Model Checking and Modular Verification

ORNA GRUMBERG
The Technion

and

DAVID E. LONG
Carnegie-Mellon University

We describe a framework for compositional verification of finite-state processes. The framework is based on two ideas: a subset of the logic CTL for which satisfaction is preserved under composition, and a preorder on structures which captures the relation between a component and a system containing the component. Satisfaction of a formula in the logic corresponds to being below a particular structure (a tableau for the formula) in the preorder. We show how to do assume-guarantee-style reasoning within this framework. Additionally, we demonstrate efficient methods for model checking in the logic and for checking the preorder in several special cases. We have implemented a system based on these methods, and we use it to give a compositional verification of a CPU controller.

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Requirements/Specifications; D.2.2 [**Software Engineering**]: Tools and Techniques—*computer-aided software engineering; decision tables;* D.2.4 [**Software Engineering**]: Program Verification; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*relations among models;* F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*mechanical verification*

General Terms: Algorithms, Design, Languages, Theory, Verification

Additional Key Words and Phrases: Computer-aided verification, CTL, formal verification, model checking, Moore machines, temporal logics

This research was sponsored in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, Ohio 45433-6543 under Contract F33615-90-C-1465, ARPA Order No. 7597, and in part by the National Science Foundation under Contract No. CCR-9005992 and in part by the U.S.-Israeli Binational Science Foundation.
The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. government.
Authors' addresses: O. Grumberg, Computer Science Department, The Technion, Haifa 32000, Israel; email: orna@cs.technion.ac.il; D. E. Long, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213; email: long+@cs.cm.edu.

1. INTRODUCTION

Temporal logic model-checking procedures are useful tools for the verification of finite-state systems [Clarke et al. 1986; Emerson and Lei 1986; Lichtenstein and Pnueli 1985]. However, these procedures have traditionally suffered from the state explosion problem. This problem arises in systems which are composed of many parallel processes; in general, the size of the state space grows exponentially with the number of processes. By introducing symbolic representations for sets of states and transition relations and using a symbolic model-checking procedures, systems with very large state spaces (10^{100} or more states) can be verified [Burch et al. 1990; Coudert et al. 1990]. Further, the time and space requirements with these techniques may in practice be polynomial in the number of *components* of the system. Unfortunately, the symbolic procedures still have limits, and many realistic problems are not tractable due to their size. Thus, we are motivated to search for additional methods of handling the state explosion problem, methods which work well in conjunction with the symbolic techniques.

An obvious method for trying to avoid the state explosion problem is to use the natural decomposition of the system. The goal is to verify properties of individual components, infer that these hold in the complete system, and use them to deduce additional properties of the system. When verifying properties of the components, it may also be necessary to make assumptions about the environment. This approach is exemplified by Pnueli's [1984] assume-guarantee paradigm. A formula in his logic is a triple $\langle \varphi \rangle M \langle \psi \rangle$ where φ and ψ are temporal formulas, and M is a program. The formula is true if whenever M is part of a system satisfying φ, the system must also satisfy ψ. A typical proof shows that $\langle \varphi \rangle M \langle \psi \rangle$ and $\langle true \rangle M' \langle \varphi \rangle$ hold and concludes that $\langle true \rangle M \| \mathrm{M}' \langle \psi \rangle$ is true.

In order to automate this approach, a model checker must have several properties. It must be able to check that a property is true of *all* systems which can be built using a given component. More generally, it must be able to restrict to a given class of environments when doing this check. It must also provide facilities for performing temporal reasoning. Most existing model checkers were not designed to provide these facilities. Instead, they typically assume that they are given complete systems.

An elegant way to obtain a system with the above properties is to provide a preorder on the finite-state models that captures the notion of "more behaviors" and to use a logic whose semantics relate to the preorder. The preorder should preserve satisfaction of formulas of the logic, i.e., if a formula is true for a model, it should also be true for any model which is smaller in the preorder. Additionally, composition should preserve the preorder, and a system should be smaller in the preorder than its individual components. Finally, satisfaction of a formula should correspond to being smaller than a particular model (a tableau for the formula) in the preorder. In such a framework, the above reasoning sequence might be expressed as: T is the tableau of φ, $M \| T \models \psi$, $M' \preceq T$, and hence $M \| M' \models \psi$. Note that assumptions may be given either as formulas or directly as finite-state models, whichever is more concise or convenient. More complex forms of reasoning such as induction [Kurshan and McMillan 1989] are also possible within this framework.

In choosing a computational model, a logic and a preorder to obtain a system such as this we are guided by the following considerations. First, we must be able to realistically model physical systems such as circuits. Second, there should be efficient procedures for model checking and for checking the preorder. Finally, it should be possible to implement these procedures effectively using symbolic techniques.

In this article, we propose a preorder for use with a subset of the logic CTL* [Emerson and Halpern 1986]. This subset is strictly more expressive than LTL. Further, the induced subset of CTL, called ∀CTL, is expressive enough for most verification tasks and has an efficient model-checking algorithm. We also give a tableau construction for this CTL subset. The construction provides a means of temporal reasoning and makes it possible to use formulas as assumptions. The main advantage of CTL over linear temporal logics (e.g., LTL) is in its low-complexity algorithms for model checking and equivalence and preorder testing (polynomial for CTL, exponential for LTL).

Our preorder and the semantics of our logics both include a notion of fairness. This is essential for modeling systems such as communication protocols. We show how to use our results to verify systems composed of Moore machines. Moore machines have an explicit notion of input and output and are particularly suitable to model synchronous circuits. Finally, we suggest efficient methods for checking the preorder in several interesting cases. We have implemented a system based on these results; the system supports efficient compositional verification and temporal reasoning for ∀CTL.

Our article is organized as follows. Section 2 surveys some related work. In Section 3, we present the logic and its semantics (for Kripke structures). The preorder and some of its properties are given in Section 4. The next section defines the semantics of the logic for Moore machines. Given a Moore machine and a formula, we show how to efficiently check whether for all environments, the Moore machine in the environment satisfies the formula. Section 6 presents the tableau construction and demonstrates how to use it for temporal reasoning. Methods for checking the preorder are discussed in Section 7. Section 8 gives a compositional verification of a simple CPU controller. We conclude with a summary and some directions for future work.

2. RELATED WORK

Much of the work on reducing the complexity of automatic verification can be grouped into two classes. The first class includes methods to build a reduced global-state graph or to expand only the needed portion of the global-state graph.

Local-model-checking algorithms [Cleaveland 1990; Stirling and Walker 1989; Winskel 1989] based on logics like the μ-calculus use a tableau-based procedure to deduce that a specific state (the initial state of the system) satisfies a given logical formula. The state space can be generated as needed in such an algorithm, and for some formulas, only a small portion of the space may have to be examined. The main drawback of these algorithms is that often the entire space is generated (for example, when checking that a property holds globally). It is also not clear whether the algorithms can take good advantage of symbolic representations.

Graf and Steffen [1990] describe a method for generating a reduced version of the global-state space given a description of how the system is structured and specifications of how the components interact. Clark et al. [1989a] describe a similar attempt. Both methods will still produce large state graphs if most of the states in the system are not equivalent, and much of the verification must be redone if part of the system changes. Shtadler and Grumberg [1989] show how to verify networks of processes whose structure is described by grammars. In this approach, which involves finding the global behavior of each component, networks of arbitrary complexity can be verified by checking one representative system. For many systems, however, the number of states may still be prohibitive, and it is not clear whether the method can use symbolic representations.

The methods in the second class are *compositional*; properties of the individual components are verified, and properties of the global system are deduced from these. A representation of the global-state space is not built.

Josko [1989] gives an algorithm for checking whether a system satisfies a CTL specification in all environments. His algorithm also allows assumptions about the environment to be specified in a restricted linear-time logic. The system is able to handle assume-guarantee reasoning. The method is fairly *ad hoc* however, and more complex forms of reasoning such as induction cannot be easily incorporated into the system.

Within the framework of CCS [Milner 1980], there have been a number of suggestions for compositional reasoning. Larsen [1993] investigates the expressive power of formalisms for specifying the behavior of a process in a system. He suggests equivalence, refinement, and satisfaction (of a formula) as three interesting relations between an implementation and its specification. However, he does not discuss the applicability of these ideas to verification, nor does he suggest how how they can be implemented. Walker [1988] demonstrates how to use a preorder plus knowledge of how a system should operate to simplify the verification of bisimulation equivalence. Cleaveland and Steffen [1990] use a similar idea. Winskel, in a draft copy of "Compositional Checking of Validity on Finite State Processes," proposes a method for decomposing specifications into properties which the components of a system must satisfy for the specification to hold. The approach is very appealing, but unfortunately, dealing with parallel composition is difficult. It is not apparent whether any of these methods will work well with symbolic representations.

Kurshan [1989] describes a verification methodology based on testing containment of ω-regular languages. Homomorphic reductions are used to map implementations to specifications, and the specifications may be used as implementations at the next level of abstraction. Dill [1989] proposes an elegant form of trace theory which can be used in a similar manner, but the framework does not handle liveness properties as well. Both approaches depend on specifications being deterministic for efficiency, and neither approach makes provisions for using logical formulas as specifications or assumptions.

Shurek and Grumberg [1990] describe criteria for obtaining a modular framework, and they illustrate the idea using CTL* with only universal path quantifiers. This system is closest to the work presented here, but they give no provisions for handling fairness efficiently, using formulas as assumptions, or supporting temporal reasoning. Models in their system are also associated with a fixed decomposition into components; hence it is unclear how to perform inductive reasoning in the framework.

3. TEMPORAL LOGIC

The logics presented in this section are branching-time temporal logics. In order to be able to efficiently decide whether a formula is true in all systems containing a given component, we eliminate the existential path quantifier from the logics. Thus, a formula may include only the universal quantifier $\forall$ over paths, but unlike in linear-time temporal logic, nesting of path quantifiers is allowed. To ensure that existential path quantifiers do not arise via negation, we will assume that formulas are expressed in negation normal form. In other words, negations are applied only to atomic propositions. As a result, the logics contain both $\vee$ and $\wedge$ as boolean operators. The temporal operators are **X** ("nexttime"), **U** ("until"), and **V** ("releases"). **V** is the dual of **U**, i.e., $p\mathbf{V}q$ is true whenever $\neg p\mathbf{U}\neg q$ is false; $p\mathbf{V}q$ is true of a path when q is true starting at the first state on the path, and q remains true up to and including the first point where p becomes true. Thus, p becoming true "releases" the obligation that q remain true. There is no requirement that p ever become true; a path where q was invariantly true would also satisfy $p\mathbf{V}q$. The more familiar $\mathbf{G}p$ can be viewed as an abbreviation for *false* $\mathbf{V}p$. The logics are interpreted over a form of Kripke structure with fairness constraints. Path quantifiers range over the fair paths in the structures.

Definition 1. (**∀CTL***) The logic ∀CTL* is the set of state formulas given by the following inductive definition.

(1) The constants *true* and *false* are state formulas. For every atomic proposition p, p and $\neg p$ are state formulas.
(2) If φ and ψ are state formulas, then $\varphi \wedge \psi$ and $\varphi \vee \psi$ are state formulas.
(3) If φ is a path formula, then $\forall(\varphi)$ is a state of formula.
(4) If φ is a state formula, then φ is a path formula.
(5) If φ and ψ are path formulas, then so are $\varphi \wedge \psi$, $\varphi \vee \psi$.
(6) If φ and ψ are path formulas, then so are

(a) $\mathbf{X}\varphi$,
(b) $\varphi\mathbf{U}\psi$, and
(c) $\varphi\mathbf{V}\psi$.

We also use the following abbreviations: $\mathbf{F}\varphi$ and $\mathbf{G}\varphi$, where φ is a path formula, denote $(\mathit{true}\ \mathbf{U}\varphi)$ and $(\mathit{false}\ \mathbf{V}\varphi)$ respectively.

∀CTL is a restricted subset of ∀CTL* in which the ∀ path quantifier may only precede a restricted set of path formulas. More precisely, ∀CTL is the logic obtained by eliminating rules 3 through 6 above and adding the following rule.

