Formula-Dependent Equivalence for Compositional CTL Model Checking

Adnan Aziz Thomas R. Shiple Vigyan Singhal
Alberto L. Sangiovanni-Vincentelli
Email: {adnan,shiple,vigyan,alberto}@ic.eecs.berkeley.edu

Department of EECS, University of California, Berkeley, CA 94720

Abstract. We present a state equivalence that is defined with respect to a given CTL formula. Since it does not attempt to preserve all CTL formulas, like bisimulation does, we can expect to compute coarser equivalences. We use this equivalence to manage the size of the transition relations encountered when model checking a system of interacting FSMs. Specifically, the equivalence is used to reduce the size of each component FSM, so that their product will be smaller. We show how to apply the method, whether an explicit representation is used for the FSMs, or BDDs are used. Also, we show that in some cases our approach can detect if a formula passes or fails, without composing all the component machines. The method is exact and fully automatic, and handles full CTL.

1 Introduction

Formal design verification is the process of verifying that a design has certain properties that the designer intended. A well known verification technique is computation tree logic (CTL) model checking. In this approach, a design is modeled as a finite state machine (FSM), properties are stated using CTL formulas, and a "model checker" is used to prove that the FSM satisfies the given CTL formulas [6]. The complexity of model checking a formula is linear in the number of states of the FSM.

Oftentimes, large designs are constructed by linking together a set of FSMs. The straightforward approach to model checking such a design is to first form the product of the component FSMs to yield a single FSM, and then proceed to model check this single FSM. However, the size of the product machine can be exponential in the number of component machines, and hence the model checker may take exponential time. This is known as the "state explosion problem" when using explicit representations, or the "representation explosion problem" when using implicit representations, like ordered binary decision diagrams (BDDs). As it turns out, we cannot hope to do better than this in the worst case, because the problem of model checking a system of interacting FSMs is PSPACE-complete [1].

Our goal is to develop an algorithm that alleviates the explosion problem by identifying equivalent states in each component machine. These equivalent

states are then used to simplify the components before taking their product, thus leading to a smaller product machine. It is well known that *bisimulation equivalence* is the coarsest (or weakest) equivalence that preserves the truth of *all* CTL formulas [4]. However, in general we are interested in model checking a system with respect to just a few formulas, and hence preserving all CTL formulas is stronger than needed. Thus, we investigate a formula-dependent equivalence that preserves the truth of a particular formula of interest, but possibly not of other formulas. This leads to a coarser equivalence, and thus to a greater opportunity for simplification. If an explicit representation is used for the FSMs, then this equivalence is used to form the quotient machines of the components. If BDDs are used, then the equivalence relation is used to define a range of permissible transition relations, among which we want to use the one with the smallest BDD.

Consider for example the FSM M described in Figure 1. The CTL formula $\phi = \forall G(\text{REQ} \rightarrow \forall F \text{ACK})$ expresses the property that every request is eventually acknowledged. The behaviors from state 1 and 5 are different. However, since there are no behaviors from states 4 and 8 where REQ is produced, then ϕ is always true at these states. Hence, states 1 and 5 can actually be merged, with respect to ϕ. Consequently, M can be replaced by the 5-state machine M': verifying ϕ on a product machine containing the component M is equivalent to verifying ϕ on the product machine with M replaced by M'.

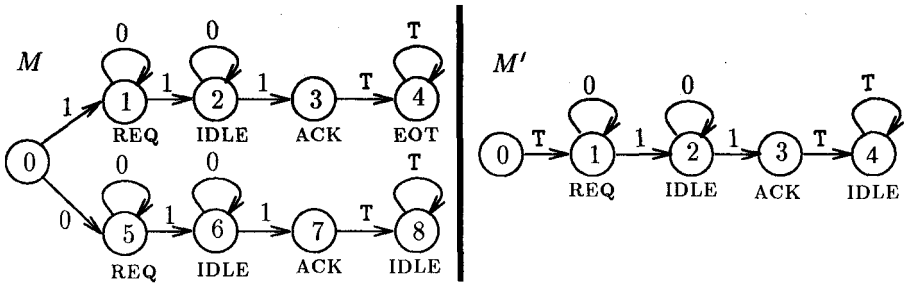

Fig. 1. Finite state machine M with inputs 0 and 1 and outputs REQ, ACK, IDLE and EOT. The symbol T means "true", the union of all input assignments.

The approach we have developed can be applied to *any* formula of CTL. Thus, we can handle formulas that refer to atomic propositions of any number of the component machines, and the formulas can be nested arbitrarily. The approach is fully automatic and it is exact, that is, it returns exactly the set of product states satisfying the formula of interest. Finally, in some cases the approach can detect if a formula passes or fails, without composing all the component machines.

Section 2 discusses related work, and Section 3 presents some preliminaries. In Section 4 we develop our formula-dependent equivalence, and in Section 5 we discuss how this equivalence can be used to simplify compositional model checking. Finally, Section 6 mentions future work and gives conclusions. Proofs for the propositions and theorems can be found in [2].

2 Related Work

Other researchers have addressed the problem of reducing the complexity of model checking. As mentioned in the introduction, bisimulation preserves the truth of all CTL formulas, and hence can be used to identify equivalent states to derive smaller component machines. This technique has been used by [3].

Clarke *et al.* presented the *interface rule*, which can be applied when a CTL formula refers to the atomic propositions of just one machine, the "main" machine [7]. In this case, the outputs of the other machines that cannot be sensed by the main machine, can be "hidden". After hiding such outputs, some states in the other machines may become equivalent, and hence the number of states can be reduced. This technique is orthogonal to our approach, and thus the two approaches could be combined. In general, any output not referred to by the formula, and not observable by other machines, can be hidden.

