AUTOMATIC REDUCTION IN CTL COMPOSITIONAL MODEL CHECKING

by

Massimiliano Chiodo, Thomas R. Shiple,
Alberto Sangiovanni-Vincentelli, and Robert K. Brayton

Memorandum No. UCB/ERL M92/55

3 January 1992

COVER PAGE

AUTOMATIC REDUCTION IN CTL COMPOSITIONAL MODEL CHECKING

by

Massimiliano Chiodo, Thomas R. Shiple,
Alberto Sangiovanni-Vincentelli, and Robert K. Brayton

Memorandum No. UCB/ERL M92/55

3 January 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

AUTOMATIC REDUCTION IN CTL COMPOSITIONAL MODEL CHECKING

by

Massimiliano Chiodo, Thomas R. Shiple,
Alberto Sangiovanni-Vincentelli, and Robert K. Brayton

Memorandum No. UCB/ERL M92/55

3 January 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Automatic Reduction in CTL Compositional Model Checking

Massimiliano Chiodo
Magneti Marelli
Pavia, Italy

Thomas R. Shiple, Alberto Sangiovanni-Vincentelli, Robert K. Brayton
Department of Electrical Engineering and Computer Science
University of Californa, Berkeley, California

Abstract

In this project, we develop a method to reduce the complexity of verifying an arbitrary CTL formula on a set of communicating FSMs. We have proven that our method yields the correct result. It remains to determine the practical usefulness of our approach.

1 Introduction

Temporal model checking procedures are potentially a powerful verification tool for finite state systems. However, when the system under examination consists of several communicating parallel machines, the problem of explosion in the size of the representation arises. That is, the representation of the parallel composition of a number of relatively small FSMs may grow extremely large.

To avoid the representation explosion, one may try to verify the machines individually and then piece the results together to conclude that the entire system is correct. Unfortunately, it is easy to come up with examples where a critical property of some machine is not preserved when this machine is composed with other machines.

The compositional model checking algorithm proposed by Clarke *et al.* [5] is a technique that allows verification of a property on a single machine, by composing it with reduced versions of the other machines in the system. Here the reduction is based on *output variable* observability and it is property *independent.*

Our approach is to extract from the component machines the information relevant to the verification of a given property, and use only this to build the representation of a reduced system that preserves all of the behavior needed to verify the property. To verify a property on a system of interacting finite state machines with the technique described here, we first reduce every machine with respect to the property and with respect to the interaction with the other machines, compose the machines together, and finally check this reduced composition. This reduction is property *dependent*. The aim of this work is to show that in some cases the size of the representation of these reduced machines is smaller than the size of the representation of the entire system, thus increasing the potential to cope with otherwise intractable systems.

In this work we represent Boolean relations as characteristic functions implemented by BDDs [1]. Our ultimate target is to verify an arbitrary CTL formula on a complex system of interacting finite state machines without going through the computation of the full composition machine which could be much too big as a BDD. We do not give any guarantee that our reduction technique will achieve this goal. However, we apply a number of BDD manipulation heuristics (namely projection cofactor and smoothing) that give good results in many cases.

This report is organized as follows: In section 2 and 3 we present some basic concepts on transition relations and paths. In section 4 we present the CTL logic. In section 5 we describe in detail the CTL verification algorithms for a single machine. In section 6 we introduce the concepts of automatic reduction and CTL verification of a system of interacting machines. In section 7 we apply our technique to a simple example. In sections 8 and 9 we present a discussion of the work and its potential developments. Detailed proofs of the theorems in the body of the paper are given in the appendix.

In the sequel of this paper, we use the following notation:

1. S_xT - existential quantification of the variable x on the relation T. That is,

$$S_xT = T_{\overline{x}} + T_x.$$

   This is equivalent to projecting T onto the subspace orthogonal to x.

2. $[T]_{x \to x'}$ - substitution of each occurrence of the variable x in T by the variable x'.

3. The symbol "$\cdot$" will be used for Boolean AND, and the symbol "+" will be used for Boolean OR.

2 Transition Relations

Transition relations are at the core of our approach.