(3′) If φ and ψ are state formulas, then $\forall\mathbf{X}\varphi$, $\forall(\varphi\mathbf{U}\psi)$, and $\forall(\varphi\mathbf{V}\psi)$ are state formulas.

In practice, we have found that many of the formulas which are used in specifying and verifying systems are expressible in ∀CTL, and almost all are expressible in ∀CTL*. An example formula which is not expressible in ∀CTL* is a weak form of absence of deadlock: $\forall\mathbf{G}\exists\mathbf{F}p$ states that it should always be possible to reach a state where p holds.

We will give the semantics of the logic using a form of Kripke structure with fairness constraints.

Definition 2. (*structure*) A structure $M = \langle S, S_0, \mathscr{A}, \mathscr{L}, R, \mathscr{F}\rangle$ is a tuple of the following form.

(1) S is a finite set of states.
(2) $S_0 \subseteq S$ is set of initial states.
(3) $\mathscr{A}$ is a finite set of atomic propositions.
(4) $\mathscr{L}$ is a function that maps each state to the set of atomic propositions *true* in that state.
(5) $R \subseteq S \times S$ is a transition relation.
(6) $\mathscr{F}$ is a Streett acceptance condition, represented by pairs of sets of states.

Definition 3. A path in M is an infinite sequence of states $\pi = s_1 s_1 s_2 \ldots$ such that for all $i \in \mathscr{N}$, $R(s_i, s_{i+1})$.

Definition 4. Define $\inf(\pi) = \{s \mid s = s_i$ for infinitely many $i\}$; π is a fair path in M iff for every $(P, Q) \in \mathscr{F}$, if $\inf(\pi) \cap P \neq \emptyset$ then $\inf(\pi) \cap Q \neq \emptyset$.

The notation π^n will denote the suffix of π which begins at s_n. We now consider the semantics of the logic ∀CTL* with atomic propositions drawn from the set $\mathscr{A}$.

Definition 5. (*satisfaction of a formula*) Satisfaction of a state formula φ by a state s ($s \models \varphi$) and of a path formula ψ by a fair path π ($\pi \models \psi$) is defined inductively as follows.

(1) $s \models \mathit{true}$, and $s \not\models \mathit{false}$. $s \models p$ iff $p \in \mathscr{L}(s)$. $s \models \neg p$ iff $p \notin \mathscr{L}(s)$.
(2) $s \models \varphi \wedge \psi$ iff $s \models \varphi$ and $s \models \psi$. $s \models \varphi \vee \psi$ iff $s \models \varphi$ or $s \models \psi$.
(3) $s \models \forall(\varphi)$ iff for every fair path π starting at s, $\pi \models \varphi$.

(4) $\pi \models \varphi$, where φ is a state formula, iff the first state of π satisfies the state formula.

(5) $\pi \models \varphi \wedge \psi$ iff $\pi \models \varphi$ and $\pi \models \psi$. $\pi \models \varphi \vee \psi$ iff $\pi \models \varphi$ or $\pi \models \psi$.

(6) (a) $\pi \models \mathbf{X}\varphi$ iff $\pi^1 \models \varphi$.

(b) $\pi \models \varphi\mathbf{U}\psi$ iff there exists $n \in \mathscr{N}$ such that $\pi^n \models \psi$ and for all $i < n$, $\pi^i \models \varphi$.

(c) $\pi \models \varphi\mathbf{V}\psi$ iff for all $n \in \mathscr{N}$, if $\pi^i \not\models \varphi$ for all $i < n$, then $\pi^n \models \psi$.

$M \models \varphi$ indicates that for every $s_0 \in S_0$, $s_0 \models \varphi$.

Emerson and Halpern [1986] compared the expressive power of the three logics LTL, CTL, and CTL*. They showed that LTL and CTL have incomparable expressive power, while CTL* is strictly more expressive than either of the others. Eliminating the existential path quantifier from CTL and CTL* does not affect the relative expressive power of the logics. ∀CTL* trivially encompasses LTL and ∀CTL. The formula $\forall\mathbf{F}\forall\mathbf{G}p$ is a formula of ∀CTL that does not have an equivalent LTL formula. On the other hand, there is no ∀CTL formula that is equivalent to the LTL formula $\forall\mathbf{F}\mathbf{G}p$. Thus, LTL and ∀CTL are incomparable, and both are strictly less expensive than ∀CTL*.

4. SIMULATION RELATIONS AND COMPOSITION OF STRUCTURES

In this section, we define the preorder which we use and examine some of its properties. We also show how these properties make assume-guarantee-style reasoning possible.

Definition 6. (*structure simulation*) Let M and M' be two structures with $\mathscr{A} \supseteq \mathscr{A}'$, and let t and t' be states in S and S', respectively. A relation $H \subseteq S \times S'$ is a simulation relation from (M, t) to (M', t') iff the following conditions hold.

(1) $H(t, t')$.

(2) For all s and s', $H(s, s')$ implies

(a) $\mathscr{L}(s) \cap \mathscr{A}' = \mathscr{L}'(s')$; and

(b) for every fair path $\pi = s_0 s_1 \ldots$ from $s = s_0$ in M there exists a fair path $\pi' = s'_0 s'_1 \ldots$ from $s' = s'_0$ in M' such that for every $i \in \mathscr{N}$, $H(s_i, s'_i)$.

When H satisfies property (2), we say H is a simulation relation. H is a simulation from M to M' iff for every $s_0 \in S_0$ there is $s'_0 \in S'_0$ such that $H(s_0, s'_0)$. To indicate that two paths correspond as in item (2b) above, we write $H(\pi, \pi')$.

Definition 7. For $s \in S$ and $s' \in S'$, $(M, s) \preceq (M', s')$ iff there is a simulation relation from (M, s) to (M', s'). $M \preceq M'$ iff there exists a simulation relation from M to M'.

When M and M' are understood, we sometimes write $s \preceq s'$. Intuitively, one state can be simulated by another if their labels agree on the atomic propositions of the second structure and if for every fair path from the first state there is a corresponding fair path from the second state. One structure can be simulated by another if for every initial state of the first, there is a corresponding initial state of the second. One may view the second structure as a specification and the first as its implementation. Since a specification may hide some of the implementation details, it may have a smaller set of atomic propositions.

Definition 8. (*composition of structures*) Let M and M' be two structures. The composition of M and M', denoted $M \| M'$, is the structure M'' defined as follows.

(1) $S'' = \{s, s') | \mathscr{L}(s) \cap \mathscr{A}' = \mathscr{L}'(s') \cap \mathscr{A}\}$.
(2) $S''_0 = (S_0 \times S'_0) \cap S''$.
(3) $\mathscr{A}'' = \mathscr{A} \cup \mathscr{A}'$.
(4) $\mathscr{L}''((s, s')) = \mathscr{L}(s) \cup \mathscr{L}'(s')$.
(5) $R''((s, s'), (t, t'))$ iff $R(s, t)$ and $R'(s', t')$.
(6) $\mathscr{F}'' = \{((P \times S') \cap S'', (Q \times S') \cap S'') | (P, Q) \in \mathscr{F}\}$
$\cup \{((S \times P') \cap S'', (S \times Q') \cap S'') | (P', Q') \in \mathscr{F}'\}$.

The choice of this definition of composition is motivated by its correspondence with composition of Moore machines. Each transition of the composition is a joint transition of the components, and states of the composition are pairs of component states that agree on their common atomic propositions. We first note that this composition operator has the usual properties.

THEOREM 1. *Composition of structures is commutative and associative (up to isomorphism).*

PROOF. Straightforward but tedious. □

We now turn to the connections between the relation $\preceq$ and composition. To begin, we note that a path in $M \| M'$ is fair iff its restriction to each component results in a fair path.

LEMMA 1. *Let $M'' = M \| M'$. The following conditions are equivalent.*

(1) $\pi'' = (s_0, s'_0)(s_1, s'_1) \ldots$ *is a fair path in* M''.
(2) $\pi = s_0 s_1 \ldots$ *and* $\pi' = s'_0 s'_1 \ldots$ *are fair paths in M and M' respectively, and (s_i, s'_i) is a state of M'' for all $i \in \mathscr{N}$.*

PROOF. Assume condition (1) above. By the definition of composition, $\pi = s_0 s_1 \ldots$ is a path in M. Let $(P, Q) \in \mathscr{F}$, and suppose $\inf(\pi) \cap P \neq \emptyset$. Now $(P'', Q'') = ((P \times S') \cap S'', (Q \times S') \cap S'') \in \mathscr{F}''$, and $\inf(\pi'') \cap P'' \neq \emptyset$. By the definition of a fair path, $\inf(\pi'') \cap Q'' \neq \emptyset$. Hence $\inf(\pi) \cap Q \neq \emptyset$, and so π is a fair path in M. Similarly, $\pi' = s'_0 s'_1 \ldots$ is a fair path in M'.

Assume condition (2) above. From the definition of composition, $\pi'' = (s_0, s'_0)(s_1, s'_1) \ldots$ is a path in M''. Suppose $(P'', Q'') \in \mathscr{F}''$ and $\inf(\pi'') \cap P'' \neq \emptyset$. Either $(P'', Q'') = ((P \times S') \cap S'', (Q \times S') \cap S'')$ for some $(P, Q) \in \mathscr{F}$, or $(P'', Q'') = ((S \times P') \cap S'', (S \times Q') \cap S'')$ for some $(P', Q') \in \mathscr{F}'$. In the first case, we have $\inf(\pi) \cap P \neq \emptyset$, and so $\inf(\pi) \cap Q \neq \emptyset$. This implies $\inf(\pi'') \cap Q'' \neq \emptyset$. The second case is similar. Hence π'' is a fair path in M''. □

THEOREM 2.

(1) $\preceq$ *is a preorder*
(2) *For all M and M', $M \| M' \preceq M$.*
(3) *For all M, M' and M'', if $M \preceq M'$ then $M \| M'' \preceq M' \| M''$.*
(4) *For all M, $M \preceq M \| M$.*

PROOF.

(1) The relation $H = \{(s, s) | s \in S\}$ is a simulation relation from M to M, so $\preceq$ is reflexive. Thus it only remains to show that $\preceq$ is transitive. Assume $M \preceq M'$ and $M' \preceq M''$. Let H_0 be a simulation relation from M to M', and let H_1 be a simulation relation from M' to M''. Define H_2 as the relational product of H_0 and H_1, i.e.,

$$H_2 = \{(s, s'') | \exists s' [H_0(s, s') \wedge H_1(s', s'')]\}.$$

If $s_0 \in S_0$, then by the definition of simulation relation, there exists $s'_0 \in S'_0$ such that $H_0(s_0, s'_0)$. Similarly, there exists $s''_0 \in S''_0$ such that $H_1(s'_0, s''_0)$, and hence $H_2(s_0, s''_0)$.

Suppose $H_2(s, s'')$ and let s' be such that $H_0(s, s')$ and $H_1(s', s'')$. By the definition of simulation relation, $\mathscr{L}(s) \cap \mathscr{A}' = \mathscr{L}'(s')$ and $\mathscr{L}'(s') \cap \mathscr{A}'' = \mathscr{L}''(s'')$. Then since $\mathscr{A}' \supseteq \mathscr{A}''$, we have $\mathscr{L}(s) \cap \mathscr{A}'' = \mathscr{L}''(s'')$. If π is a fair path in M from s, then there exists a fair path π' from s' in M' such that $H_0(\pi, \pi')$, Since H_1 is a simulation relation, there exists a fair path π'' from s'' in M'' such that $H_1(\pi', \pi'')$. But then $H_2(\pi, \pi'')$, and hence H_2 is a simulation relation from M to M''. Thus $M \preceq M''$.

(2) Define H by

$$H = \{((s, s'), s) | (s, s') \in S^{M \| M'}\}.$$

If (s_0, s'_0) is an initial state of $M \| M'$, then $s_0 \in S_0$. The label of (s, s') is $\mathscr{L}(s) \cup \mathscr{L}'(s')$, and $(\mathscr{L}(s) \cup \mathscr{L}'(s')) \cap \mathscr{A} = \mathscr{L}(s)$. If $(s_0, s'_0)(s_1, s'_1)\ldots$ is a fair path in $M \| M'$, then by the previous lemma, $s_0, s_1 \ldots$ is a fair path in M. By the definition of H, we have $H((s_i, s'_i), s_i)$ for every i. Hence H is a simulation relation, and $M \| M' \preceq M$.

