots$ in M_ψ such that $\mathcal{L}(M_i) = L'(\rho)$. Hence, $\mathcal{L}(M_i) \subseteq \mathcal{L}(M_{\psi_A}^{w_0})$, and $H_i(w_0^i, w_0)$. Let w and w' be such that $H(w, w')$. First, by the definition of H, the states w and w' agree on their labels. Let $\pi = w_0, w_1, \ldots$ be a fair computation in M_i with $w_0 = w$. Since $\mathcal{L}(M_i^w) \subseteq \mathcal{L}(M_{\psi_A}^{w'})$, there exists a fair computation $\pi' = w'_0, w'_1, \ldots$ in M_{ψ_A} such that $w'_0 = w'$ and $L(\pi) = L'(\pi')$. We prove that for all $j \geq 0$, we have $H(w_j, w'_j)$. First, since $L(\pi) = L'(\pi')$, then clearly $L(w_j) = L'(w'_j)$. In addition, as $\mathcal{L}(M_i)$ is a singleton, it must be that $\mathcal{L}(M_i^{w_j}) \subseteq \mathcal{L}(M_{\psi_A}^{w'_j})$. It follows that $M_i \leq M_{\psi_A}$. Then, the relation $\bigcup_i H_i$ is a simulation relation from M to M_{ψ_A}. Hence, $M \leq M_{\psi_A}$, implying that $M \models \psi_A$, and we are done. $\Box$

We now use the characterization of indifferent modules in order to check a given module M for being indifferent.

Theorem 3.5 *Given a module M, the problem of checking M for indifference is NLOGSPACE-complete.*

Proof. For the upper bound, we show that checking whether M is not indifferent can be done in NLOGSPACE. The result then follows from the NLOGSPACE = co-NLOGSPACE equivalence [Imm88, Sze88]. Let $M = \langle AP, W, W_0, R, L, \alpha \rangle$. By Theorem 3.4, M is not indifferent iff there exists an initial state $w_0^i \in W_0$ such that the language $\mathcal{L}(M_i)$ is not a singleton. The algorithm searches for a witness for M not being indifferent. For that, it guesses an initial state w_0^i and two prefixes $\rho_1 \cdot w_1$ and $\rho_2 \cdot w_2$ of computations (ρ_1 and ρ_2 are sequences in W^*, and w_1 and w_2 are states in W) such that $L(\rho_1) = L(\rho_2)$, $L(w_1) \neq L(w_2)$, and both $\mathcal{L}(M^{w_1})$ and $\mathcal{L}(M^{w_2})$ are nonempty. Membership in NLOGSPACE then follows from the known NLOGSPACE complexity for the reachability problem and the nonemptiness problem for fair Rabin modules. The lower bound follows by an easy reduction from the graph reachability problem [Jon75]. $\Box$

By Theorem 3.5, it is easy to check a given module M for being indifferent. On the other hand, by the characterization in Theorem 3.4, we are unlikely to find indifferent modules in practice. Indeed, branches in M originate from nondeterminism in the program that M models. As such, different branches usually induce different traces. So, the idea of looking at the modules to be checked does not seem very appealing too.

So far, we examined two approaches for relating $A\psi$ and ψ_A. The first checks the equivalence of $A\psi$ and ψ_A with respect to all models, and the second checks the indifference of M with respect to all formulas. Neither approach seems too appealing. In the context of model checking, however, we do not care about the equivalence of $A\psi$ and ψ_A with respect to all models, neither about the indifference of M with respect to all formulas. Rather, we care only about the equivalence of $A\psi$ and ψ_A in the module M we wish to check. This equivalence can be checked by solving the model-checking problem $M \models A\psi \leftrightarrow \psi_A$. The model-checking problem $M \models A\psi \leftrightarrow \varphi$ for ψ in LTL and φ in $\forall$CTL can be solved in PSPACE [EL85]. What about the special case where $\varphi = \psi_A$? One could hope that the tight syntactic relation between ψ and ψ_A would make it easier. In the next theorem we refute this hope.