Grümberg *et al.* defined a subset of CTL, known as ACTL, which permits only universal path quantification, and not existential path quantification [11]. They go on to develop an approach to compositional model checking for ACTL. If an ACTL formula is true of one component in a system, then it is true of the entire system. Thus, in some cases the full product machine can be avoided. However, the formula may be true of the entire system, *without* being true of any one component in isolation, i.e. their approach is conservative, and not exact. In this case, some components must be composed, and the procedure repeated. The user has the option of manually forming abstractions for some of the machines. If the formula is false, then the product machine must always be formed. An asset of this approach is that it handles fairness constraints on the system.

Dams *et al.* have also devised an approach using ACTL [9]. Like our method, they compute an equivalence with respect to a *single* formula. Although they are limited to formulas of ACTL, it may turn out that coarser equivalences are possible by restricting to a subset of CTL. They do not address how their equivalence can be used in *compositional* model checking, where a formula may refer to the atomic propositions of several interacting machines.

Our experience indicates that existential path properties are useful for determining if a system *can* exhibit a certain behavior. This is especially true when ascertaining if the environment for a system has been correctly modeled so that it can produce the stimuli of interest. Hence, we are interested in techniques that can handle *full* CTL.

The work of Chiodo *et al.* [5] has similar aims as ours, and the current work can be seen as an outgrowth of that work. Both approaches are exact, fully automatic, and formula dependent. We have extended Chiodo's method (see Section 5.2), and have cast our extension as an equivalence on states.

3 Preliminaries

3.1 Finite States Machines

The systems that we want to verify are synchronous, interacting FSMs. Each component FSM receives a set of binary-valued inputs, and produces another

set of binary-valued outputs. Formally, an FSM is a 5-tuple $M = (S, I, J, T, O)$, where S is a finite set of states, $I = \{\alpha_1, \ldots, \alpha_m\}$ is a set of m inputs supplied by the environment of the FSM, $J = \{\beta_1, \ldots, \beta_n\}$ is a set of n outputs, T is the transition relation, and O is the output function. T relates a starting state, an assignment to the inputs, and an ending state, i.e. $T \subseteq S \times \Sigma \times S$, where $\Sigma = 2^I$. We require the transition relation to be *complete*, so that for each $a \in \Sigma$ and $x \in S$, there exists at least one $y \in S$ such that $(x, a, y) \in T$. The output function takes a state in S and returns an assignment to the outputs, i.e. $O : S \rightarrow 2^J$. Our definition of FSM is equivalent to that of a Moore machine in [11].

Composition is defined in the usual way. In composing two interacting FSMs, some inputs of each machine may be equal to the outputs of the other machine, whereas other inputs may come from the environment of the composed FSM. Thus, the inputs of the composition are the inputs of the components that are not outputs of either component. The outputs of the composition are all the outputs of the components. Figure 2 shows an example, where M_1 has states $\{1, 2, 3\}$, and M_2 has states $\{1', 2'\}$. The sets of inputs and outputs for M_1 are $\{a, q\}$ and $\{p\}$ respectively; and for M_2 are $\{a, p\}$ and $\{q\}$ respectively. For the composition $M_1 \times M_2$, the sets of inputs and outputs are $\{a\}$ and $\{p, q\}$ respectively.

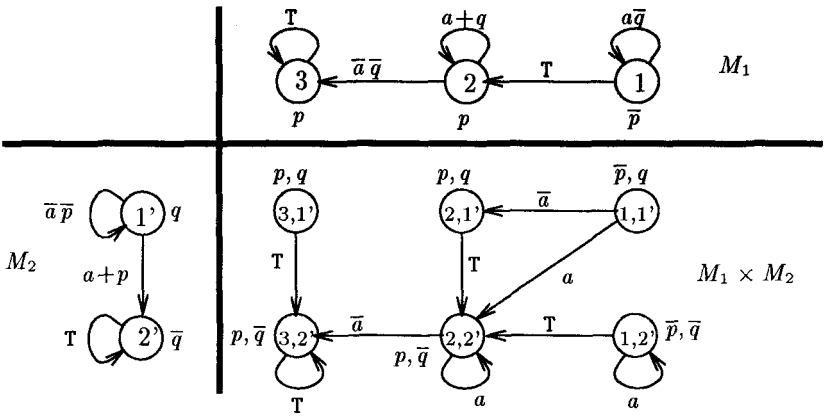

Fig. 2. Example of FSM composition: p is the output of M_1, q is the output of M_2, and a is an external input. $a\bar{q}$ is shorthand for the subset $\{\{a\}\} \subseteq 2^{\{a,q\}}$. The union of $a\bar{q}$, $\bar{a}q$ and aq is denoted by $a+q$.

3.2 Computation Tree Logic

Computation tree logic is a language used to describe properties of state transition systems. We are interested in checking CTL formulas that describe properties of the composition of a set of interacting FSMs. Since the composition of a set of FSMs is again an FSM, we give the syntax and semantics of CTL for a single FSM M. We allow two types of atomic propositions:

1. each output variable is an atomic proposition, and
2. each subset of states is an atomic proposition

The second type arises naturally when recursively checking formulas. With this, the set of CTL formulas is defined inductively as follows:

- p is a CTL formula, where p is an output variable or a subset of states, and
- if ψ_1 and ψ_2 are CTL formulas, then so are $\neg\psi_1$, $\psi_1 \vee \psi_2$, $\exists X\psi_1$, $\exists G\psi_1$, and $\exists[\psi_1\ U\ \psi_2]$.