(3) Let H_0 be a simulation relation from M to M'. Define H_1 by

$$H_1 = \{((s, s''), (s', s'')) | H_0(s, s')\}.$$

We show that H_1 is a simulation relation. Let (s_0, s''_0) be an initial state of $M \| M''$. By the definition of composition, $s_0 \in S_0$ and $s''_0 \in S''_0$. Since $M \preceq M'$, there exists $s'_0 \in S'_0$ such that $H_0(s_0, s'_0)$. Now (s'_0, s''_0) is a state of $M' \| M''$ since

$$\begin{aligned}\mathscr{L}'(s'_0) \cap \mathscr{A}'' &= (\mathscr{L}(s_0) \cap \mathscr{A}') \cap \mathscr{A}'' \\ &= (\mathscr{L}(s_0) \cap \mathscr{A}'') \cap \mathscr{A}' \\ &= (\mathscr{L}''(s''_0) \cap \mathscr{A}) \cap \mathscr{A}' \\ &= \mathscr{L}''(s''_0) \cap \mathscr{A}'.\end{aligned}$$

Further, (s'_0, s''_0) is an initial state of $M' \| M''$ by the definition of composition. By definition of H_1, we have $H_1((s_0, s''_0), (s'_0, s''_0))$.
Suppose $H_1((s, s''), (s', s''))$. First note that

$$\begin{aligned}(\mathscr{L}(s) \cup \mathscr{L}''(s'')) \cap (\mathscr{A}' \cup \mathscr{A}'') &= (\mathscr{L}(s) \cap \mathscr{A}') \cup (\mathscr{L}(s) \cap \mathscr{A}'') \\ &\quad \cup (\mathscr{L}''(s'') \cap (\mathscr{A}' \cup \mathscr{A}'')) \\ &= \mathscr{L}'(s') \cup (\mathscr{L}''(s'') \cap \mathscr{A}) \cup \mathscr{L}''(s'') \\ &= \mathscr{L}'(s') \cup \mathscr{L}''(s'').\end{aligned}$$

Let $(s_0, s''_0)(s_1, s''_1)\ldots$ be a fair path in $M \| M''$ from $(s, s'') = (s_0, s''_0)$. Then for every $i \in \mathscr{N}$, we have $\mathscr{L}(s_i) \cap \mathscr{A}'' = \mathscr{L}''(s''_i) \cap \mathscr{A}$. By the previous lemma, $\pi = s_0 s_1 \ldots$ is a fair path in M starting at s, and $\pi'' = s''_0 s''_1 \ldots$ is a fair path in M'' from s''. Since $H_0(s, s')$, there is a path $\pi' = s'_0 s'_1 \ldots$ from $s' = s'_0$ in M' such that for every $i \in \mathscr{N}$, we have $H_0(s_i, s'_i)$. By the definition of simulation relation, $\mathscr{L}(s_i) \cap \mathscr{A}' = \mathscr{L}'(s'_i)$ for all i. Arguing as above, we then have $\mathscr{L}'(s'_i) \cap \mathscr{A}'' = \mathscr{L}''(s''_i) \cap \mathscr{A}'$ for each i, and so each (s'_i, s''_i) is a state in $M' \| M''$. Now $H_1((s_i, s''_i), (s'_i, s''_i))$ by the definition of H_1. Applying the previous lemma, we find that $(s'_0, s''_0)(s'_1, s''_1)\ldots$ is a fair path starting in (s', s'') and corresponding to the path $(s_0, s''_0)(s_1, s''_1)\ldots$.

(4) First note that for every state s of M, (s, s) is a state of $M \| M$. Define $H = \{(s, (s, s)) | s \in S\}$. If $s_0 \in S_0$, then by the definition of composition, (s_0, s_0) is an initial state of $M \| M$; (s, s) trivially has the same label as s. Using the previous lemma and the definition of composition, we find that, if $s_0 s_1 \ldots$ is a fair path in M, then $(s_0, s_0)(s_1, s_1)\ldots$ is a fair path in $M \| M$. By the definition of H, we have $H(s_i, (s_i, s_i))$ for all i. Hence H is a simulation relation, and $M \preceq M \| M$. □

THEOREM 3. *Let s and s' be states of M and M', and let H be a simulation relation such that $H(s, s')$. Let π and π' be fair paths such that $H(\pi, \pi')$. Then*

(1) *for every $\forall CTL^*$ (state) formula φ (with all atomic propositions in $\mathscr{A}'$), if $s' \models \varphi$, then $s \models \varphi$; and*
(2) *for every $\forall CTL^*$ path formula φ (with all atomic propositions in $\mathscr{A}'$), if $\pi' \models \varphi$, then $\pi \models \varphi$.*

PROOF. The proof proceeds by induction on the structure of the formula.

(1) If $\varphi = true$ or $\varphi = false$, the result is trivial. If $\varphi = p$, an atomic proposition, then $s' \models \varphi$ if and only if $p \in \mathscr{L}'(s')$. By the definition of simulation relation, $\mathscr{L}(s) \cap \mathscr{A}' = \mathscr{L}'(s')$, and so $p \in \mathscr{L}(s)$ iff $p \in \mathscr{L}'(s')$. Thus $s \models \varphi$. The case where $\varphi = \neg p$ is similar.
(2) If $\varphi = \varphi_1 \wedge \varphi_2$, then $s' \models \varphi$ iff $s' \models \varphi_1$ and $s' \models \varphi_2$. The induction hypothesis implies $s \models \varphi_1$ and $s \models \varphi_2$; hence $s \models \varphi$. The case where $\varphi = \varphi_1 \vee \varphi_2$ is similar.
(3) If $\varphi = \forall(\varphi_1)$, then $s \models \varphi$ iff for every fair path π from s, $\pi \models \varphi_1$. Let π be any fair path from s. By the definition of simulation relation, there exists a fair path π' from s' such that $H(\pi, \pi')$. If $s' \models \varphi$, then $\pi' \models \varphi_1$ for any

π' from s'. The induction hypothesis then implies $\pi \models \varphi_1$, and hence $s \models \varphi$.

(4) If φ is a path formula consisting of only a state formula and $\pi' \models \varphi$, then the initial state s' of π' satisfies φ. By the induction hypothesis, $s \models \varphi$, and since s is the initial state of π, $\pi \models \varphi$.

(5) The cases for the conjunction and disjunction of path formulas are similar to case 2.

(6) (a) If $\varphi = \mathbf{X}\varphi_1$, then $\pi' \models \varphi$ implies $\pi'^1 \models \varphi_1$. Now since $H(\pi, \pi')$, we also have $H(\pi^1, \pi'^1)$. Then the induction hypothesis implies $\pi^1 \models \varphi_1$. Thus $\pi \models \varphi$.

(b) If $\varphi = \varphi_1 \mathbf{U} \varphi_2$, then $\pi' \models \varphi$ implies that there exists n such that $\pi'^n \models \varphi_2$ and for all $i < n$, $\pi'^i \models \varphi_1$. Since $H(\pi, \pi')$, we have $H(\pi^j, \pi'^j)$ for any j. Applying the induction hypothesis, $\pi^n \models \varphi_2$ and $\pi^i \models \varphi_1$ for all $i < n$. Hence $\pi \models \varphi$.

(c) The case where $\varphi = \varphi_1 \mathbf{V} \varphi_2$ is similar to the previous two cases. □

COROLLARY 1. *Suppose $M \preceq M'$. Then for every* ∀*CTL* formula φ (with atomic propositions in $\mathscr{A}'$), $M' \models \varphi$ implies $M \models \varphi$.*

PROOF. Immediate. □

Using Theorem 2 and this corollary, we see that a standard CTL (CTL*) model-checking algorithm [Clarke et al. 1986], when restricted to ∀CTL (∀CTL*), can be viewed as determining whether a formula is true of all systems containing a given component. This is the key to compositional verification. With the theorem and corollary, it is also straightforward to justify the soundness of the assume-guarantee paradigm when assumptions are given as structures. (The connection between structures and formulas will be examined in Section 6.) Discharging an assumption involves checking for the relation $\preceq$. Suppose that we wish to check that $M \| M' \models \varphi$ and that we have verified the following relationships:

$$M \preceq A$$
$$A \| M' \preceq A'$$
$$M \| A' \models \varphi.$$

In other words, M discharges assumption A; M' under assumption A discharges assumption A'; and M under assumption A' satisfies the desired formula. From Theorem 2, we have

$$\begin{aligned} M \| M' &\preceq M \| M \| M' \\ &\preceq M \| A \| M' \\ &\preceq M \| A'. \end{aligned}$$

Then Corollary 1 implies that $M \| M' \models \varphi$. The theorem and corollary also show that any system containing $M \| M'$ will satisfy φ. Note that φ is not necessarily true in either M or M' and may involve atomic propositions from both M and M'.

5. MOORE MACHINES

We have seen that the structures defined earlier (Definition 2), can be used for compositional reasoning about synchronous systems. However, such systems are typically given using a more common finite-state model such as Moore machines [Hopcroft and Ullman 1979]. Moore machines are models of computation with an explicit notion of inputs and outputs. Since the inputs originate from an external, uncontrolled environment, the machine can always receive any combination of input values. Moore machines are synchronous; in a composition of Moore machines, each machine makes a single step at every point. Thus, they are most suitable for modeling synchronous circuits. In this section, we show a natural correspondence between Moore machines with an empty set of inputs and the structures defined earlier. We use this correspondence to define the semantics of ∀CTL* with respect to Moore machines, and we show how to use compositional reasoning to verify a system composed of Moore machines.

Definition 9. (*Moore machine*) A Moore machine $M = \langle S, S_0, I, O, \mathscr{L}, R\rangle$ is a tuple of the following form

(1) S is a finite set of states.
(2) $S_0 \subseteq S$ is a set of initial states.
(3) I is a finite set of input propositions.
(4) O is a finite set of output propositions.
(5) $\mathscr{L}$ is a function that maps each state to the set of output propositions true in that state.
(6) $R \subseteq S \times 2^I \times S$ is the transition relation.

We require that $I \cap O = \varnothing$ and that for every $s \in S$ and $v \subseteq I$, there exists some $t \in S$ such that $R(s, v, t)$. We also let $\mathscr{A}$ denote $I \cup O$.

Definition 10. (*composition of Moore machines*) Let M and M' be Moore machines with $O \cap O' = \varnothing$. The composition of M and M', denoted $M \| M'$, is the Moore machine M'' defined as follows.

(1) $S'' = S \times S'$.
(2) $S''_0 = S_0 \times S'_0$.
(3) $I'' = (I \cup I') \setminus (O \cup O')$.
(4) $O'' = O \cup O'$.
(5) $\mathscr{L}''((s, s')) = \mathscr{L}(s) \cup \mathscr{L}'(s')$.
(6) $R''((s s'), v, (t, t'))$ iff $R(s, (v \cup \mathscr{L}'(s')) \cap I, t)$ and $R'(s', (v \cup \mathscr{L}(s)) \cap I', t')$.

We now turn to the question of how to define satisfaction of a specification by a Moore machine M. The key consideration is that we wish to have a compositional method of reasoning. Thus, *M satisfying a specification should mean that M plus any environment satisfies that specification.* We will achieve this by considering the behavior of complete systems involving M.

Definition 11. A Moore machine M is called closed if $I = \emptyset$.

Intuitively, the behavior of a closed machine cannot be altered. For such a machine, there is a structure which naturally corresponds to it. We define this structure precisely now. The definition here is actually slightly more general in that it assigns a structure to nonclosed machines as well.

Definition 12. (*structure for a Moore machine*) The structure M' corresponding to a Moore machine M, denoted by $K(M)$, is defined as follows.