Theorem 3.6 *Given a module M and an LTL formula ψ, solving the model-checking problem $M \models A\psi \rightarrow \psi_A$ is PSPACE-complete.*

Proof. The upper bound follows from CTL* model-checking complexity [EL85]. For the lower bound, we do a reduction from polynomial space Turing machines. In order to prove that the model-checking problem for LTL is hard for PSPACE, Sistla and Clarke associate with a polynomial space Turing machine T, a module M and an LTL formula ψ, such that $M \models E\psi$ iff T accepts the empty tape [SC85]. The language of M consists of all the possible sequences of configurations of T, and the formula ψ is a conjunction, $\psi = \psi^1 \wedge \psi^2$, of two requirements: ψ^1, which requires the sequence of configurations to encode a legal computation of T on the empty tape, and ψ^2, which requires the sequence of configurations to visit an accepting configuration. Consider the CTL* formula $E\psi^1 \wedge E\psi^2$. The formula holds in M independent of whether T accepts the empty tape. Indeed, it must be that some sequence of configurations encodes a legal computation of T on the empty tape, and it must also be that some sequence of configurations visits an accepting configuration. Consider now the formula $\psi_E = \psi_E^1 \wedge \psi_E^2$. By dualizing Lemma 3.2, we know that $E\psi^1 \rightarrow \psi_E^1$, and $E\psi^2 \rightarrow \psi_E^2$. Hence, the formula ψ_E also holds in M independent of whether T accepts the empty tape. Let $\tilde{\psi}$ be the LTL formula (in our positive normal form) that is equivalent to $\neg\psi$. The $\forall$CTL formula $\tilde{\psi}_A$ holds in exactly these modules that do not satisfy ψ_E. Therefore, by the above, the formula $\tilde{\psi}_A$ does not hold in M independent of whether T accepts the empty tape. Hence, $M \models A\tilde{\psi} \rightarrow \tilde{\psi}_A$ iff $M \not\models A\tilde{\psi}$, which holds iff T accepts the empty tape. $\square$

So, checking whether $A\psi$ and ψ_A are equivalent with respect to a given model is not easier than performing LTL model checking of ψ with respect to M. There is, however, a method to avoid this check, or the $A\psi \rightarrow \psi_A$ implication check, and still use the relation between ψ and ψ_A in order to benefit from CTL model-checking tools. The idea is very similar to an idea used in the model checker Spin. There, Spin avoids the check of its specifications for being closed under stuttering and is still able to use its partial-order reductions [HK96]. Given a module M and an LTL formula ψ, we suggest to proceed as follows (see Figure 1).

Instead of checking $A\psi$, verification starts by checking M with respect to ψ_A. This can be done using a CTL model checker. If $M \models \psi_A$, then, by Lemma 3.2, $M \models \mathcal{A}\psi$ as well, in which case M is correct. If $M \not\models \psi_A$, the model checker provides a counterexample. This counterexample is a witness that M does not satisfy the formula ψ_A. As such, it is some module M' embedded in M (formally, $M' \leq M$) that satisfies the $\exists$CTL formula $\neg\psi_A$. The fact that M' satisfies $\neg\psi_A$ does not imply that M' satisfies $\neg A\psi$ as well. Verification then should continue, by checking M' with respect to $A\psi$. For this check, we do need an LTL model checker. If $M' \not\models A\psi$, then, as $M' \leq M$, it follows that $M \not\models A\psi$ as well, in which case M is not correct. If $M' \models A\psi$, then, as $M' \not\models \psi_A$, the module M' is a witness that ψ is not branchable. The user can then choose to repeat the last step with a