Note that inputs are *not* allowed as atomic propositions. However, by modeling an input by an FSM whose output describes the expected behavior of the input, one can implicitly use an input as an atomic proposition.

The semantics of CTL is usually defined on finite Kripke structures, which are directed graphs where each node is labeled by a set of atomic propositions [6]. To extend these semantics to FSMs, we just ignore the labels on the transitions of the FSMs, and we view the outputs as atomic propositions. Let $M = (S, I, J, T, O)$ be an FSM. A *path* from state x_0 is an infinite sequence of states $x_0x_1x_2\ldots$ such that for every i, there exists an $a \in \Sigma$ such that $(x_i, a, x_{i+1}) \in T$. The notation $M, x_0 \models \phi$ means that ϕ is true in state x_0 of FSM M. The semantics of CTL is defined inductively as follows:

- $M, x_0 \models p$, where $p \in J$, iff $p \in O(x_0)$.
- $M, x_0 \models p$, where $p \subseteq S$, iff $x_0 \in p$.
- $M, x_0 \models \neg\psi_1$ iff $M, x_0 \not\models \neg\psi_1$.
- $M, x_0 \models \psi_1 \vee \psi_2$ iff $M, x_0 \models \psi_1$ or $M, x_0 \models \psi_2$.
- $M, x_0 \models \exists X\psi_1$ iff there exists a path $x_0x_1x_2\ldots$ such that $M, x_1 \models \psi_1$.
- $M, x_0 \models \exists G\psi_1$ iff there exists a path $x_0x_1x_2\ldots$ such that for all i, $M, x_i \models \psi_1$.
- $M, x_0 \models \exists[\psi_1\ U\ \psi_2]$ iff there exists a path $x_0x_1x_2\ldots$ and some $i \geq 0$ such that $M, x_i \models \psi_2$ and for all $j < i$, $M, x_j \models \psi_1$.

For example in machine $M_1 \times M_2$ of Figure 2, state $\langle 1, 2'\rangle$ satisfies the formula $\exists G(\neg p \wedge \neg q)$, whereas none of the other state do. The expression $\exists F\psi$ is an abbreviation for $\exists[\mathit{true}\ U\ \psi]$, where *true* is a logical tautology. Lastly, we define the *CTL model checking problem* as the problem of determining *all* states of the system that satisfy a given formula.[1]

4 Formula-Dependent Equivalence

Our goal is to define an equivalence on the states of each component machine that is as coarse as possible with respect to a given CTL formula ϕ, while being efficiently computable. Section 5 explains how we intend to apply this equivalence to model checking, but the main idea is to merge equivalent states to minimize the size of each component. The minimized machines are then composed. Optionally, the product can be computed incrementally by composing a

[1] If a set of initial states is known, then we can restrict our attention to the reachable state space. In this case, we can apply known techniques for exploiting the unreachable states, such as minimizing the transition relation with respect to unreachable states; these techniques are orthogonal to those discussed in this paper.

few of the minimized machines, and then computing a new equivalence for this sub-product. When the top level is reached and just a single machine remains, the usual CTL model checking algorithm is applied to determine the states that satisfy ϕ.

Our formula dependent equivalence can be best explained by comparing it to bisimulation. ("strong bisimulation" of Milner [12, p. 88]) Given an FSM $M = (S, I, J, T, O)$, the bisimulation equivalence relation, denoted by $\sim$, is the coarsest equivalence relation satisfying the following:

For all $x, y \in S$, $x \sim y$ implies

- $O(x) = O(y)$ and
- for all $a \in \Sigma$ (recall from Section 3 that $\Sigma = 2^I$)
  - whenever $x \xrightarrow{a} t$, then for some w, $y \xrightarrow{a} w$ and $t \sim w$, and
  - whenever $y \xrightarrow{a} w$, then for some t, $x \xrightarrow{a} t$ and $t \sim w$.

The soundness of this definition follows from the observation that the class of equivalence relations satisfying the above definition contains the identity, and is closed under union. Intuitively, two states are bisimilar if their corresponding infinite computation trees[2] "match". This means that the two states have the same outputs, and on each input, the two states have next states whose infinite computation trees again match.

We use the notion of *PASS* and *FAIL* states to ease the strict requirement of bisimulation that the infinite computation trees of two states match. Loosely, if a state is a $PASS^\phi$ state with respect to a CTL formula ϕ, then it satisfies ϕ in all environments; likewise, if a state is $FAIL^\phi$, then it does not satisfy ϕ in any environment. Given $PASS^\phi$ and $FAIL^\phi$ states, the first modification to bisimulation we make is that subtrees rooted at $FAIL^\phi$ states are ignored. This means that transitions to $FAIL^\phi$ states from one state need not be matched by the other state. This works because only potential witnesses to a formula need to be preserved. The second modification is that two states are equivalent if they are both $PASS^\phi$ states. A consequence of this is that whereas bisimulation requires the infinite computation trees of next states to match, now it is sufficient that the next states are both $PASS^\phi$ states. This is what we mean by two infinite trees matching *up to* $PASS^\phi$ states. Essentially then, we say that two states are equivalent with respect to ϕ if

1. they are equivalent with respect to the immediate subformulas of ϕ, and
2. either they are both $PASS^\phi$ states or both $FAIL^\phi$ states, or the infinite computation trees of the two states match up to $PASS^\phi$ states, ignoring all subtrees rooted at $FAIL^\phi$ states.