(1) $S' = S \times 2^I$.
(2) $S'_0 = S_0 \times 2^I$.
(3) $\mathscr{A}' = \mathscr{A} = I \cup O$.
(4) $\mathscr{L}'((s, v)) = \mathscr{L}(s) \cup v$.
(5) $R'((s, v_1), (t, v_2))$ iff $R(s, v_1, t)$.
(6) $\mathscr{F}' = \emptyset$.

Definition 13. A Moore machine M' is called a closing environment for M if $O \cap O' = \emptyset$, $I \subseteq O'$, and $I' \subseteq O$.

If M' is a closing environment for M, then M and M' can be composed, and the resulting Moore machine will be closed. We now define satisfaction of a formula by a Moore machine.

Definition 14. (*satisfaction in a Moore machine*) If M is a Moore machine and φ is a $\forall$CTL* formula with atomic propositions over $\mathscr{A}$, the $M \models \varphi$ iff for every closing environment M' for M, $K(M \| M') \models \varphi$.

We must now demonstrate how to efficiently check whether $M \models \varphi$.

LEMMA 2. *If M and M' are Moore machines with $O \cap O' = \emptyset$, then $K(M \| M')$ is isomorphic to $K(M) \| K(M')$.*

PROOF. Define ϕ mapping the states of $K(M \| M')$ to the states of $K(M) \| K(M')$ as follows.

$$\phi(((s, s'), v)) = ((s, (v \cup \mathscr{L}'(s')) \cap I), (s', (v \cup \mathscr{L}(s)) \cap I'))$$

Suppose $((s, s'), v)$ and $((t, t'), u)$ both map to the same state of $K(M) \| K(M')$. Then from the definition of ϕ, we immediately have $s = t$ and $s' = t'$. Also, $(v \cup \mathscr{L}'(s')) \cap I = (u \cup \mathscr{L}'(t')) \cap I$, and $(v \cup \mathscr{L}(s)) \cap I' = (u \cup \mathscr{L}(t)) \cap I'$. By the definition of Moore machine composition, v and u are disjoint from $O \cup O'$. Hence $v \cap I = u \cap I$, and $v \cap I' = u \cap I'$. This implies $v \cap (I \cup I') = u \cap (I \cup I')$, i.e., $v = u$. Hence ϕ is an injection.

To argue that ϕ is surjective, we consider the cardinalities of the two sets of states. First, we have

$$|S^{K(M \| M')}| = |S| \cdot |S'| \cdot 2^{|(I \cup I') \setminus (O \cup O')|}.$$

Now consider $|S^{K(M) \| K(M')}|$. This is the number of states in the cross product $S^{K(M)} \times S^{K(M')}$ which have compatible labelings. Fix a pair of states of s and s'. There are $2^{|I|}$ states in $K(M)$ with s as their first component and $2^{|I'|}$ in

$K(M')$ with s' as the first component. Thus there are potentially $2^{|I|} \cdot 2^{|I'|}$ states in $K(M) \| K(M')$ corresponding to s and s'. However, each pair must correspond on the atomic propositions in $I \cap O'$, $I' \cap O$, and $I \cap I'$. Thus there are exactly

$$\frac{2^{|I|} \cdot 2^{|I'|}}{2^{|I \cap O'|} \cdot 2^{|I' \cap O|} \cdot 2^{|I \cap I'|}}$$

states in $K(M) \| K(M')$ corresponding to s and s'. Thus we have

$$\begin{aligned}
|S^{K(M) \| K(M')}| &= \frac{|S| \cdot |S'| \cdot 2^{|I|} \cdot 2^{|I'|}}{2^{|I \cap O'|} \cdot 2^{|I' \cap O|} \cdot 2^{|I \cap I'|}} \\
&= \frac{|S| \cdot |S'| \cdot 2^{|I \cup I'|}}{2^{|I \cap O'|} \cdot 2^{|I' \cap O|}} \\
&= \frac{|S| \cdot |S'| \cdot 2^{|I \cup I'|}}{2^{|I \cap O'|} \cdot 2^{|I \cap O|} \cdot 2^{|I' \cap O|} \cdot 2^{|I' \cap O'|}} \\
&= \frac{|S| \cdot |S'| \cdot 2^{|I \cup I'|}}{2^{|(I \cup I') \cap (O \cup O')|}} \\
&= |S| \cdot |S'| \cdot 2^{|(I \cup I') \setminus (O \cup O')|} \\
&= |S^{K(M \| M')}|.
\end{aligned}$$

Hence ϕ is a bijection.

If $((s_0, s_0'), v)$ is an initial state of $K(M \| M')$, then $s_0 \in S_0$ and $s_0' \in S_0'$. Then $\phi((s_0, s_0'), v)$ is an initial state of $K(M) \| K(M')$ since $s_0 \in S_0$ implies that $(s_0, (v \cup \mathscr{L}'(s_0')) \cap I)$ is an initial state of $K(M)$, and $s_0' \in S_0'$ implies that $(s_0', (v \cup \mathscr{L}(s_0)) \cap I')$ is an initial state of $K(M')$. Similarly, if $\phi((s, s'), v)$ is an initial state of $K(M) \| K(M')$, then $((s, s'), v)$ is an initial state of $K(M \| M')$.

The sets of atomic propositions of the two structures are clearly identical. The labeling of $((s, s'), v)$ is $\mathscr{L}(s) \cup \mathscr{L}'(s') \cup v$. The labeling of $\phi((s, s'), v)$ is

$$\begin{aligned}
&\mathscr{L}^{K(M)}((s, (v \cup \mathscr{L}'(s')) \cap I)) \cup \mathscr{L}^{K(M')}((s', (v \cup \mathscr{L}(s)) \cap I')) \\
&\quad = \mathscr{L}(s) \cup ((v \cup \mathscr{L}'(s')) \cap I) \cup \mathscr{L}'(s') \cup ((v \cup \mathscr{L}(s)) \cap I') \\
&\quad = \mathscr{L}(s) \cup \mathscr{L}'(s') \cup (v \cap (I \cup I')) \\
&\quad = \mathscr{L}(s) \cup \mathscr{L}'(s') \cup v \\
&\quad = \mathscr{L}^{K(M \| M')}(((s, s'), v)).
\end{aligned}$$

$R^{K(M \| M')}(((s, s'), v), ((t, t'), u))$ iff $R^{M \| M'}((s, s'), v, (t, t'))$ iff $R(s, (v \cup \mathscr{L}'(s')) \cap I, t)$, and $R'(s', (v \cup \mathscr{L}(s)) \cap I', t')$ iff $R^{K(M)}((s, (v \cup \mathscr{L}'(s')) \cap I), (t, (u \cup \mathscr{L}'(t')) \cap I))$; and $R^{K(M')}((s', (v \cup \mathscr{L}(s)) \cap I'), (t', (u \cup \mathscr{L}(t)) \cap I'))$ iff $R^{K(M) \| K(M')}(\phi(((s, s'), v)), \phi((t, t'), u))$. The fairness sets of both structures are empty. □

Definition 15. If M is a Moore machine, the maximal closing environment for M, denoted $E(M)$, is the Moore machine M' defined as follows.

(1) $S' = 2^I$.

(2) $S_0' = S'$.

(3) $I' = \emptyset$.
(4) $O' = I$.
(5) $\mathscr{L}'(s') = s'$.
(6) $R'(s', \emptyset, t')$ is identically true.

The maximal environment (for M) represents an environment which can do anything at each step. Intuitively, a possible behavior of M in an arbitrary environment must also be a possible behavior of M in the maximal environment. The logics we use specify properties that should hold for every possible behavior of a system. Hence, if M plus its maximal environment satisfies a formula, then M in any environment should satisfy that formula. Note that the above holds also for environments that cannot be described as finite-state Moore machines.

LEMMA 3. *Suppose M' is a closing environment for M, and suppose $M'' = E(M)$. Then $K(M') \preceq K(M'')$.*

PROOF. Define

$$H = \{(s', s'') | \mathscr{L}'(s') \cap \mathscr{A}'' = \mathscr{L}''(s'')\}.$$

Note that for every $s' \in S'$, there is some $s'' \in S''$ such that $H(s', s'')$ (in particular, the state $\mathscr{L}'(s') \cap \mathscr{A}''$ in $K(M'')$). Thus, if $s'_0 \in S'_0$, there is s''_0 which is related to it by H, and every state in $K(M'')$ is an initial state.

If $H(s', s'')$, then by the definition of H, we have $\mathscr{L}'(s') \cap \mathscr{A}'' = \mathscr{L}''(s'')$. If π' is a fair path in $K(M')$, then the fact that every state in $K(M')$ is related to some state in $K(M'')$ plus the fact that R'' is identically true implies that there is a path π'' in $K(M'')$ such that $H(\pi', \pi'')$. Further, every path in $K(M'')$ is fair. Thus H is a simulation relation. □

LEMMA 4. *Let M be a Moore machine. Then $K(M)$ is isomorphic to $K(M \| E(M))$.*

PROOF. Let $M' = K(M)$ and $M'' = K(M \| E(M))$. Define ϕ mapping the states of M'' to the states of M' by $\phi(((s, v), \emptyset)) = (s, v)$; ϕ is obviously an injection, and ϕ is a surjection since each subset of 2^I is a state of $E(M)$.

If $((s_0, v), \emptyset) \in S''_0$, then s_0 must be in S_0. Hence $(s_0, v) \in S'_0$. Similarly, if $(s_0, v) \in S'_0$, then $s_0 \in S_0$ and so $((s_0, v), \emptyset) \in S''_0$. $\mathscr{A}''$ and $\mathscr{A}'$ are trivially equal. We also have

$$\begin{aligned} \mathscr{L}''(((s, v), \emptyset)) &= \mathscr{L}^{M \| E(M)}((s, v)) \cup \emptyset \\ &= \mathscr{L}(s) \cup \mathscr{L}^{E(M)}(v) \\ &= \mathscr{L}(s) \cup v \\ &= \mathscr{L}'((s, v)). \end{aligned}$$

Finally, we have $R''(((s, v_1), \emptyset), ((t, v_2), \emptyset))$ iff $R^{M \| E(M)}((s, v_1), \emptyset, (t, v_2))$ iff $R(s, v_1, t)$ iff $R'((s, v_1), (t, v_2))$. $\mathscr{F}''$ and $\mathscr{F}'$ are both empty. □

THEOREM 4. *If M is a Moore machine, then $M \models \varphi$ iff $K(M) \models \varphi$.*

PROOF. Suppose $K(M) \models \varphi$. By Lemma 4, we find $K(M \| E(M)) \models \varphi$, and then by Lemma 2, $K(M) \| K(E(M)) \models \varphi$. Let M' be any closing environment for M. By Lemma 3, $K(M') \preceq K(E(M))$. Hence by Theorem 2, $K(M) \| K(M') \preceq K(M) \| K(E(M))$. Applying Corollary 1, we have $K(M) \| K(M') \models \varphi$. By Lemma 2, $K(M) \| K(M')$ is isomorphic to $K(M \| M')$, and thus $K(M \| M') \models \varphi$. Since M' was arbitrary, $M \models \varphi$.

If $M \models \varphi$, then $K(M \| E(M)) \models \varphi$, and hence by Lemma 4, $K(M) \models \varphi$. □

Thus, to determine if a system $M_1 \| M_2 \| \ldots \| M_n$ satisfies a formula φ, we instead check that $K(M_1 \| M_2 \| \ldots \| M_n)$ satisfies φ. By Lemma 2, this is equivalent to checking that $K(M_1) \| K(M_2) \| \ldots \| K(M_n)$ satisfies the formula. As illustrated in the previous section, we can use the assume-guarantee paradigm to try to verify this latter relation. Thus, during an actual verification we will be working with structures even though the thing we want to verify is a property of a composition of Moore machines.

6. THE TABLEAU CONSTRUCTION

In this section, we give a tableau construction for ∀CTL formulas (for a similar construction for LTL, see Burch et al. [1990]; other tableau constructions for CTL are given by Clarke and Emerson [1981] and Ben-Ari et al. [1983], and LTL tableau constructions were given by Wolper [1983] and Pnueli and Sherman [1981]). We show that the tableau of a formula is a maximal model for the formula under the relation $\preceq$. Thus, the structure generated in the construction can be used as an assumption by composing the structure with the desired system before applying the model-checking algorithm. Discharging the assumption is simply a matter of checking that the environment satisfies the formula. We also indicate how the tableau can be used to do temporal reasoning. For the remainder of this section, fix a ∀CTL formula ψ.