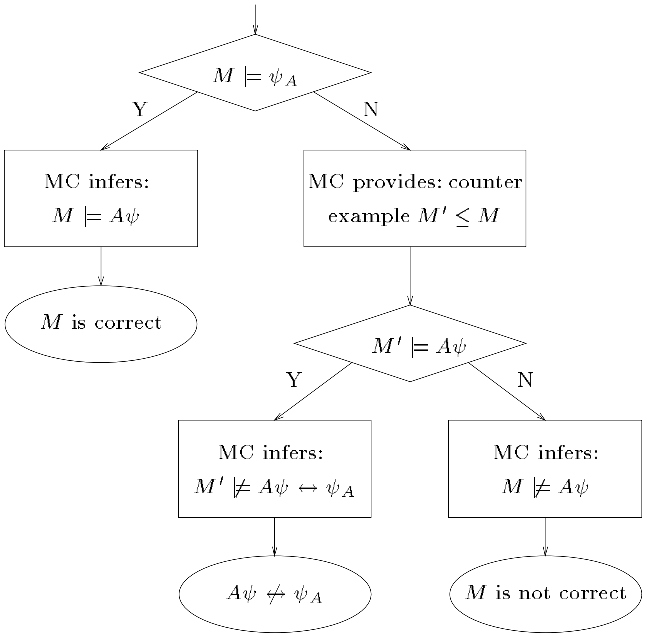

Figure 1 The algorithm.

different counterexample, or to repeat the verification with a revised specification with which he may be luckier*, or to give up and check $A\psi$ in M.

Note that in some cases, the check of M with respect to $A\psi$ proceeds successfully using ψ_A and avoiding LTL model checking even when ψ is not branchable (and even when $A\psi$ has no equivalence of CTL). Our algorithm is not a good one for merely checking whether ψ is branchable. First, when M is proved to be either correct or incorrect, no information on being ψ branchable is obtained. Second, as M is usually much bigger than ψ, executing our algorithm may be more expensive than a checking whether ψ is branchable directly, as described in Theorem3.3. The question we want to answer, however, is not whether ψ is branchable, but whether M is correct with respect to ψ, and we want to do so as efficiently as possible. Note also that even in the unfortunate case, where verification employs an LTL model checker, the module M' that is checked is typically much smaller than the original module M^*. Finally, even in the very unfortunate case, where verification employs an LTL model checker and ends up without a solution to the original model-checking problem, the amount of work being invested is not greater than the one required for the ordinary LTL model-checking algorithm, which can now be performed.

*There are heuristics that make ψ_A likely to be equivalent to $A\psi$ whenever $A\psi$ has a CTL equivalent. For example, one can try to push disjunction inward, thus transforming $Xp \vee Xq$ to $X(p \vee q)$.

*Since it is known that M' does not satisfy ψ_A, one could hope that checking $A\psi$ in M' would be easier than checking $A\psi$ in an arbitrary module. Theorem 3.6, however, refutes this hope.

4 IN PRACTICE

In the previous section, we describe a number of attempts to utilize the relation between ψ and ψ_A in order to use CTL model-checking tools in the process of model checking ψ. Our motivation, recall, is the computational advantage of CTL over LTL. Indeed, while the model-checking problem $M \models \psi$ can be solved in time $O(|M| \cdot |\psi|)$ for ψ in CTL, it requires time $|M| \cdot 2^{O(|\psi|)}$ for ψ in LTL. These bounds, however, refer to worst-case complexity. In this section we claim that, in practice, LTL model checkers perform nicely on branchable LTL formulas. In fact, they often proceed essentially as the ones for CTL.

We cannot state formally that if an LTL formula ψ is branchable, then checking ψ can be done efficiently. The reason is that ψ may contain some subformulas that are not branchable and that are hard to be checked. For example, ψ may be of the form $\theta \wedge (\varphi \vee \neg\varphi)$ where θ is branchable and φ is an LTL formula whose model checking requires polynomial space. Though ψ is equivalent to θ, LTL model checkers may not detect this equivalence and may not check ψ efficiently.

What we do claim is that for a natural branchable LTL specification ψ, LTL model checkers that check ψ do essentially the same work as CTL model checkers that check ψ_A. For simplicity, we consider modules M with no fairness conditions; i.e., modules in which all the computations are fair. As the "representative" CTL model checker we take the bottom-up labeling procedure of [CES86]. There, in order to check whether M satisfies φ, we label the states of M by subformulas of φ, starting from the innermost formulas and proceeding such that, when labeling a formula, all its subformulas are already labeled. Labeling subformulas that are atomic propositions, Boolean combinations of other subformulas, or of the form $AX\theta$ or $EX\theta$ is straightforward. Labeling subformulas of the form $A\theta_1 U\theta_2$, $E\theta_1 U\theta_2$, $A\theta_1 \tilde{U}\theta_2$, or $E\theta_1 \tilde{U}\theta_2$ involves a backward reachability test. As the "representative" LTL model checker, we take the automata-based algorithm of [VW86]. There, in order to check whether M satisfies ψ, we construct a Büchi word automaton $\mathcal{A}_{\neg\psi}$ for $\neg\psi$ and check whether the intersection of the language of M with that of $\mathcal{A}_{\neg\psi}$ is nonempty. In practice, the latter check proceeds by checking whether there exists an initial state in the intersection that satisfies CTL formula EG**true**. For the construction of $\mathcal{A}_{\neg\psi}$, we follow the definition in [GPVW95], which improves [VW86] by being demand-driven; that is, the state space of $\mathcal{A}_{\neg\psi}$ is restricted to states that are reachable from the initial state.