Before formally defining our equivalence relation, we define the $PASS^\phi$ and $FAIL^\phi$ sets. For a given formula ϕ, $PASS^\phi$ and $FAIL^\phi$ sets are defined for each component. In the following definition, we assume a system of just two components, M and M'. In defining the $PASS^\phi$ and $FAIL^\phi$ sets for M, M' is referenced because the atomic propositions in ϕ may refer to M'. The symbols p_o and p_i

[2] The infinite computation tree of a state is formed by "unrolling" the FSM starting from that state.

are used to distinguish those output atomic propositions produced by M and those produced by M'.

Definition 1. Let $M = (S, I, J, T, O)$ and $M' = (S', I', J', T', O')$ be FSMs, and let ϕ be a CTL formula. Let $p_o \in J$, $p_i \in J'$, and $p_s \subseteq S \times S'$. $PASS^\phi$ and $FAIL^\phi$ for M are subsets of S, as follows:

ϕ		
p_i	$PASS^\phi$	$\emptyset$
	$FAIL^\phi$	$\emptyset$
p_o	$PASS^\phi$	$\{x \in S \mid p_o \in O(x)\}$
	$FAIL^\phi$	$S \backslash PASS^\phi$
p_s	$PASS^\phi$	$\{x \in S \mid \forall s' \in S', (x, s') \in p_s\}$
	$FAIL^\phi$	$\{x \in S \mid \forall s' \in S', (x, s') \notin p_s\}$
$\neg\psi$	$PASS^\phi$	$FAIL^\psi$
	$FAIL^\phi$	$PASS^\psi$
$\psi_1 \vee \psi_2$	$PASS^\phi$	$PASS^{\psi_1} \cup PASS^{\psi_2}$
	$FAIL^\phi$	$FAIL^{\psi_1} \cap FAIL^{\psi_2}$
$\exists X\psi$	$PASS^\phi$	$\{x \in S \mid \forall a \in \Sigma, \exists t$ s.t. $x \xrightarrow{a} t$ and $t \in PASS^\psi\}$
	$FAIL^\phi$	$\{x_0 \in S \mid$ for every path $x_0 x_1 x_2 \ldots, x_1 \in FAIL^\psi\}$
$\exists G\psi$	$PASS^\phi$	greatest fixed-point of: $R_0 = PASS^\psi$; $R_{i+1} = R_i \cap \{x \in S \mid \forall a \in \Sigma, \exists t$ s.t. $x \xrightarrow{a} t$ and $t \in R_i\}$
	$FAIL^\phi$	$\{x_0 \in S \mid$ for every path $x_0 x_1 x_2 \ldots,$ there exists $i \geq 0$ s.t. $x_i \in FAIL^\psi\}$
$\exists[\psi_1 U \psi_2]$	$PASS^\phi$	least fixed-point of: $R_0 = PASS^{\psi_2}$; $R_{i+1} = R_i \cup \{x \in S \mid x \in PASS^{\psi_1}$, and $\forall a \in \Sigma, \exists t$ s.t. $x \xrightarrow{a} t$ and $t \in R_i\}$
	$FAIL^\phi$	$\{x_0 \in S \mid$ for every path $x_0 x_1 x_2 \ldots$, either 1) there exists $i \geq 0$ s.t. $x_i \in FAIL^{\psi_1}$ and $\forall j \leq i, x_j \in FAIL^{\psi_2}$, or 2) $\forall i \geq 0, x_i \in FAIL^{\psi_2}\}$

As an example of $PASS^\phi$ and $FAIL^\phi$, consider the FSM in Figure 3. For $\psi = p$, states 1, 2, 3, 5, 6 and 7 lie in $PASS^\psi$ and states 4 and 8 lie in $FAIL^\psi$. For $\phi = \exists Gp$, states 3 and 7 lie in $PASS^\phi$, while states 4 and 8 lie in $FAIL^\phi$, and states 1, 2, 5 and 6 lie in neither. The following proposition says that, indeed, if x is in $PASS^\phi$, then any product state with x as a component satisfies ϕ.

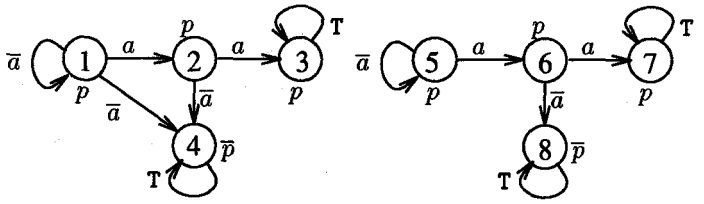

Fig. 3. Illustrating $PASS^\phi$ and $FAIL^\phi$, and the fact that $\mathcal{E}^\phi$ is coarser than bisimulation.

Proposition 2. Let ϕ be a CTL formula, let x be a state of M, and let t be a state of any FSM M'. If $x \in PASS^{\phi}$, then $M \times M', \langle x,t \rangle \models \phi$. Likewise, if $x \in FAIL^{\phi}$, then $M \times M', \langle x,t \rangle \not\models \phi$.