Definition 1 A *finite state machine* with states X and inputs I, is uniquely specified by the *transition relation* $T \subseteq X \times I \times X$. For computational purposes, we represent the set T as a characteristic function. Note that if T has n binary inputs, then the input alphabet I consists of 2^n elements. ■

Definition 2 Each $(x, i, x') \in T$ is a *transition* from *present state* $x \in X$ to *next state* $x' \in X$. A transition is a **single minterm** in the space $X \times I \times X$ (see Figure 1). ■

Definition 3 An *edge* $\langle x, x' \rangle$ is a subset of T such that each transition in $\langle x, x' \rangle$ has present state x and next state x'. In other words, an edge $\langle x, x' \rangle$ is the union of all transitions (x, i, x') in T. Formally, $\langle x, x' \rangle = T \cdot x \cdot x'$ (see Figure 1). ■

S	a	b	S'	T
x	0	0	x'	1
x	0	1	x'	0
x	1	0	x'	1
x	1	1	x'	1

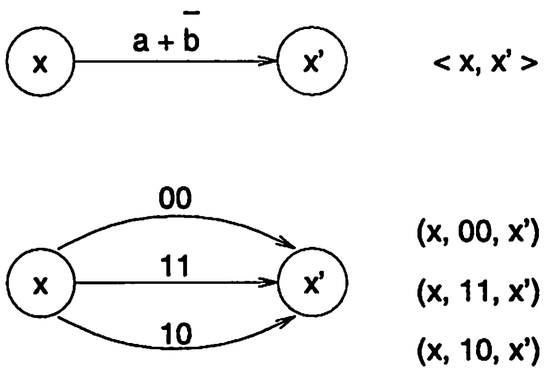

Figure 1: Edges and Transitions

3 Paths

In Section 5, we use the notion of path to define the output of the model checker computations.

Definition 4 A *path* $\pi(x_0, x_k)$ in T, of length k, is a finite sequence of edges

$$\langle x_0, x_1\rangle, \langle x_1, x_2\rangle, \ldots, \langle x_{k-1}, x_k\rangle.$$

A path is defined inductively as follows:

1. An edge $\langle x, y\rangle \subseteq T$ is a path $\pi(x, y)$ in T.
2. If $\pi(x, y)$ is a path in T and $\langle y, z\rangle \subseteq T$, then the union $\pi(x, y) + \langle y, z\rangle$ is a path $\pi(x, z)$ in T. ∎

This definition requires that a path have a unique starting vertex and a unique terminal vertex. Otherwise, a path may have cycles. In other words, from the start vertex, you should be able to traverse all edges of a path without "lifting your pencil" (see Figure 2.)

Notice that edges, paths, and transition relations are all sets of transitions, that is sets of minterms in the space $X \times I \times X$. Thus, we use the following notation:

$$(x, i, x') \in \langle x, y\rangle \subseteq \pi \subseteq T.$$

Definition 5 An *infinite path* $\pi(x_0, x_k)$ is a path such that $k = i$, for $0 \leq i \leq k$. That is, a path that terminates in a cycle. (see Figure 3) ∎

4 Computation Tree Logic

Computation Tree Logic (CTL) can be used to assert properties about FSMs. The CTL formula syntax is outlined as follows. There are two *path quantifiers*:

1. $\forall$ - for all computation paths, and

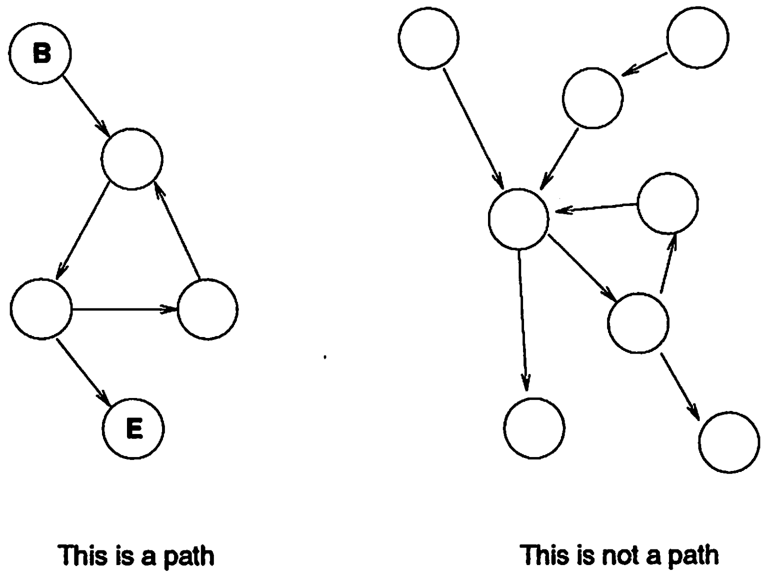

Figure 2: Paths

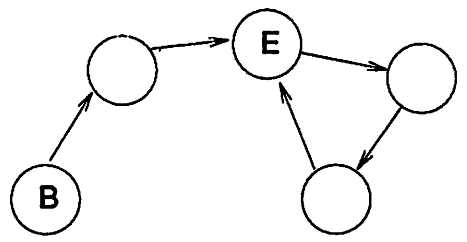

Figure 3: Infinite Path

2. $\exists$ - for some computation path.

There are four *forward-time operators*:

1. G - globally or invariantly,
2. F - sometime in the future,
3. X - nexttime,
4. U - until.

Thus, there are eight different combinations of path quantifiers and forward-time operators, which are listed below. In each case, the formula holds at a particular state s [2]. In our approach, we actually compute the set of states at which a formula holds, and then check to see if the set contains s.

In each of the following, f and g are *propositions*, or Boolean functions, of the state variables (i.e. $f : X \to \{0,1\}$). If f is a function of present states, it will be written as $f(x)$, and if a function of next states, it will be written as $f(x')$.

CTL formulas can be nested. That is, the propositions (f and g) of a CTL formula can be CTL formulas themselves. We will refer to non-nested formulas as *simple* formulas. This point is expanded in section 6.2.

1. $s \models \exists X f$ - s has a successor s' such that f is true at s'.
2. $s \models \exists G f$ - There is an infinite path starting at s such that f holds at each state on the path.
3. $s \models \exists [f U g]$ - There exists a path of length zero or more starting at s such that f is true until g is true.
4. $s \models \forall X f$ - f holds at all successor states of s (f must hold at the next state).
5. $s \models \exists F f$ - For some path starting at s, there exists a state on the path at which f holds (f is possible in the future).
6. $s \models \forall F f$ - For every path starting at s, there exists a state on the path at which f holds (f is inevitable in the future).
7. $s \models \forall G f$ - For every infinite path starting at s, at every node on each path, f holds (f holds globally or invariantly along all paths).
8. $s \models \forall [f U g]$ - For every path of length zero or more starting at s, f holds until g holds.

Normally, all eight CTL formulas would be expressed in terms of formulas 1 through 3. However, because we define the output of our model checker as a set of transitions, and not as a set of states, this is not convenient. Specifically, formula 3 does not fit cleanly into our formulation of the model checker. Thus, we introduce the following formula which does fit nicely into our definitions, and then show how we can express formula 3 in terms of this formula.

9. $s \models \exists [f R\, g]$ - There exists a path of length one or more starting at s such that f is repeatedly true until g is true; that is, $\overline{f} \cdot g$ cannot be true at state s.

Keep in mind that formula 9 is used only for computational purposes.

Definition 6 Formulas 1, 2, and 9 are the *base* CTL formulas. ∎

Proposition 7 CTL formulas 3 through 8 can be expressed in terms of a Boolean combination of the base CTL formulas and propositions of the state variables.

5 Model Checker

The input to our model checker is a transition relation T and a base CTL formula F. The output is a set of transitions T^* such that F holds at the present state of each transition. Thus, to produce the set of states in T for which F holds, we simply smooth away the input and next state variables from T^*.

As shown in Proposition 7, we can express formulas of type 3 through 8 in terms of the base formulas. Thus for any CTL formula, we can produce the set of states at which the formula holds by smoothing the input variables and the next state variables from the output of the model checker.