We say that a Büchi automaton $\mathcal{A}$ is a *weak automaton* iff there exists a partition of Q into disjoint sets, $Q_1, \ldots, Q_n$, such that for each set Q_i, either $Q_i \subseteq F$, in which case Q_i is an *accepting set*, or $Q_i \cap F = \emptyset$, in which case Q_i is an *rejecting set*. In addition, there exists a partial order 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 set Q_i or a lower one. It follows that every run of a weak automaton ultimately gets trapped within some set Q_i. The run is accepting iff Q_i is accepting. We say that a weak automaton $\mathcal{A}$ with state space Q is a 1-weak automaton iff all the sets in the partition of Q are singletons.

In the examples below we demonstrate that for natural LTL specifications ψ, the automaton $\mathcal{A}_{\neg\psi}$ is a 1-weak automaton.

Example 4.1 Consider the LTL formula $\psi = G(p \rightarrow Xq)$. The formula $\varphi = F(p \wedge X\neg q)$ complements ψ. The automaton that corresponds to φ by Theorem 2.3 is the 1-weak automaton

$$\mathcal{A}_\varphi = \langle 2^{\{p,q\}}, \{\varphi, \neg q, \textbf{true}, \textbf{false}\}, \{\varphi\}, \delta, \{\textbf{true}\}\rangle,$$

where δ is as follows.

- $\delta(\varphi, \{p,q\}) = \delta(\varphi, \{p\}) = \{\varphi, \neg q\}$.
- $\delta(\varphi, \{q\}) = \delta(\varphi, \emptyset) = \{\varphi\}$.
- $\delta(\neg q, \{p,q\}) = \delta(\neg q, \{q\}) = \{\textbf{false}\}$.
- $\delta(\neg q, \{p\}) = \delta(\neg q, \emptyset) = \{\textbf{true}\}$.
- For all $\sigma \in 2^{AP}$, we have $\delta(\textbf{true}, \sigma) = \{\textbf{true}\}$.
- For all $\sigma \in 2^{AP}$, we have $\delta(\textbf{false}, \sigma) = \{\textbf{false}\}$.

Example 4.2 Consider the LTL formula $\psi = G(p \rightarrow Gq)$. The formula $\varphi = F(p \wedge F\neg q)$ complements ψ. The automaton that corresponds to φ by Theorem 2.3 is the 1-weak automaton

$$\mathcal{A}_\varphi = \langle 2^{\{p,q\}}, \{\varphi, F\neg q, \textbf{true}\}, \{\varphi\}, \delta, \{\textbf{true}\}\rangle,$$

where δ is as follows.

- $\delta(\varphi, \{p,q\}) = \{\varphi, F\neg q\}$.
- $\delta(\varphi, \{p\}) = \{\textbf{true}\}$.
- $\delta(\varphi, \{q\}) = \delta(\varphi, \emptyset) = \{\varphi\}$.
- $\delta(F\neg q, \{p,q\}) = \delta(F\neg q, \{q\}) = \{F\neg q\}$.
- $\delta(F\neg q, \{p\}) = \delta(F\neg q, \emptyset) = \{\textbf{true}\}$.
- For all $\sigma \in 2^{AP}$, we have $\delta(\textbf{true}, \sigma) = \{\textbf{true}\}$.