Note that the converse is not true. For example, consider a component M and the formula $\phi = q \wedge \neg q$, where q is an output of some other component. Then $FAIL^{\phi}$ for M is empty (because $FAIL^{q}$ and $PASS^{q}$ are empty by case p_i), even though ϕ is not satisfiable (i.e. for any component M', no state in $M \times M'$ satisfies ϕ). In fact, by generalizing this reasoning, we can show that if $FAIL^{\phi}$ were defined in such a way that the converse of Proposition 2 did hold, then $FAIL^{\phi}$ would be EXPTIME-hard to compute. The reduction is from CTL satisfiability, which is known to be EXPTIME-complete [10]. To check if a formula ϕ is satisfiable, compute $FAIL^{\phi}$ for the component M shown in Figure 4, where p is some atomic proposition *not* in ϕ. We can show that $x \in FAIL^{\phi}$ if and only if ϕ is not satisfiable, and thus satisfiability can be answered if we could compute $FAIL^{\phi}$ exactly. Similarly, since $x \in FAIL^{\phi}$ if and only if $x \in PASS^{\neg\phi}$, the same reduction shows that $PASS^{\phi}$ would also be EXPTIME-hard to compute.

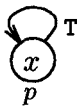

Fig. 4. Component machine used to show that computing $FAIL^{\phi}$ exactly is EXPTIME-hard.

Now we formally define our equivalence relation. Let $M = (S, I, J, T, O)$ and $M' = (S', I', J', T', O')$ be FSMs, and let ϕ be a CTL formula. Following Milner's development of bisimulation, we define the equivalence relation $\mathcal{E}^{\phi}$ on the states of FSM M as the coarsest equivalence relation satisfying the following:

For $x, y \in S$, $\mathcal{E}^{\phi}(x,y)$ iff:
Case $\phi = p_i$: $(x,y) \in S \times S$.
Case $\phi = p_o$: $x \in PASS^{\phi}$ and $y \in PASS^{\phi}$, or $x \in FAIL^{\phi}$ and $y \in FAIL^{\phi}$.
Case $\phi = p_s$: for all $s' \in S'$, $(x,s') \in p_s$ iff $(y,s') \in p_s$.
Case $\phi = \neg\psi$: $\mathcal{E}^{\psi}(x,y)$.
Case $\phi = \psi_1 \vee \psi_2$: $\mathcal{E}^{\psi_1}(x,y)$ and $\mathcal{E}^{\psi_2}(x,y)$.
Case $\phi = \exists X\psi$: $\mathcal{E}^{\psi}(x,y)$ and
1. $x \in FAIL^{\phi}$ and $y \in FAIL^{\phi}$, or $x \in PASS^{\phi}$ and $y \in PASS^{\phi}$, or
2. $O(x) = O(y)$, and for all $a \in \Sigma$
   - whenever $x \xrightarrow{a} t$ and $t \notin FAIL^{\psi}$, $\exists w$ s.t. $y \xrightarrow{a} w$ and $\mathcal{E}^{\psi}(t,w)$, and
   - whenever $y \xrightarrow{a} w$ and $w \notin FAIL^{\psi}$, $\exists t$ s.t. $x \xrightarrow{a} t$ and $\mathcal{E}^{\psi}(t,w)$.

Case $\phi = \exists G\psi$: $\mathcal{E}^{\psi}(x,y)$ and
1. $x \in FAIL^{\phi}$ and $y \in FAIL^{\phi}$, or $x \in PASS^{\phi}$ and $y \in PASS^{\phi}$, or
2. $O(x) = O(y)$, and for all $a \in \Sigma$
   - whenever $x \xrightarrow{a} t$ and $t \notin FAIL^{\phi}$, $\exists w$ s.t. $y \xrightarrow{a} w$ and $\mathcal{E}^{\phi}(t,w)$, and
   - whenever $y \xrightarrow{a} w$ and $w \notin FAIL^{\phi}$, $\exists t$ s.t. $x \xrightarrow{a} t$ and $\mathcal{E}^{\phi}(t,w)$.

Case $\phi = \exists[\psi_1 \, U \, \psi_2]$: $\mathcal{E}^{\psi_1}(x,y)$ and $\mathcal{E}^{\psi_2}(x,y)$ and

1. $x \in FAIL^\phi$ and $y \in FAIL^\phi$, or $x \in PASS^\phi$ and $y \in PASS^\phi$, or
2. $O(x) = O(y)$, and for all $a \in \Sigma$
   - whenever $x \xrightarrow{a} t$ and $t \notin FAIL^\phi$, $\exists w$ s.t. $y \xrightarrow{a} w$ and $\mathcal{E}^\phi(t, w)$, and
   - whenever $y \xrightarrow{a} w$ and $w \notin FAIL^\phi$, $\exists t$ s.t. $x \xrightarrow{a} t$ and $\mathcal{E}^\phi(t, w)$.

In a manner similar to Milner, we can show that $\mathcal{E}^\phi$ is the maximum fixed-point of a certain functional. Hence, using a standard fixed-point computation, $\mathcal{E}^\phi$ can be computed in polynomial time.

Notice that $\mathcal{E}^\phi$ requires equivalence on all subformulas. As the following example shows, this requirement is warranted. Consider M_1 in Figure 5. For $\phi = \exists F(p \wedge \exists F(\overline{p} \wedge q))$, states 2, 3 and 5 lie in $FAIL^\phi$ because p is false in these states. So with respect to ϕ, the infinite computation trees of 1 and 4 match when $FAIL^\phi$ states are ignored, and if we did not require equivalence on subformulas, they would be $\mathcal{E}^\phi$-equivalent. However, if we were to compose M_1 with M_2, ϕ holds in state $\langle 1, 1' \rangle$ but does not hold in state $\langle 4, 1' \rangle$. Thus, it would be wrong to have 1 and 4 be $\mathcal{E}^\phi$-equivalent. Requiring equivalence on all subformulas fixes this problem.