Definition 16. The set sub(φ) of subformulas of the formula φ is defined by the following equations.

(1) If $\varphi = \mathit{true}$ or $\varphi = \mathit{false}$ or $\varphi = p$, an atomic proposition, then $\mathrm{sub}(\varphi) = \{\varphi\}$. If $\varphi = \neg p$, a negated atomic proposition, then $\mathrm{sub}(\varphi) = \{\varphi, p\}$.
(2) If $\varphi = \varphi_1 \wedge \varphi_2$ or $\varphi = \varphi_1 \vee \varphi_2$, then $\mathrm{sub}(\varphi) = \{\varphi\} \cup \mathrm{sub}(\varphi_1) \cup \mathrm{sub}(\varphi_2)$.
(3) (a) If $\varphi = \forall \mathbf{X} \varphi_1$, then $\mathrm{sub}(\varphi) = \{\varphi\} \cup \mathrm{sub}(\varphi_1)$.
 (b) If $\varphi = \forall(\varphi_1 \mathbf{U} \varphi_2)$, then $\mathrm{sub}(\varphi) = \{\varphi\} \cup \mathrm{sub}(\varphi_1) \cup \mathrm{sub}(\varphi_2)$.
 (c) If $\varphi = \forall(\varphi_1 \mathbf{V} \varphi_2)$, then $\mathrm{sub}(\varphi) = \{\varphi\} \cup \mathrm{sub}(\varphi_1) \cup \mathrm{sub}(\varphi_2)$.

Definition 17. The set el(φ) of elementary formulas of the formula φ is defined by the following equations.

(1) If $\varphi = \mathit{true}$ or $\varphi = \mathit{false}$, then $\mathrm{el}(\varphi) = \emptyset$. If $\varphi = p$, an atomic proposition, or $\varphi = \neg p$, then $\mathrm{el}(\varphi) = \{p\}$.
(2) If $\varphi = \varphi_1 \wedge \varphi_2$ or $\varphi = \varphi_1 \vee \varphi_2$, then $\mathrm{el}(\varphi) = \mathrm{el}(\varphi_1) \cup \mathrm{el}(\varphi_2)$.

(3) (a) If $\varphi = \forall\mathbf{X}\varphi_1$, then $\mathrm{el}(\varphi) = \{\forall\mathbf{X}\varphi_1\} \cup \mathrm{el}(\varphi_1)$.
(b) If $\varphi = \forall(\varphi_1\mathbf{U}\varphi_2)$, then $\mathrm{el}(\varphi) = \{\forall\mathbf{X}\ \mathit{false}, \forall\mathbf{X}\forall(\varphi_1\mathbf{U}\varphi_2)\} \cup \mathrm{el}(\varphi_1) \cup \mathrm{el}(\varphi_2)$.
(c) If $\varphi = \forall(\varphi_1\mathbf{V}\varphi_2)$, then $\mathrm{el}(\varphi) = \{\forall\mathbf{X}\ \mathit{false}, \forall\mathbf{X}\forall(\varphi_1\mathbf{V}\varphi_2)\} \cup \mathrm{el}(\varphi_1) \cup \mathrm{el}(\varphi_2)$.

The special elementary subformula $\forall\mathbf{X}$ *false* denotes the nonexistence of a fair path; $s \models \forall\mathbf{X}$ *false* indicates that no fair path begins at s; $\mathrm{el}(\varphi)$ contains propositions and formulas with $\forall\mathbf{X}$ as their top-level operator. Subsets of $\mathrm{el}(\varphi)$ are states of the tableau defined below. Thus, states of the tableau represent possible valuations for the atomic propositions, plus information about what subformulas should be true in the next state. This latter information is used to determine the legal transitions between states. As we will see later, every elementary formula that is part of a state is in fact true at that state. The map Φ defined below uses the elementary subformulas to assign a set of states to every subformula. Thus, it can be thought of as a type of satisfaction relation, mapping subformulas to the sets of states where they should be true.

Definition 18. (*tableau of a formula*) The tableau of ψ, denoted $\mathcal{T}(\psi)$, is the structure $\langle S, S_0, \mathcal{A}, \mathcal{L}, R, \mathcal{F}\rangle$ defined as follows.

(1) $S = 2^{\mathrm{el}(\psi)}$.
(2) $S_0 = \Phi(\psi)$, where Φ is the map from $\mathrm{el}(\psi) \cup \mathrm{sub}(\psi) \cup \{\mathit{true}, \mathit{false}\}$ to 2^S defined by the following equations.
(a) $\Phi(\mathit{true}) = S$; $\Phi(\mathit{false}) = \emptyset$. If $\varphi \in \mathrm{el}(\psi)$, then $\Phi(\varphi) = \{s \mid \varphi \in s\}$. If $\varphi = \neg\varphi_1$, then $\Phi(\varphi) = S \setminus \Phi(\varphi_1)$.
(b) If $\varphi = \varphi_1 \wedge \varphi_2$, then $\Phi(\varphi) = \Phi(\varphi_1) \cap \Phi(\varphi_2)$. If $\varphi = \varphi_1 \vee \varphi_2$, then $\Phi(\varphi) = \Phi(\varphi_1) \cup \Phi(\varphi_2)$.
(c) (i) If $\varphi = \forall(\varphi_1\mathbf{U}\varphi_2)$, then $\Phi(\varphi) = (\Phi(\varphi_2) \cup (\Phi(\varphi_1) \cap \Phi(\forall\mathbf{X}\varphi))) \cup \Phi(\forall\mathbf{X}\ \mathit{false})$.
(ii) If $\varphi = \forall(\varphi_1\mathbf{V}\varphi_2)$, then $\Phi(\varphi) = (\Phi(\varphi_2) \cap (\Phi(\varphi_1) \cup \Phi(\forall\mathbf{X}\varphi))) \cup \Phi(\forall\mathbf{X}\ \mathit{false})$.
(3) $\mathcal{A} = \{p \mid p \in \mathrm{el}(\psi)\}$.
(4) $\mathcal{L}(s) = \{p \mid p \in s\}$.
(5) $R(s,t)$ iff for each formula $\forall\mathbf{X}\varphi$ in $\mathrm{el}(\psi)$, $\forall\mathbf{X}\varphi \in s$ implies $t \in \Phi(\varphi)$.
(6) $\mathcal{F} = \{(\Phi(\forall\mathbf{X}\forall(\varphi_1\mathbf{U}\varphi_2)), \Phi(\varphi_2)) \mid \forall\mathbf{X}\forall(\varphi_1\mathbf{U}\varphi_2) \in \mathrm{el}(\psi)\}$.

LEMMA 5. *For all subformulas φ of ψ, if $s \in \Phi(\varphi)$, then $s \models \varphi$.*

PROOF. The proof proceeds by induction on the structure of φ.

(1) If $\varphi = \mathit{true}$, then $\Phi(\varphi) = S$, and every state satisfies *true*. If $\varphi = \mathit{false}$, then $\Phi(\varphi) = \emptyset$, so the result is trivial. If $\varphi = p$, an atomic proposition, then $\Phi(\varphi) = \{s \mid p \in s\}$. But $\mathcal{L}(s) = \{q \mid q \in s\}$, and so $p \in \mathcal{L}(s)$ and $s \models p$. If $\varphi = \neg p$, a negated atomic proposition, then $\Phi(\varphi) = S \setminus \{s \mid p \in s\}$. Since $\mathcal{L}(s) = \{q \mid q \in s\}$, we have that $p \notin \mathcal{L}(s)$, and so $s \models \neg p$.

(2) If $\varphi = \varphi_1 \wedge \varphi_2$, then $\Phi(\varphi) = \Phi(\varphi_1) \cap \Phi(\varphi_2)$, and hence $s \in \Phi(\varphi_1)$ and $s \in \Phi(\varphi_2)$. By the induction hypothesis, $s \models \varphi_1$ and $s \models \varphi_2$, which implies $s \models \varphi_1 \wedge \varphi_2$. The case where $\varphi = \varphi_1 \vee \varphi_2$ is similar.

(3) (a) If $\varphi = \forall \mathbf{X} \varphi_1$, then $\Phi(\varphi) = \{s | \forall \mathbf{X}(\varphi_1) \in s\}$, and so $\forall \mathbf{X}(\varphi_1) \in s$. Suppose $R(s,t)$. By the definition of R, we have $t \in \Phi(\varphi_1)$, and then the induction hypothesis implies $t \models \varphi_1$. Since t was chosen arbitrarily, any fair path from s satisfies φ_1 at its second state, and hence $s \models \forall \mathbf{X}(\varphi_1)$.

(b) If $\varphi = \forall(\varphi_1 \mathbf{U} \varphi_2)$, then $\Phi(\varphi) = (\Phi(\varphi_2) \cap (\Phi(\varphi_1) \cap \Phi(\forall \mathbf{X} \varphi))) \cup \Phi(\forall \boldsymbol{X} \mathit{false})$. Let t be any state in $\Phi(\varphi)$. Then either

(i) $t \in \Phi(\forall \mathbf{X}\, \mathit{false})$, in which case t has no successors and $t \models \varphi$ trivially, or

(ii) $t \in \Phi(\varphi_2)$, in which case the induction hypothesis implies $t \models \varphi_2$, or

(iii) $t \in \Phi(\varphi_1) \cap \Phi(\forall \boldsymbol{X} \varphi)$. In this case, the induction hypothesis implies $t \models \varphi_1$. By the definition of R, we also know that if $R(t,u)$, then $u \in \Phi(\varphi)$.

Let $s = s_0$, and consider a fair path $\pi = s_0 s_1 s_2 \ldots$ from s. Note that no state on this path can satisfy the first condition above. There are two cases to consider.

(i) There is some j such that $s_j \models \varphi_2$. Let s_i be the first such state on the path. By the above, for every $j < i$, $s_j \models \varphi_1$. Hence the path satisfies $\varphi_1 \mathbf{U} \varphi_2$.

(ii) For every j, $s_j \not\models \varphi_2$. Then the above implies that for every j, $s_j \in \Phi(\forall \mathbf{X} \varphi)$. By the induction hypothesis, we know that each s_j is not in $\Phi(\varphi_2)$. But then $\inf(\pi) \cap \Phi(\forall \mathbf{S} \varphi) \neq \emptyset$ and $\inf(\pi) \cap \Phi(\varphi_2) = \emptyset$. By the definition of $\mathscr{F}$, this contradicts the fact the π is fair, and so this case is impossible.

Thus $s \models \forall(\varphi_1 \mathbf{U} \varphi_2)$.

(c) If $\varphi = \forall(\varphi_1 \mathbf{V} \varphi_2)$, then $\Phi(\varphi) = (\Phi(\varphi_2) \cap (\Phi(\varphi_1) \cup \Phi(\forall \mathbf{X} \varphi))) \cup \Phi(\forall \mathbf{X}\, \mathit{false})$. If t is any state in $\Phi(\varphi)$, then either

(i) $t \in \Phi(\forall \mathbf{X}\, \mathit{false})$, in which case t has no successors and $t \models \varphi$ trivially, or

(ii) $t \in \Phi(\varphi_2)$. In this case, we also have either $t \in \Phi(\varphi_1)$ or for every u such that $R(t,u)$, $u \in \Phi(\varphi)$.

Let $s = s_0$, and let $\pi = s_0 s_1 s_2 \ldots$ be a fair path from s. Note that no s_i can satisfy the first condition above. If s_i is such that for all $j < i$, $s_j \not\models \varphi_1$, then the induction hypothesis implies that $s_j \notin \Phi(\varphi_1)$. Hence $s_j \in \Phi(\varphi_2)$, and also $s_i \in \Phi(\varphi_2)$; then the induction hypothesis implies for all $j \leq i$, $s_j \models \varphi_2$. Thus the path satisfies $\varphi_1 \mathbf{V} \varphi_2$, and hence we have $s \models \forall(\varphi_1 \mathbf{V} \varphi_2)$. □

Now let $M = \mathscr{T}(\psi)$, and fix a structure M'. The two lemmas below suggest a simulation relation from any structure M' that satisfies ψ to $\mathscr{T}(\psi)$. This is not a standard requirement for a tableau and stems from the kind of reasoning we wish to apply.