Example 4.3 Consider the LTL formula $\psi = G(p \rightarrow rUq)$. The formula $\varphi = F(p \wedge ((\neg r)\tilde{U}(\neg q))$ complements ψ. The automaton that corresponds to φ by Theorem 2.3 is the 1-weak automaton

$$\mathcal{A}_\varphi = \langle 2^{\{p,r,q\}}, \{\varphi, (\neg r)\tilde{U}(\neg q), \textbf{true}, \textbf{false}\}, \{\varphi\}, \delta, \{(\neg r)\tilde{U}(\neg q), \textbf{true}\}\rangle,$$

where δ is as follows.

- For $\sigma \in 2^{AP}$ with $p \notin \sigma$ or $q \in \sigma$, we have $\delta(\varphi, \sigma) = \{\varphi\}$.
- For $\sigma \in 2^{AP}$ with $p \in \sigma$ and $q \notin \sigma$, we have $\delta(\varphi, \sigma) = \{\varphi, (\neg r)\tilde{U}(\neg q)\}$.
- For $\sigma \in 2^{AP}$ with $r \in \sigma$ and $q \notin \sigma$, we have $\delta((\neg r)\tilde{U}(\neg q), \sigma) = \{(\neg r)\tilde{U}(\neg q)\}$.
- For $\sigma \in 2^{AP}$ with $r \notin \sigma$ and $q \notin \sigma$, we have $\delta((\neg r)\tilde{U}(\neg q), \sigma) = \{\textbf{true}\}$.
- For $\sigma \in 2^{AP}$ with $q \in \sigma$, we have $\delta((\neg r)\tilde{U}(\neg q), \sigma) = \{\textbf{false}\}$.
- For all $\sigma \in 2^{AP}$, we have $\delta(\textbf{true}, \sigma) = \{\textbf{true}\}$.
- For all $\sigma \in 2^{AP}$, we have $\delta(\textbf{false}, \sigma) = \{\textbf{false}\}$.

Note that in all the three examples, each state of $\mathcal{A}_\varphi$ is associated with a single subformula θ of φ. As such, the size of $\mathcal{A}_\varphi$ is linear in the length of φ. Each state θ constitutes a singleton set $\{\theta\}$ in the partition of the state space. Also, the partial order on the sets in the partition is such that $\{\theta_1\} \leq \{\theta_2\}$ iff θ_1 is a subformula of θ_2 (we regard **true** and **false** as subformulas of all formulas).

Consider a module M and a branchable LTL formula ψ. We claim that when $\mathcal{A}_{\neg\psi}$ is a 1-weak automaton as above, then checking the nonemptiness of the product automaton $M \times \mathcal{A}_{\neg\psi}$ proceeds essentially as the bottom-up CTL model-checking algorithm when performed on M and ψ_A. Checking a weak automaton $\mathcal{A}$ for nonemptiness proceeds in phases as follows (see [BVW94]). Let $Q_1 \leq \ldots \leq Q_n$ be the partial order on the sets of $\mathcal{A}$. The algorithm proceeds up the partial order. In phase i, each state q in Q_i is labeled with either 'T' (in the case that the automaton $\mathcal{A}$ with initial state q is nonempty) or 'F' (in the case that $\mathcal{A}$ with initial state q is empty). The label of q depends on labels of states in sets lower than Q_i as well as on the classification of Q_i as either an accepting or a rejecting set. The nonemptiness of $\mathcal{A}$ is then determined according to the labels of its initial states.