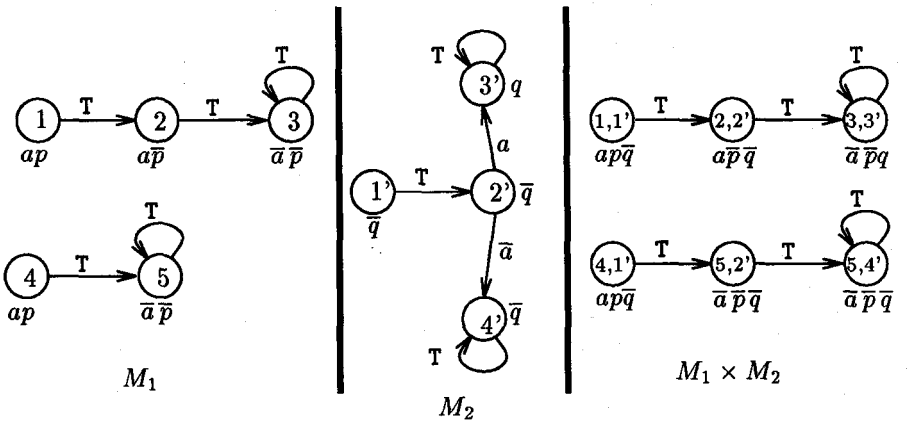

Fig. 5. Equivalence on subformulas is required. Only the states reachable from $\langle 1, 1' \rangle$ and $\langle 4, 1' \rangle$ are shown in $M_1 \times M_2$.

Since we define CTL so that formulas may refer directly to states via atomic propositions, then any formula-*independent* equivalence (e.g. bisimulation) will distinguish every pair of states, whereas $\mathcal{E}^\phi$ may make some states equivalent. However, even if we could not refer to states, $\mathcal{E}^\phi$ is still coarser than bisimulation. As stated earlier, one reason for this is that the subtrees rooted at $FAIL^\phi$ states are ignored. This is illustrated in Figure 3: if $\phi = \exists Gp$, then 4 is a $FAIL^\phi$ state, and thus 1 and 5 are $\mathcal{E}^\phi$-equivalent; however, they are not bisimilar.

On the other hand, there are cases where $\mathcal{E}^\phi$ distinguishes two states that can actually be merged. Consider the FSM M_1 in Figure 6 and the formula $\phi = \exists Gq$, where q is an output of some component not shown. Since q is an input to M_1, the sets $PASS^\phi$ and $FAIL^\phi$ are empty, and hence $\mathcal{E}^\phi$ reduces to bisimulation.

States 1 and 3 are not bisimilar because 2 and 4 have different outputs, and thus 1 and 3 are not $\mathcal{E}^\phi$-equivalent. However, q must be false to reach states 2 and 4, and thus the difference between states 1 and 3 does not affect the validity of ϕ. Hence, states 1 and 3 can be merged with respect to ϕ, but $\mathcal{E}^\phi$ will not merge them.

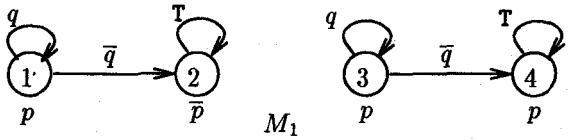

Fig. 6. $\mathcal{E}^\phi$ equivalence is incomplete. The input to M_1 is q, and the output is p. States 1 and 3 can be safely merged with respect to the formula $\phi = \exists G q$.

The following proposition says that $\mathcal{E}^\phi$-equivalent states cannot be distinguished, with respect to ϕ, by any environment. This is key in proving Theorem 4, the theorem of correctness.

Proposition 3. Let ϕ be a CTL formula, and let x and y be states of M such that $\mathcal{E}^\phi(x, y)$. Then for any state t of any FSM M', the following holds: $M \times M', \langle x, t\rangle \models \phi$ iff $M \times M', \langle y, t\rangle \models \phi$.

As an aside, note that the converse of Proposition 3 is not true. In fact, just because two states cannot be distinguished with respect to ϕ by any environment, this does not imply that they can be merged. For example, consider M_1 in Figure 5, and the formula $\phi = \exists F(p \wedge \exists F(\overline{p} \wedge q))$. As stated earlier, states 2 and 5 lie in $FAIL^\phi$, and thus for any state t of any FSM M', $M_1 \times M', \langle 2, t\rangle \models \phi$ iff $M_1 \times M', \langle 5, t\rangle \models \phi$ (i.e. by Proposition 2, neither $\langle 2, t\rangle$ nor $\langle 5, t\rangle$ satisfies ϕ). However, if we were to merge states 2 and 5 into a single state, states 1 and 4 would become equivalent. But, as discussed earlier, it would be wrong to have 1 and 4 be $\mathcal{E}^\phi$-equivalent.

Ultimately, the purpose of computing $\mathcal{E}^\phi$ is to be able to merge equivalent states, thus leading to smaller component machines. Given an equivalence relation on the states of an FSM, we define the *quotient machine* in the usual way. As we describe in the next section, if we are using an explicit representation for FSMs, then we use the quotient machine of each FSM in place of the original component. The following theorem asserts that doing this does not alter the result returned by the model checker. If we are using an implicit representation, then we use $\mathcal{E}^\phi$ to define a range of permissible transition relations for each component, among which we want to use the smallest.