Formally, let $\mathcal{F}$ be the set of possible base CTL formulas, and let $(X \times I \times X)$ be the set of possible transition relations. In the definitions and propositions that follow, $T(x, i, x')$ is any transition relation, and F is any CTL formula in $\mathcal{F}$.

5.1 Model Checker Computations

Definition 8 The *model checker* implements a function $mc : (X \times I \times X) \times \mathcal{F} \rightarrow (X \times I \times X)$. We denote the output of the model checker as T^*, that is, $T^* = mc(T, F)$. ■

We next precisely define the output of the model checker, and show how to compute the output, for each of the three base formulas.

Definition 9 ($\exists X f$) The transition relation $T^* = mc(T, \exists X f)$ contains all transitions $(x, i, x') \in T$ such that $f(x') = 1$. That is,

$$T^* = \{(x, i, x') \in T | f(x') = 1\}.$$

Or equivalently, T^* is the union of all paths $[\langle x, x'\rangle]$ of length one such that $f(x') = 1$ (i.e. $T^* = \bigcup[\langle x, x'\rangle]$ s.t. $f(x') = 1$) (see Figure 4).

T^* is computed as follows:

$$T^* = f(x') \cdot T \quad ■$$

Definition 10 ($\exists G f$) A *$\pi_{G(f)}$-path* is an infinite path such that for all $\langle x, x'\rangle \subseteq \pi_{G(f)}$,

$$f(x) \cdot f(x') = 1.$$

Then, the transition relation $T^* = mc(T, \exists G f)$ contains all transitions $(x, i, x') \in T$ such that $(x, i, x') \in \pi_{G(f)}$ for some $\pi_{G(f)}$-path. That is,

$$T^* = \{(x, i, x') \in T | \exists\ \pi_{G(f)} \text{ s.t. } (x, i, x') \in \pi_{G(f)}\}.$$

Or equivalently, T^* is the union of all $\pi_{G(f)}$-paths (i.e. $T^* = \bigcup \pi_{G(f)}$ (see Figure 4).

T^* is found by the following *greatest* fixed point computation. The initial set T_0 is all edges from and to states where f is true. T_0 is finite. At each iteration, edges in T_n that go to present states not in T_n are removed from T_n to produce T_{n+1}. The iteration terminates when $T_{n+1} = T_n$. This iterative process must terminate because at each iteration we either remove at least one edge, or we have reached the fixed point.

$$\begin{aligned} T_0 &= T \cdot f(x) \cdot f(x') \\ T_{n+1} &= T_n \cdot (S_i S_{x'}(T_n))_{x \rightarrow x'} \\ T^* &= T_n, \text{ s.t. } T_{n+1} = T_n \quad ■ \end{aligned}$$

Definition 11 ($\exists[f R\ g]$) A *$\pi_{R(f,g)}$-path* is a path $\pi(x_0, x_k)$ such that for all $\langle x_{j-1}, x_j\rangle$, where $1 \leq j < k$

$$f(x_{j-1}) \cdot f(x_j) = 1,$$

and such that for $\langle x_{k-1}, x_k\rangle$

$$f(x_{k-1}) \cdot g(x_k) = 1.$$

Then, the transition relation $T^* = mc(T, \exists[f R g])$ contains all transitions $(x, i, x') \in T$ such that $(x, i, x') \in \pi_{R(f,g)}$ for some $\pi_{R(f,g)}$. That is,

$$T^* = \{(x, i, x') \in T | \exists\ \pi_{R(f,g)} \text{ s.t. } (x, i, x') \in \pi_{R(f,g)}\}.$$