Let $M = \langle AP, W, W_0, R, L\rangle$ and let $\mathcal{A}_{\neg\psi} = \langle 2^{AP}, Q, Q_0, \delta, F\rangle$ be a 1-weak automaton. The product automaton $\mathcal{A} = M \times \mathcal{A}_{\neg\psi}$ has state space $W \times Q$ and is a weak automaton. Indeed, if $\{q_1\} \leq \ldots \leq \{q_n\}$ is the partial order on the (singleton) sets of $\mathcal{A}_\varphi$, then $W \times \{q_1\}, \ldots, W \times \{q_n\}$ is a partition of $W \times Q$ into sets, and $W \times \{q_1\} \leq \ldots \leq W \times \{q_n\}$ is a partial order on the sets. When, as in the examples above, each of the states q_i is a subformula θ of φ, then each set Q_i of $\mathcal{A}$ is a copy of M annotated by θ. Recall that when we check $\mathcal{A}$ for nonemptiness, each phase of the algorithm handles a set and labels its states with either 'T' or 'F'. Labeling a state $\langle w, \theta\rangle$ with 'T' means that the state w does not satisfy $A\neg\theta$. Dually, labeling a state $\langle w, \theta\rangle$ with 'F' means that the state w satisfies $A\neg\theta$. When we model check M with respect to ψ_A we follow essentially the same steps: labeling starts from the innermost subformulas and proceeds, as the partial order in $\mathcal{A}_\varphi$, to outer subformulas. The way in which the nonemptiness algorithm decides the label of a state $\langle w, \theta\rangle$ in $\mathcal{A}$ coincides with the way in which the CTL model checker decides whether to label the state w of M with $(\neg\theta)_A$. In both algorithms, the decision for a formula θ that is either a propositional assertion, a Boolean combination of other formulas, or involve the X temporal operator is immediate, and the decision for the more complicated U and $\tilde{U}$ formulas involves a reachability test. It follows that the running time of the two algorithms coincides as well.

We claim that most of the LTL formulas ψ that are used in practice are such that $\mathcal{A}_{\neg\psi}$ is a 1-weak automaton. We conjecture that if an LTL formula ψ is such that all the subformulas φ of ψ are branchable, then $\mathcal{A}_{\neg\psi}$ is a 1-weak automaton. The restriction to branchable subformulas takes care of formulas like $\theta \wedge (\varphi \vee \neg\varphi)$ discussed above, and still leaves us with the bulk of LTL formulas used in practice.

5 CONCLUSIONS

The discussion of the relative merits of linear versus branching temporal logics is almost as early as these paradigms [Lam80]. We mainly refer here to the linear temporal logic LTL and the branching temporal logic CTL. One of the beliefs dominating this discussion has been "while specifying is easier in LTL, model checking is easier for CTL". The attractive complexity of CTL model checking have compensated for its lack of expressive power and branching-time model-checking tools that can handle systems with more than 10^{120} states [McM93, CGL93] are incorporated into industrial development of new designs [BBG+94].

If we examine the history of these issues more closely, we find that things are not that simple. On one hand, the inability of LTL to quantify computations existentially is considered by many a serious drawback [EH86]. In addition, the introduction of fair-CTL [CES86] and of many other extensions to CTL [Lon93, BBG+94, KG96], have made CTL a basis for specification languages that maintain the efficiency of CTL model checking and yet overcome many of its expressiveness limitations. On the other hand, the computational superiority of CTL is also not that clear. For example, comparing the complexities of CTL and LTL model checking for concurrent programs, both are in PSPACE [VW86, BVW94]. As shown in [Var95, KV95], the advantage that CTL enjoys over LTL disappears also when the complexity of modular verification is considered. The distinction between closed an open systems questions the computational superiority of the branching-time paradigm further. As shown in [KV96, KV97], while for LTL specifications, the model-checking paradigm is applicable also for the verification of open systems, this is not true for CTL specifications. The ability of CTL to quantify computations existentially requires a more careful check, in which all possible environments are taken into considerations. The corresponding problem, module checking, is EXPTIME-complete for CTL. Moreover, in the presence of incomplete information, when the system has variables that the environment can not read, the time complexity is exponential in the size of the system.

In this paper, we went back to the original model-checking problem and examined the computational advantage of CTL over LTL there. We first tried to relate LTL and CTL model checking, aiming to utilize CTL model-checking tools in the process of LTL model checking. Neither of our attempts suggested a method that is guaranteed to perform better than usual LTL model checkers. Then, we showed that, in practice, LTL model checkers perform nicely on the bulk of LTL formulas. In fact, they often proceed essentially as the ones for CTL. It follows that linear and branching model checkers do not need to be related: practically, the complexity of checking most LTL formulas is the same as the one required for checking their equivalent CTL formulas. Intuitively, both model checkers proceed according to the semantics of the formula and are insensitive to the syntax in which it is given. Experimental results of LTL and CTL model checking of common specifications support our observation and show no advantage to the branching paradigm [Cla97]. The computational disadvantage of LTL originates from formulas that do not have equivalent CTL formulas and from less common specifications that we rarely meet in practice.