Theorem 4. Let ϕ be a CTL formula, and let $M_1, \ldots, M_n$ be FSMs. Let $M_i/\mathcal{E}_i^\phi$ be the quotient of M_i with respect to $\mathcal{E}_i^\phi$, and let $[s_i]$ denote the equivalence class of $\mathcal{E}_i^\phi$ containing s_i. Then for all product states $\langle s_1, \ldots, s_n\rangle$,

$$M_1 \times \ldots \times M_n, \langle s_1, \ldots, s_n\rangle \models \phi \text{ iff } M_1/\mathcal{E}_1^\phi \times \ldots \times M_n/\mathcal{E}_n^\phi, \langle [s_1], \ldots, [s_n]\rangle \models \phi.$$

5 Compositional Model Checking

The equivalence relation $\mathcal{E}^\phi$ can be used to manage the size of the transition relations encountered in compositional model checking. The assumptions are that each component machine is relatively small and easy to manipulate, and that the full product machine is too large to build and manipulate. The general idea is to minimize each component machine, with respect to $\mathcal{E}^\phi$ for that machine, before composing it with other machines. We can incrementally build the product machine by composing machines into clusters, and again applying minimization to each cluster. When just one machine remains, we apply a standard CTL model checker. Figure 7 outlines a procedure for this approach.

```
function compositional_model_checker(φ, M_1, ..., M_n) {
  if (n = 1)
    return model_checker(φ, M_1);
  for (i = 1; i ≤ n; i++)
    M_i* = minimize(M_i, φ);
  M_1', ..., M_l' = form_clusters(M_1*, ..., M_n*);
  compositional_model_checker(φ, M_1', ..., M_l');
}
```

Fig. 7. Outline of procedure for compositional model checking: minimize and form product incrementally.

The question of how to minimize a component with respect to $\mathcal{E}^\phi$ depends on what sort of data representation is used for the transition relations. If an explicit representation is used (e.g. adjacency lists), then minimization is simply a matter of forming the quotient machines $M_i/\mathcal{E}_i^\phi$. After the model checker is applied to the product of the quotient machines, Theorem 4 can be directly applied to recover the product states in the original state space that satisfy ϕ.

If an implicit representation is used, then minimization becomes more complicated. We focus on the case where BDDs are used. There is no correlation between the size of the BDD for a transition relation, and the number of transitions in the relation. Thus, the idea behind minimization in this case is to use $\mathcal{E}^\phi$ to define a range of transition relations, any of which can be used in place of the original transition relation, and then choose the relation in this range with the smallest BDD. It should be noted however, that smaller component BDDs do not *guarantee* a smaller product BDD—this is only a heuristic.

For a component M, we take the upper bound of the range to be T^{max}, which is the relation formed by adding to T any transition between two states for which there exists a transition between equivalent states (e.g. if $s \xrightarrow{a} s'$ is in T and $\mathcal{E}^\phi(x,s)$ and $\mathcal{E}^\phi(x',s')$, then $x \xrightarrow{a} x'$ is added). The lower bound is T itself. Given these bounds, a heuristic like *restrict* [8] is used to find a small BDD between T and T^{max}. It can be shown that any transition relation between T and T^{max} can be used without altering the result returned by the model checker. Alternatively, instead of looking for a small relation between T and T^{max}, we can just use T^{min}, which is the transition relation of the quotient machine, if it turns out that T^{min} is small.

5.1 Early Pass/Fail Detection

Sometimes the model checking problem is posed as: given a formula ϕ and a subset of product states Q, is Q contained in the set of states satisfying ϕ? For example, Q may be the set of initial states. Since our method returns all states satisfying ϕ, a simple containment check answers the question. However, in some cases, we may be able to answer the question without composing all the machines, yielding a further savings in time. This is known as early pass/fail detection.

Let $Q = \{q^1, q^2, \ldots, q^m\}$, where q^j is the product state $\langle s_1^j, s_2^j, \ldots, s_n^j \rangle$, and let $FAIL_i^\phi$ be the $FAIL^\phi$ states in component i. If $s_i^j \in FAIL_i^\phi$, then any product state $\langle t_1, \ldots, t_{i-1}, s_i^j, t_{i+1}, \ldots, t_n \rangle$ does not satisfy ϕ, so in particular, q^j does not satisfy ϕ. Hence, the answer to the above question is "no". So in summary, if for any i, $FAIL_i^\phi$ intersects the ith state component of the set Q, then the answer is "no".

On the other hand, to reach an early "yes" answer, we need each state in Q to be "covered" by at least one $PASS^\phi$ state. If $s_i^j \in PASS_i^\phi$, then every state in Q with s_i^j as its ith component is guaranteed to satisfy ϕ. So in summary, if for every state in Q, at least one of its component states is a $PASS^\phi$ state, then the answer is "yes".

5.2 Processing Subformulas

As the number of subformulas in ϕ increases, the equivalence $\mathcal{E}^\phi$ becomes finer because equivalence on all subformulas is required. However, if some of the subformulas of ϕ are first replaced by fresh atomic propositions representing the product states satisfying the subformulas, then this may lead to a coarser equivalence. This follows since knowing which product states satisfy a subformula adds information to what was originally known, information that can be used at the component level in computing $\mathcal{E}^\phi$ (for the new ϕ).

This is illustrated by the system in Figure 2, where $\phi = (\exists G(p \wedge q)) \wedge Q$, and Q is the set $\{\langle 1, 1' \rangle, \langle 2, 1' \rangle\}$ of product states. Lines 1 through 6 of Table 1 show the equivalence classes calculated for M_1 on the subformulas of ϕ. The end result (line 6) is that no states are equivalent; hence, we have gained nothing. Instead of processing all of ϕ, we could stop after computing the equivalence for $\exists G(p \wedge q)$. In this case, states 2 and 3 are equivalent (line 4), and thus a smaller machine can be built for M_1. When this quotient machine is composed with M_2 and the model checker is applied, we discover that no product states satisfy $\exists G(p \wedge q)$. At this point, we can create a fresh atomic proposition, Q', to represent this (empty) set of states. Then when we calculate the equivalence on M_1 for $Q' \wedge Q$ (which is the same as the original ϕ), we see that states 1 and 2 are now equivalent (line 8), so we can again construct a smaller machine for M_1.