LEMMA 6. *Define a relation $H \subseteq S' \times S$ by*

$$H = \{(s', s) | s = \{\varphi | \varphi \in \mathrm{el}(\psi), s' \models \varphi\}\}.$$

If $H(s', s)$, then for every subformula or elementary formula φ of ψ, $s' \models \varphi$ implies $s \in \Phi(\varphi)$.

PROOF. The proof proceeds by induction on the structure of φ, where the base cases for the induction are the elementary subformulas of ψ, plus *true* and *false*.

(1) If $\varphi = true$, then $\Phi(\varphi) = S$, so the result is trivial. If $\varphi = false$, then s' cannot satisfy φ. If $\varphi \in \mathrm{el}(\psi)$, then by the definition of H, $s' \models \varphi$ implies $\varphi \in s$. Now $\Phi(\varphi) = \{s | \varphi \in s\}$, so $s \in \Phi(\varphi)$.
(2) If $\varphi = \neg p$, a negated atomic proposition, then $s' \models \varphi$ implies $p \notin s$. Since $\Phi(\varphi) = S \setminus \{s | p \in S\}$, $s \in \Phi(\varphi)$.
(3) If $\varphi = \varphi_1 \wedge \varphi_2$, then $\Phi(\varphi) = \Phi(\varphi_1) \cap \Phi(\varphi_2)$. We have $s' \models \varphi$ implies $s' \models \varphi_1$ and $s' \models \varphi_2$. By the induction hypothesis, $s \in \Phi(\varphi_1)$ and $s \in \Phi(\varphi_2)$, and so $s \in \Phi(\varphi_1) \cap \Phi(\varphi_2)$. The case when $\varphi = \varphi_1 \vee \varphi_2$ is similar.
(4) If $\varphi = \forall(\varphi_1 \mathbf{U} \varphi_2)$, then $\Phi(\varphi) = (\Phi(\varphi_2) \cup (\Phi(\varphi_1) \cap \Phi(\forall\mathbf{X}\varphi))) \cup \Phi(\forall\mathbf{X}$ *false*$)$. Given $s' \models \varphi$, there are three cases.
   (a) If no fair paths start at s', then $s' \models \forall\mathbf{X}$ *false*. The induction hypothesis implies $s \in \Phi(\forall\mathbf{X}$ *false*$)$, and so $s \in \Phi(\varphi)$.
   (b) If $s' \models \varphi_2$, then by the induction hypothesis, $s \in \Phi(\varphi_2)$, and so $s \in \Phi(\varphi)$.
   (c) Otherwise, $s' \models \varphi_1$ and $s' \models \forall\mathbf{X}\varphi$. By the induction hypothesis, $s \in \Phi(\varphi_1)$ and $s \in \Phi(\forall\mathbf{X}\varphi)$ (since $\forall\mathbf{X}\varphi \in \mathrm{el}(\psi)$). Hence $s \in \Phi(\varphi)$.

   In all cases, $s \in \Phi(\forall(\varphi_1 \mathbf{U} \varphi_2))$.
(5) If $\varphi = \forall(\varphi_1 \mathbf{V} \varphi_2)$, then $\Phi(\varphi) = (\Phi(\varphi_2) \cap (\Phi(\varphi_1) \cup \Phi(\forall\mathbf{X}\varphi))) \cup \Phi(\forall\mathbf{X}$ *false*$)$. Since $s' \models \varphi$, either
   (a) no fair paths start at s', in which case $s' \models \forall\mathbf{X}$ *false* and the induction hypothesis implies $s \in \Phi(\varphi)$, or
   (b) $s' \models \varphi_2$; and so by the induction hypothesis, $s \in \Phi(\varphi_2)$. Also, either $s' \models \varphi_1$ or $s' \models \forall\mathbf{X}\varphi$. Applying the induction hypothesis again, either $s \in \Phi(\varphi_1)$ or $s \in \Phi(\forall\mathbf{X}\varphi)$. In both cases, $s \in \Phi(\varphi)$.

   Thus in all cases, $s \in \Phi(\forall(\varphi_1 \mathbf{V} \varphi_2))$. □

LEMMA 7. *The relation H given above is a simulation relation.*

PROOF. Note that for every state s' of M', there is a (single) state s of M such that $H(s', s)$. Let A be the set of atomic propositions for M, and assume $H(s', s)$. We have $\mathscr{L}(s) = \{p | p \in s\}$. From the definition of H, $p \in s$ implies $s' \models p$. Further, If $s' \models p$ and $p \in A$, then $p \in \mathrm{el}(\psi)$, and hence $p \in s$, $p \in \mathscr{L}(s)$. Thus we find $\mathscr{L}'(s') \cap A = \mathscr{L}(s)$.

Let $s'_0 = s'$, and suppose $\pi' = s'_0 s'_1 s'_2 \ldots$ is a fair path from s'. Let $\forall\mathbf{X}\varphi_1$, $\forall\mathbf{X}\varphi_2, \ldots, \forall\mathbf{X}\varphi_n$ be all the formulas of the form $\forall\mathbf{X}\varphi$ in $\mathrm{el}(\psi)$ which s' satisfies. Then we have $s'_1 \models \varphi_1, s'_1 \models \varphi_2, \ldots, s'_1 \models \varphi_n$. Let s_1 be the state of M related to s'_1 by H. By the previous lemma, $s_1 \in \Phi(\varphi_1), s_1 \in \Phi(\varphi_2), \ldots, s_1 \in \Phi(\varphi_n)$. Now by the definition of H, the formulas of the form $\forall\mathbf{X}\varphi$ in s must be exactly $\forall\mathbf{X}\varphi_1, \forall\mathbf{X}\varphi_2, \ldots, \forall\mathbf{X}\varphi_n$. Then from the definition of R, we see that $R(s, s_1)$. Since $H(s'_1, s_1)$, we can continue the process. Defining $s_0 = s$, we get a sequence of states $\pi = s_0 s_1 s_2 \ldots$ starting at s such that $H(s'_i, s_i)$ for all i. To complete the proof, we must show that this sequence is fair.

Assume that π is not fair. Looking at $\mathcal{F}$, we see that there must be some elementary subformula $\forall\mathbf{X}\forall(\varphi_a\mathbf{U}\varphi_b)$ such that $\mathrm{inf}(\pi) \cap \Phi(\forall\mathbf{X}\forall(\varphi_a\mathbf{U}\varphi_b)) \neq \emptyset$ and $\mathrm{inf}(\pi) \cap \Phi(\varphi_b) = \emptyset$. Consider one of the states s_i; $s_i \in \Phi(\forall\mathbf{X}\forall(\varphi_a\mathbf{U}\varphi_b))$ iff $\forall\mathbf{X}\forall(\varphi_a\mathbf{U}\varphi_b) \in s_i$, and then the definition of H implies $s'_i \models \forall\mathbf{X}\forall(\varphi_a\mathbf{U}\varphi_b)$. Additionally, the previous lemma implies that if $s_i \notin \Phi(\varphi_b)$, then $s'_i \not\models \varphi_b$. Choose i so that $s_i \in \Phi(\forall\mathbf{X}\forall(\varphi_a\mathbf{U}\varphi_b))$ and so that for all $j \geq i$, $s_j \notin \Phi(\varphi_b)$. Then $s'_i s'_{i+1} \ldots$ is a fair path in M' starting at s'_i, and every state on this path satisfies $\neg\varphi_b$. But $s'_i \models \forall\mathbf{X}\forall(\varphi_a\mathbf{U}\varphi_b)$, a contradiction. Hence π is in fact a fair path in M. □

THEOREM 5. *$M' \models \psi$ iff $M' \preceq \mathcal{T}(\psi)$.*

PROOF. Suppose $M' \preceq \mathcal{T}(\psi)$. By Lemma 5 and the definition of the tableau, every initial state of $\mathcal{T}(\psi)$ satisfies ψ, i.e., $\mathcal{T}(\psi) \models \psi$. Then since $M' \preceq \mathcal{T}(\psi)$, $M' \models \psi$.

If $M' \models \psi$, then by definition, every $s'_0 \in S'_0$ satisfies ψ. By the definition of H, every such s'_0 is paired with a (unique) s_0. Lemma 6 implies that $s_0 \in \Phi(\psi)$, and by the definition of the tableau, $s_0 \in S_0$. By Lemma 7, H is a simulation relation, so $M' \preceq \mathcal{T}(\psi)$. □

The tableau construction can also be used to reason about formulas. We are typically interested in whether every model of a formula φ is also a model of some other formula ψ. Let $\varphi \models \psi$ denote this semantic relation.

Proposition 1. $\varphi \models \psi$ iff $\mathcal{T}(\varphi) \models \psi$.

PROOF. If $\varphi \models \psi$, then every model of φ, in particular $\mathcal{T}(\varphi)$, is also a model of ψ. Assume $\mathcal{T}(\varphi) \models \psi$, and let $M \models \varphi$. By the previous theorem, $M \preceq \mathcal{T}(\varphi)$. Since $\mathcal{T}(\varphi) \models \psi$, $\mathcal{T}(\varphi) \preceq \mathcal{T}(\psi)$. Hence $M \preceq \mathcal{T}(\psi)$, i.e., $M \models \psi$. □

We will sometimes extend the set of elementary formulas of a formula by adding additional atomic propositions. For example, if we wished to check whether *true* implied p, we would extend the set of atomic propositions for *true* to include p (another way to view this is to imagine rewriting *true* as *true* $\wedge (p \vee \neg p)$. The formula ψ has a nontrivial model iff it is not the case that $\psi \models \forall\mathbf{X}$ *false*; ψ is true in every model iff *true* $\models \psi$.

7. CHECKING FOR SIMULATION

In this section, we discuss the problem of determining whether there exists a simulation relation between two structures M and M'. Our goal is to efficiently determine if $M \preceq M'$. First note that if H_1 and H_2 are simulation relations, then $H_1 \cup H_2$ is a simulation relation. Also, $\emptyset$ is trivially a simulation relation. These facts imply that there is a maximal simulation relation under set inclusion. This maximal simulation relation is in fact $\preceq$, and this is what we will actually compute.

We also note the following facts.

(1) If s is a state of M and no fair paths start at s, then s is simulated by exactly those states s' in M' for which $\mathscr{L}(s) \cap \mathscr{A}' = \mathscr{L}'(s')$.

(2) If s' is a state of M' and no fair paths start at s', then s' simulates exactly those states s in M which are the start of no fair path and for which $\mathscr{L}(s) \cap \mathscr{A}' = \mathscr{L}'(s')$.

States which are the start of no fair path can be detected in polynomial time [Emerson and Lei 1986] and eliminated in a preprocessing step. Hence, without loss of generality, we can assume that every state in M and M' is the start of some fair path. We now describe polynomial-time algorithms for checking the preorder in several important special cases. The question of how to check the preorder in the general case is left open.

The first case is similar to Milner simulation [Milner 1971] as computed within the Concurrency Workbench [Cleaveland et al. 1989]. Suppose that M' has a trivial acceptance condition, i.e., $\mathscr{F}' = \emptyset$.

Definition 19. Define a sequence of relations H_i as follows.

(1) $H_0 = \{(s, s') | \mathscr{L}(s) \cap \mathscr{A}' = \mathscr{L}'(s')\}$.

(2) $H_{i+1} = H_i \cap \{(s, s') | \forall t[R(s,t) \rightarrow \exists t'(R'(s',t') \wedge H_i(t,t'))]\}$.

Define H_ω to be the first H_i such that $H_i = H_{i+1}$ (such an i exists since $H_{j+1} \subseteq H_j$ for all j and each H_j is finite).