ACKNOWLEDGMENT O. Kupferman is supported in part by ONR YIP award N00014-95-1-0520, by NSF CAREER award CCR-9501708, by NSF grant CCR-9504469, by AFOSR contract F49620-93-1-0056, by ARO MURI grant DAAH-04-96-1-0341, by ARPA grant NAG2-892, and by SRC contract 95-DC-324.036. M.Y. Vardi is supported in part by NSF grants CCR-9628400 and CCR-9700061, and by a grant from the Intel Corporation.

REFERENCES

[BBG+94] I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, and M. Yoeli. Methodology and system for practical formal verification of reactive hardware. In *Proc. 6th CAV*, LNCS 818, pp. 182–193, 1994.

[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.

[CDK93] E. M. Clarke, I. A. Draghicescu, and R. P. Kurshan. A unified approach for showing language containment and equivalence between various types of ω-automata. *IPL* **46**, pp 301–308, 1993.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, LNCS 131, pp. 52–71, 1981.

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

[CGL93] E.M. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In Proc. *REX School*, LNCS 803, pp. 124–175, 1993.

[Cla97] E. Clarke. Private communication, 1997.

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

[EJ88] E.A. Emerson and C. Jutla. The complexity of tree automata and logics of programs. In *Proc. 29th FOCS*, pp. 368–377, 1988.

[EL85] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. In *Proc. 20th POPL*, pp. 84–96, 1985.

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

[ES84] E.A. Emerson and A. P. Sistla. Deciding branching time logic. In *Proc. 16th STOC*, 1984.

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

[GPVW95] R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. A simple on-the-fly automatic verification for linear temporal logic. In *Protocol Specification, Testing, and Verification*, pp. 3–18. Chapman & Hall, 1995.

[HK96] G. Holzmann and O. Kupferman. Not checking for closure under stuttering. In *Proc. 2nd SPIN*, pp 163–169, 1996.

[Hol97] G.J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, 1997.

[Imm88] N. Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17:935–938, 1988.

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

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

[KV95] O. Kupferman and M.Y. Vardi. On the complexity of branching modular model checking. In *Proc. 6th CONCUR*, LNCS 962. pp. 408–422,1995.

[KV96] O. Kupferman and M.Y. Vardi. Module checking. In *Proc. 8th CAV*, LNCS 1102, pp. 75–86, 1996.

[KV97] O. Kupferman and M.Y. Vardi. Module checking revisited. In *Proc. 9th CAV*, LNCS 1254, pp. 36–47, 1997.

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

[Lon93] D.E. Long. *Model checking, abstraction and compositional verification*. PhD thesis, Carnegie-Mellon University, Pittsburgh, 1993.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th POPL*, pp. 97–107, 1985.

[McM93] K.L. McMillan. *Symbolic model checking.* Kluwer Academic Publishers, 1993.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Proc. 2nd IJCAI*, pp. 481–489, 1971.

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

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

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

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

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

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

[Sch97] K. Schneider. CTL and equivalent sublanguages of CTL*. In *Proc. IFIP Conference on Computer Hardware Description Languages and Applications*, pp. 40–59, 1997.

[Sze88] R. Szelepcsinyi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[Var95] M.Y. Vardi. On the complexity of modular model checking. In *Proc. 10th LICS*, pp. 101-111, 1995.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st LICS*, pp. 322–331, 1986.

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

BIOGRAPHIES

Orna Kupferman is a postdoctoral scholar at the department of electrical engineering and computer science in UC Berkeley. She received her Ph.D. from the Technion in 1995.

Moshe Y. Vardi is a Noah Harding Professor and Chair of Computer Science at Rice University. Prior to joining Rice in 1993, he was at the IBM Almaden Research Center, where he managed the Mathematics and Related Computer Science Department. His research interests include database theory, finite-model theory, multi-agent systems, and design specification and verification. Vardi received his Ph.D. from the Hebrew University of Jerusalem in 1981. He is the author and coauthor of about 100 technical papers, as well as a book titled "Reasoning about Knowledge". Vardi is the recipient of 3 IBM Outstanding Innovation Awards. He is an editor of several international journals.