Thus, we may want to follow a strategy where a nested formula is recursively decomposed into simpler subformulas, and the compositional model checker of Figure 7 is applied to each subformula. Note that whereas Chiodo *et al.* [5] recursively decompose a formula into its *immediate* subformulas, we can decompose

	ϕ	$PASS^{\phi}$	$FAIL^{\phi}$	equiv classes
1	q	$\emptyset$	$\emptyset$	$\{1,2,3\}$
2	p	$\{2,3\}$	$\{1\}$	$\{1\},\{2,3\}$
3	$p \wedge q$	$\emptyset$	$\{1\}$	$\{1\},\{2,3\}$
4	$\exists G(p \wedge q)$	$\emptyset$	$\{1\}$	$\{1\},\{2,3\}$
5	Q	$\emptyset$	$\{3\}$	$\{1,2\},\{3\}$
6	$(\exists G(p \wedge q)) \wedge Q$	$\emptyset$	$\{1,3\}$	$\{1\},\{2\},\{3\}$
7	Q'	$\emptyset$	$\{1,2,3\}$	$\{1,2,3\}$
8	$Q' \wedge Q$	$\emptyset$	$\{1,2,3\}$	$\{1,2\},\{3\}$

Table 1. Equivalence classes for M_1 of Figure 2 on $(\exists G(p \wedge q)) \wedge Q$.

a formula into *arbitrary* subformulas, since our equivalence works on nested formulas.

Of course, even though we may be able to compute coarser equivalences with this strategy, the drawback is that a reduced product machine must be reconstructed for each subformula. Experiments are required to determine how to decompose a formula to achieve a balance between these conflicting demands.

6 Future Work and Conclusions

We have presented a formula-dependent equivalence that can be used to manage the size of the transition relations encountered in compositional CTL model checking. We have yet to implement the method, and the ultimate effectiveness of the method can be confirmed only by experimentation. Given an arbitrary CTL formula ϕ, the method works by first computing an equivalence on the states of each component machine, which preserves ϕ. If an explicit representation for transition relations is used, then the quotient machine is constructed for each component, and the quotient machines are used to build a smaller product machine.

If BDDs are used, then the equivalence for each component is used to determine a range of permissible transition relations. More work remains to derive a procedure for efficiently choosing a relation from this range that will ultimately lead to a smaller product machine.

Our approach can be applied incrementally to build the product machine by clustering some minimized machines, forming their product, and repeating the equivalence computation. Research is needed to understand how best to cluster the components to achieve the smallest sub-products. Also, we outlined how our approach can be applied to the subformulas of a formula, to achieve a coarser equivalence. We need to devise a heuristic to intelligently decompose a formula into subformulas to take advantage of this.

An important part of a CTL model checker is the ability to generate counter-examples. Since we are altering the product machine, a counter-example in the altered product may not actually exist in the full product. A method needs to be developed to handle this. Finally, we plan to extend our method to fair-CTL model checking, and we would like to apply similar ideas to the language containment paradigm.

Acknowledgments

We wish to thank the reviewers for their helpful comments. This work was supported by SRC grant 94-DC-008, SRC contract 94-DC-324, and NSF/DARPA grant MIP-8719546. In addition, the second author was supported by an SRC Fellowship.

References

1. A. Aziz and R. K. Brayton. Verifying interacting finite state machines. Technical Report UCB/ERL M93/52, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, July 1993.
2. A. Aziz, T. R. Shiple, V. Singhal, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Formula-dependent equivalence for compositional CTL model checking. Technical report, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, 1994.
3. A. Bouajjani, J.-C. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel. Minimal state graph generation. *Science of Computer Programming*, 18(3):247–271, 1992.
4. M. C. Browne, E. M. Clarke, and O. Grumberg. Characterizing Kripke structures in temporal logic. Technical Report CS 87-104, Department of Computer Science, Carnegie Mellon University, 1987.
5. M. Chiodo, T. R. Shiple, A. L. Sangiovanni-Vincentelli, and R. K. Brayton. Automatic compositional minimization in CTL model checking. In *Proc. Int'l Conf. on Computer-Aided Design*, pages 172–178, Nov. 1992.
6. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, Apr. 1986.
7. E. M. Clarke, D. E. Long, and K. L. McMillan. Compositional model checking. In *4th Annual Symposium on Logic in Computer Science*, Asilomar, CA, June 1989.
8. O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In J. Sifakis, editor, *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 365–373. Springer-Verlag, June 1989.
9. D. Dams, O. Grumberg, and R. Gerth. Generation of reduced models for checking fragments of CTL. In C. Courcoubetis, editor, *Proceedings of the Conference on Computer-Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 479–490. Springer-Verlag, June 1993.
10. E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 995–1072. Elsevier Science Publishers B.V., 1990.
11. O. Grumberg and D. E. Long. Model checking and modular verification. In J. C. M. Baeten and J. F. Groote, editors, *CONCUR '91, International Conference on Concurrency Theory*, volume 527 of *Lecture Notes in Computer Science*. Springer-Verlag, Aug. 1991.
12. R. Milner. *Communication and Concurrency*. Prentice Hall, New York, 1989.