THEOREM 6. *For every $s \in S$ and $s' \in S'$, $s \preceq s'$ iff $H_\omega(s, s')$.*

PROOF. We first note that H_ω is the greatest fixed point of the equation

$$H = H \cap \{(s, s') | \mathscr{L}(s) \cap \mathscr{A}' = \mathscr{L}'(s)' \wedge \forall t[R(s,t) \rightarrow \exists t'(R'(s',t') \wedge H(t,t'))]\}.$$

Suppose s and s' are states such that $H_\omega(s, s')$. We have $\mathscr{L}(s) \cap \mathscr{A}' = \mathscr{L}'(s')$ immediately. Let $s_0 = s$ and $s'_0 = s'$, and assume $\pi = s_0 s_1 \ldots$ is a fair path starting from s. From the above equation, there exists a state s'_1 such that $R'(s'_0, s'_1)$ and $H_\omega(s_1, s'_1)$. Continuing in this fashion, we find a path $s'_0 s'_1 \ldots$ starting from s' such that $H_\omega(s_i, s'_i)$ for all i. Since $\mathscr{F}' = \emptyset$, this path is fair. Hence H_ω is a simulation relation from s to s', i.e., $s \preceq s'$.

To show that $s \preceq s'$ implies $H_\omega(s, s')$, we show that any simulation relation H is a fixed point of the above equation. Since H_ω is the greatest fixed point, we will have $H \subseteq H_\omega$. Hence if there is some simulation relation H such that $H(s, s')$, then $H_\omega(s, s')$. It is enough to show that H is a subset of the set

$$\{(s, s') | \mathscr{L}(s) \cap \mathscr{A}' = \mathscr{L}'(s)' \wedge \forall t[R(s,t) \rightarrow \exists t'(R'(s',t') \wedge H(t,t'))]\}.$$

If $H(s, s')$, then we have $\mathscr{L}(s) \cap \mathscr{A}' = \mathscr{L}'(s')$. If $R(s,t)$, then by our earlier assumption, there exists a fair path from t. Hence, letting $s_0 = s$ and $s_1 = t$, there is some fair path $s_0 s_1 \ldots$ from s through t. Since $H(s, s')$, there exists a fair path $s'_0 s'_1 \dots$ from $s' = s'_0$ such that $H(s_i, s'_i)$ for all i. Now if we take $t' = s'_1$, we see that (s, s') is in the above set. □

We note that $H_\omega = H_i$ for some i which is at most $|S| \cdot |S'|$. Each H_{j+1} can also be computed in polynomial time from H_j; hence H_ω can be computed in polynomial time.

Another important case is when M' is deterministic, i.e., if $R'(s', t')$ and $R'(s', u')$, then $\mathscr{L}'(t') \neq \mathscr{L}'(u')$. For this case, we show below that $s \preceq s'$ iff the language of s is contained in the language of s', where the language for a state s is the set of sequences of labelings which occur along the fair paths starting at s. More formally, see Definition 20.

Definition 20. Let M be a structure, s be a state in M, and $\mathscr{B} \subseteq \mathscr{A}$. $\text{Lan}_{\mathscr{B}}(s)$, the language of s in M restricted to $\mathscr{B}$, is the set of all sequences $l_0 l_1 \dots$ over $2^{\mathscr{B}}$, such that there is a fair path $\pi = s_0 s_1 \dots$ in M with $s_0 = s$ for which $l_i = \mathscr{L}(s_i) \cap \mathscr{B}$, for every $i \geq 0$.

Clearly, if $s \preceq s'$ then $Lan_{\mathscr{A}'}(s) \subseteq Lan_{\mathscr{A}'}(s')$. Below we show that if M' is deterministic then the converse is also true.

Definition 21. Define a relation $H \subseteq S \times S'$ as follows.

$$H = \{(s, s') | Lan_{\mathscr{A}'}(s) \subseteq Lan_{\mathscr{A}'}(s')\}$$

THEOREM 7. *If M' is deterministic then H is a simulation relation.*

PROOF. Suppose s and s' are states such that $H(s, s')$. Then, $\text{Lan}_{\mathscr{A}'}(s) \subseteq Lan_{\mathscr{A}'}(s')$ which immediately implies that $\mathscr{L}(s) \cap \mathscr{A}' = \mathscr{L}'(s')$. Let $s_0 = s$ and $s'_0 = s'$, and assume $\pi = s_0 s_1 \dots$ is a fair path starting from s. Then, there is a fair path $\pi' = s'_0 s'_1 \dots$ from s' such that for every $i \geq 0$, $\mathscr{L}(s_i) \cap \mathscr{A}' = \mathscr{L}'(s'_i)$. Assume $H(\pi, \pi')$ does not hold, and let $k \in \mathscr{N}$ be the smallest such that $H(s_k, s'_k)$ but not $H(s_{k+1}, s'_{k+1})$. $Lan_{\mathscr{A}'}(s'_k) \subseteq Lan_{\mathscr{A}'}(s'_k)$, therefore

$$\bigcup_{R(s_k, t)} Lan_{\mathscr{A}'}(t) \subseteq \bigcup_{R(s'_k, t')} Lan_{\mathscr{A}'}(t').$$

Moreover,

$$Lan_{\mathscr{A}'}(s_{k+1}) \subseteq \bigcup_{R(s'_k, t') \wedge \mathscr{L}(s_{k+1}) \cap \mathscr{A}' = \mathscr{L}'(t')} Lan_{\mathscr{A}'}(t').$$

s'_{k+1} is a state for which $R(s'_k, s'_{k+1})$ and $\mathscr{L}(s_{k+1}) \cap \mathscr{A}' = \mathscr{L}'(s'_{k+1})$ hold. Since M' is deterministic, s'_{k+1} is the only successor of s'_k that has this property. Thus, $\text{Lan}_{\mathscr{A}'}(s_{k+1}) \subseteq Lan_{\mathscr{A}'}(s'_{k+1})$, and hence $H(s_{k+1}, s'_{k+1})$, contradicting the assumption. Consequently, there is $H(\pi, \pi')$, and H is a simulation relation. □

By the theorem above, if M' is deterministic, in order to check that $s \preceq s'$ it is sufficient to check language inclusion between the languages of s and s'. This relation can be checked in polynomial time using the techniques of Clarke et al. [1990].

Finally, if M' is the result of a tableau construction, say $M' = \mathcal{T}(\psi)$, then as shown in the previous section, checking whether $M \preceq M'$ reduces to the problem of checking whether $M \models \psi$.

8. AN EXAMPLE

We have implemented a BDD-based model checker based on the theory developed in the previous sections. The model checker is written in a combination of T (Yale's dialect of Scheme) and C. It includes facilities for model checking, temporal reasoning (via the tableau construction), and checking for simulation. To illustrate the system, we use the controller of a simple CPU as an example. The controller is written in a state machine description language called CSML [Clarke et al. 1989b] which is compiled into Moore machines. We give only a brief description of the CPU here; Clarke et al. give details. The CPU is a simple stack-based machine, i.e., part of the CPU's memory contains a stack from which instruction operands are popped and onto which results are pushed. There are two parts to the CPU controller. The first part is called the access unit and is responsible for all the CPU's memory references. The second part, called the execution unit, interprets the instructions and controls the arithmetic unit, shifter, etc. These two parts operate in parallel. The access unit and execution unit communicate via a small number of signals. Three of the signals, *push*, *pop*, and *fetch*, are inputs of the access unit and indicate that the execution unit wants to push or pop something from the stack or to get the next instruction. For each of these signals there is a corresponding ready output from the access unit. The execution unit must wait for the appropriate ready signal before proceeding. One additional signal, *branch*, is asserted by the execution unit when it wants to jump to a new program location. The access unit also has signals that it asserts to issue a memory read or write, and an input that tells it when the memory has finished servicing a request.

In order to increase performance, the access unit attempts to keep the value on the top of the stack in a special register called the TS ("top-of-stack") register. The goal is to keep the execution unit from having to wait for the memory. For example, when the TS register contains valid data, a pop operation can proceed immediately. Additionally, when a value is pushed on the stack, it is moved into this register and copied to memory at some later point. The access unit also loads instructions into a queue when possible so that fetches do not require waiting for the memory. This queue is flushed whenever the CPU branches.

Clarke, et al. [1989b] gave a number of correctness conditions for the controller. We demonstrate here how these formulas can be verified in a compositional fashion. From the form of the conditions, we divide them into three classes. The first class consists of simple safety properties of the access unit. For example, one of these formulas is

$$\forall \mathbf{G}(\mathit{SPtoMemAddr} \rightarrow \mathit{TSLoad} \vee \mathit{TSStore}),$$

which states that if the access unit outputs the top-of-stack pointer as a memory address, then it is either loading the TS register from memory or storing it to memory. The model checker verified that each of these properties held for the access unit alone. Hence, they hold in any system containing the access unit.

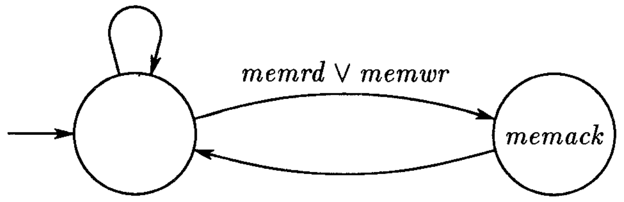
The conditions in the second class are slightly more complex. These properties are safety properties which specify what sequences of operations are allowed. For example, one condition is

$$\forall\mathbf{G}(\mathit{pushed} \rightarrow \forall\mathbf{X}\forall(\mathit{TSStored} \vee \mathit{popped}\ \mathbf{V} \neg(\mathit{pushed} \vee \mathit{TSLoad}))).$$

Here *pushed* is an abbreviation for $\mathit{push} \wedge \mathit{pushrdy}$, and *popped* abbreviates $\mathit{pop} \wedge \mathit{poprdy}$. The formula asserts that if a push operation is completed, then another push cannot be completed, and the access unit cannot attempt to load the TS register from memory until either a pop occurs or until the TS register is stored to memory. In other words, once the TS register contains a value which needs to be pushed on the stack, the CPU cannot do anything that would destroy this value until the value is either used or successfully stored in memory. Since all of the properties in this class essentially specify when the access unit may assert its ready signals, it is tempting to check whether they hold for the access unit alone also. This is not possible, however, because the properties also depend on how the memory acknowledgment signal behaves. To verify these properties, we made a simple model of the memory (see Figure 1). For conciseness, the figure shows a Moore machine; the actual model used is obtained by adding the fairness constraint shown in the figure to the structure corresponding to this Moore machine. All of the properties in this class except for one turn out to be true in the system composed of the access unit and this model of the memory. The exception is an analog to the previous formula that deals with what occurs after a pop. The counterexample produced by the model checker for this formula showed that the formula was false because a push and a pop could be requested simultaneously. When we examined the access unit, we saw that it had been designed assuming that these operations would not be requested at the same time. The formula turns out to be true with the additional assumption $\forall\mathbf{G}(\neg\,\mathit{push} \vee \neg\,\mathit{pop})$. The model checker verified this by building the tableau for this assumption, composing it with the access unit and memory model, and checking the formula.

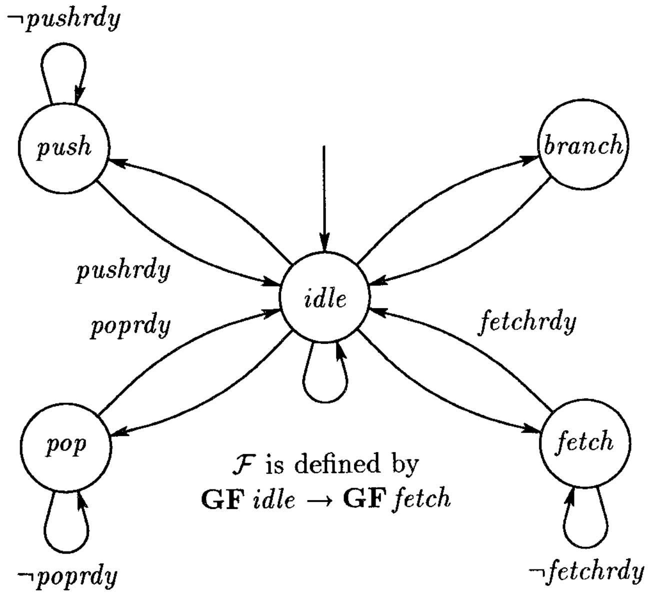
The final class of criteria consists of a single liveness property: $\forall\mathbf{G}\forall\mathbf{F}(\mathit{fetch} \wedge \mathit{fetchrdy})$. This formula states that the CPU (in the context of a memory such as that modeled by Figure 1) always fetches another instruction. We demonstrate two different ways of verifying this property.

One way is to observe that for this formula to be true, it must obviously be the case that the memory responds to requests eventually and that the execution unit does not execute infinite sequences of pushes, pops, and branches. Our memory model already has a fairness constraint ensuring the first of these, but there is nothing to guarantee the second. We can take care of this by using a simple model of the execution unit (see Figure 2). Again, the actual model is the structure derived from the Moore machine, plus the indicated fairness constraint. The output *idle* in this figure is an abbreviation for $\neg(\mathit{push} \vee \mathit{pop} \vee \mathit{fetch} \vee \mathit{branch})$. The model checker verified that the



$\mathcal{F}$ is defined by
$\mathbf{GF}(memrd \vee memwr \rightarrow memack)$

Fig. 1. Memory abstraction.



$\mathcal{F}$ is defined by
$\mathbf{GF}\, idle \rightarrow \mathbf{GF}\, fetch$

Fig. 2. Execution unit abstraction.

access unit plus the models of the execution unit and the memory satisfied the above formula. It also verified that there was a simulation relation between the (structure for the) actual execution unit and the model. Thus, we can conclude that this formula holds in the final system provided there is a simulation relation from the actual memory to our model. We also checked that the execution unit model satisfied the assumption $\forall\mathbf{G}(\neg\, push \vee \neg\, pop)$ used above. Since there is a simulation relation from the execution unit to the model, we know that the execution unit must satisfy this assumption also. This final step allows us to conclude that the composition of the access and execution units satisfied the entire specification provided the memory is simulated by the model we used.

We can also verify the final property using a series of ∀CTL assumptions. The idea will be to check the property for the execution unit. In order for the formula to be true, the access unit must eventually respond to push and pop requests and must fill the instruction queue when appropriate. We can only guarantee that the access unit meets these conditions if we know that the execution unit does not try to do two operations at once and that it will not remove a request before the corresponding operation can complete. We begin with these properties.

$$\forall\mathbf{G}(\neg(\mathit{fetch} \wedge \mathit{push}) \wedge \neg(\mathit{fetch} \wedge \mathit{pop}) \wedge \cdots \wedge \neg(\mathit{pop} \wedge \mathit{branch})) \quad (1)$$

$$\forall\mathbf{G}(\mathit{push} \rightarrow \forall(\mathit{pushed}\,\mathbf{V}\mathit{push})) \quad (2)$$

$$\forall\mathbf{G}(\mathit{pop} \rightarrow \forall(\mathit{popped}\,\mathbf{V}\mathit{pop})) \quad (3)$$

The first of these specifies that every pair of operations the execution unit can perform are mutually exclusive. The other two formulas state that if the execution unit makes a push or pop request, then it does not deassert the request until the operation completes. The model checker verified that these properties hold in the execution unit alone and (using the tableau construction) that the first property implies the assumption $\forall\mathbf{G}(\neg\mathit{push} \vee \neg\mathit{pop})$ used above. Now using formulas (1) and (2) as assumptions, we checked that the system composed of the access unit and the memory model satisfied the formula

$$\forall\mathbf{G}(\mathit{push} \rightarrow \forall(\mathit{push}\,\mathbf{U}\mathit{pushed})). \quad (4)$$

This specification states that every push operation will be completed. Similarly, using formulas (1) and (3) as assumptions, we verified

$$\forall\mathbf{G}(\mathit{pop} \rightarrow \forall(\mathit{pop}\,\mathbf{U}\mathit{popped})). \quad (5)$$

The system composed of the access unit and the memory model also satisfies the formula $\forall\mathbf{G}\forall\mathbf{F}(\mathit{fetchrdy} \vee \mathit{branch})$ (at any point, either the access unit will eventually fill the instruction queue or a branch will occur). Finally, using this formula and formulas (4) and (5) as assumptions, the model checker verified that the execution unit satisfies $\forall\mathbf{G}\forall\mathbf{F}(\mathit{fetch} \wedge \mathit{fetchrdy})$. (Again, to complete the verification we would have to demonstrate a simulation relation between the actual memory and our model of it.)

9. CONCLUSION

We have identified a subset, ∀CTL*, of CTL* which is appropriate for compositional reasoning. For this subset, satisfaction is preserved under composition; hence a standard model-checking algorithm can be used to answer the question: Is a formula true for all systems containing a specified component? We have also proposed a preorder $\preceq$ which is appropriate for ∀CTL*. The preorder captures the relation between a component and a system containing that component. It provides the basis for using an assume-guarantee style of reasoning with the logic. Assumptions which are given as structures are discharged by checking the preorder. We have given a tableau construction for the ∀CTL subset of ∀CTL*. Satisfaction of a ∀CTL formula corresponds to being below the tableau of the formula in the preorder. The construction makes it possible to use ∀CTL formulas as assumptions and to do temporal reasoning. ∀CTL also has an efficient model-checking algorithm. We have implemented a symbolic verification system based on these results and have used it to verify some nontrivial systems in a compositional fashion.

There are several directions for future work. Intuitively, the ∀CTL* subset of CTL* should be maximal in the sense that any formula for which satisfaction is preserved under composition should be equivalent to a formula of ∀CTL*, but we have not proved this. Another idea is to look at different logics with the same flavor, such as ∀CTL* extended with automata operators or the μ-calculus with only $[\cdot]$ modalities. It would also be interesting to try to extend the tableau construction of Section 6 to all of ∀CTL*. In order to accomplish this however, it will almost certainly be necessary to use a more complex type of structure than that given in Definition 2. Another question is whether it is possible to apply our ideas to branching-time logics with existential path quantifiers. For example, is there a reasonable algorithm which will determine whether a CTL formula is true in all systems containing a given component? It is fairly easy to come up with algorithms which are sound, but completeness seems more difficult to achieve. We also wish to examine the problem of efficiently checking the preorder for arbitrary structures. Finally, it is essential to try to apply the compositional-reasoning methods we have considered to more complex systems in order to evaluate the techniques.

REFERENCES

BEN-ARI, M., MANNA, Z., AND PNUELI, A. 1983. The temporal logic of branching time. *Acta Informatica 20*, 207–226.

BURCH, J. R., CLARKE, E. M., MCMILLAN, K. L., DILL, D. L., AND HWANG, J. 1990. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the 5th Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif.

CLARKE, E. M., AND EMERSON, E. A. 1981. Synthesis of synchronization skeletons for branching time temporal logic. In *Logic of Programs: Workshop* (Yorktown Heights, NY, May). Lecture Notes in Computer Science, vol. 131. Springer-Verlag, New York.

CLARKE, E. M., DRAGHICESCU, I. A., AND KURSHAN, R. P. 1990. A unified approach for showing language containment and equivalence between various types of ω-automata. In *Proceedings of the 15th Colloquium on Trees in Algebra and Programming*. Lecture Notes in Computer Science, vol. 407. Springer-Verlag, New York.

CLARKE, E. M., EMERSON, E. A., AND SISTLA, A. P. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst. 8*, 2, 244–263.

CLARKE, E. M., LONG, D. E., AND MCMILLAN, K. L. 1989a. Compositional model checking. In *Proceedings of the 4th Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif.

CLARKE, E. M., LONG, D. E., AND MCMILLAN, K. L 1989b. A language for compositional specification and verification of finite state hardware controllers. In *Proceedings of the 9th International Symposium on Computer Hardware Description Languages and their Applications*. North-Holland, Amsterdam.

CLEAVELAND, R. 1990. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica 27*, 8, 725–747.

CLEAVELAND, R., AND STEFFEN, B. 1990. When is "partial" adequate. A logic-based proof technique using partial specifications. In *Proceedings of the 5th Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif.

CLEAVELAND, R., PARROW, J., AND STEFFEN, B. 1989. The concurrency workbench. In *Proceedings of the* 1989 *International Workshop on Automatic Verification Methods for Finite State Systems* (Grenoble France). Lecture Notes in Computer Science, vol 407. Springer-Verlag, New York.

COUDERT, O., BERTHET, C., AND MADRE, J. C 1990. Verifying temporal properties of sequential machines without building their state diagrams. In *Proceedings of the 1990 Workshop on Computer-Aided Verification*. ACM, New York.

DILL, D. L. 1989. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, Cambridge, Mass.

EMERSON, E. A., AND HALPERN, J. Y. 1986. "Sometimes" and "Not Never" revisited: On branching time versus linear time. *J. ACM 33* 1, 151–178.

EMERSON, E. A., AND LEI, C.-L. 1986. Efficient model checking in fragments of the propositional mu-calculus. In *Proceedings of the 1st Annual Symposium on Logic in Computer Science* IEEE Computer Society Press, Los Alamitos, Calif.

GRAF, S., AND STEFFEN, B. 1990. Compositional minimization of finite state processes. In *Proceedings of the 1990 Workshop on Computer-Aided Verification*. ACM, New York.

HOPCROFT, J. E., AND ULLMAN, J. D. 1979 *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Mass.

JOSKO, B. 1989. Verifying the correctness of AADL-modules using model checking. In *Proceedings of the REX Workshop on Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness*. Lecture Notes in Computer Science, vol. 430. Springer-Verlag, New York.

KURSHAN, R. P. 1989. Analysis of discrete event coordination. In *Proceedings of the REX Workshop on Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness*. Lecture Notes in Computer Science, vol 430. Springer-Verlag, New York.

KURSHAN, R. P. AND MCMILLAN, K. L. 1989 A structural induction theorem for processes. In *Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing*. ACM Press, New York.

LARSEN, K. G. 1993. The expressive power of implicit specifications In *Proceedings of the 18th International Colloquium on Automata, Languages, and Programming*. Eur. Assoc. for Theoretical Computer Science. To be published.

LICHTENSTEIN, O., AND PNUELI, A. 1985. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the 12th Annual ACM Symposium on Principles of Programming Languages*. ACM, New York.

MILNER, R. 1980. *A Calculus of Communicating Systems*. Lecture Notes in Computer Science, vol. 92. Springer-Verlag, New York.

MILNER, R. 1971. An algebraic definition of simulation between programs. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence* (Sept.).

PNUELI, A. 1984. In transition for global to modular temporal reasoning about programs. In *Logics and Models of Concurrent Systems*. NATO ASI Series. Series F, Computer and System Sciences, vol. 13. Springer-Verlag, New York.

PNUELI, A., AND SHERMAN, R. 1981 Semantic tableau for temporal logic. Tech. Rep. CS81-21, The Weizmann Institute, Israel.

SHTADLER, Z., AND GRUMBERG, O. 1989. Network grammars, communication behaviors and automatic verification. In *Proceedings of the* 1989 *International Workshop on Automatic Verification Methods for Finite State Systems* (Grenoble, France). Lecture Notes in Computer Science, vol. 407. Springer-Verlag, New York

SHUREK, G., AND GRUMBERG, O. 1990. The modular framework of computer-aided verification: Motivation, solutions and evaluation criteria. In *Proceedings of the 1990 Workshop on Computer-Aided Verification*. ACM, New York.

STIRLING, C., AND WALKER, D. J. 1989. Local model checking in the modal mu-calculus. In *Proceedings of the 1989 International Joint Conference on Theory and Practice of Software Development*. Lecture Notes in Computer Science, vol. 351–352. Springer-Verlag, New York.

WALKER, D. 1988. Bisimulations and divergence. In *Proceedings of the 3rd Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif.

WINSKEL, G. 1989. Model checking in the modal ν-calculus. In *Proceedings of the 16th International Colloquium on Automata, Languages, and Programming*. Eur. Assoc. for Theoretical Computer Science.

WOLPER, P. 1983. Temporal logic can be more expressive. *Inf. Contr. 56*, 1/2 (Jan./Feb.), 72–99.

Received April 1992; revised February 1993; accepted April 1993