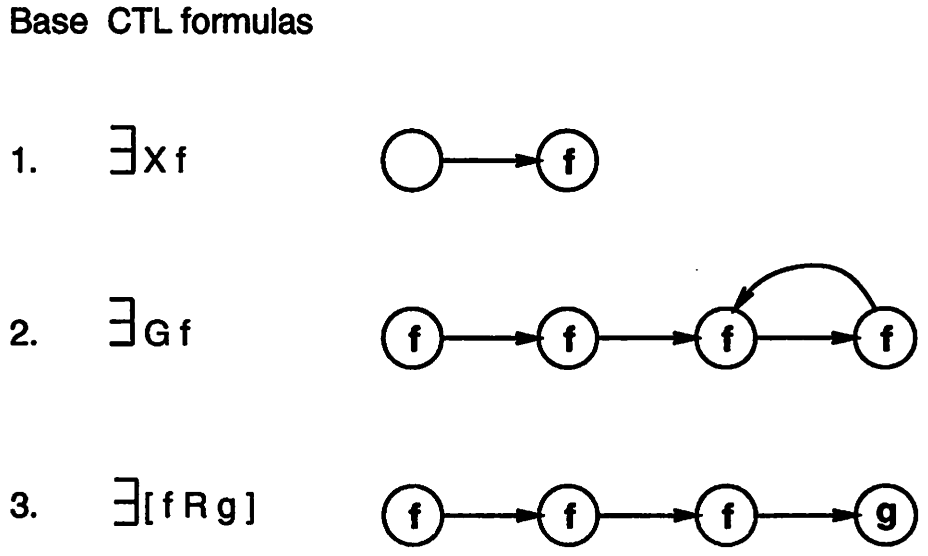

Figure 4: Base CTL Formulas

Or equivalently, T^* is the union of all $\pi_{R(f,g)}$-paths (i.e. $T^* = \bigcup \pi_{R(f,g)}$) (see Figure 4).

T^* is found by the following *least* fixed point computation. The initial set T_0 is all edges from states where f is true to states where g is true. At each iteration, edges from states where f holds to present states in T_n are added to T_n to produce T_{n+1}. The iteration terminates when $T_{n+1} = T_n$. This iterative process must terminate because at each iteration we either add a edge from the finite set T, or we have reached the fixed point.

$$\begin{aligned}\tilde{T} &= T \cdot f(x) \cdot f(x') \\ T_0 &= T \cdot f(x) \cdot g(x') \\ T_{n+1} &= \tilde{T} \cdot (S_i S_{x'}(T_n))_{x \to x'} + T_n \\ T^* &= T_n, \text{ s.t. } T_n = T_{n+1} \quad \blacksquare\end{aligned}$$

Our formulation of the model checker is slightly different than that of the CMU model checker [4]. Whereas the sets in our intermediate fixed point computations are transition relations, in CMU's computations, they are states. We choose this strategy because when we work on a system of many machines, we want to maintain precise control over which transitions are needed, and which can be discarded, for subsequent calculations (see Section 6). If we only maintain the set of states that satisfy the formula, then to retrieve a transition relation, we must take the product of this set with the original transition relation. However, for a given state satisfying the formula, we may not need all the transitions emanating from this state, and thus we would be stuck with unnecessary transitions.

5.2 Model Checker Input and Output Relations

In this section, we prove two properties on the relation between the input and output of the model checker.

Proposition 12 Let $T^* = mc(T, F)$. Then $T^* \subseteq T$, for any base CTL formula F.

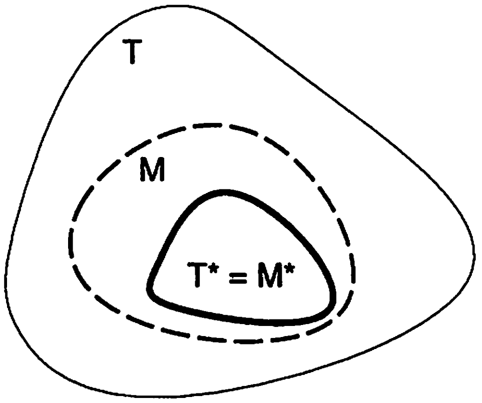

Figure 5: Property of Model Checker

This result simply says that the output of the model checker T^* is contained in the input T. This follows since the model checker only removes transitions from T; it never adds transitions. As a consequence, the transition relation returned by the model checker generally describes an incompletely specified machine.

Proposition 13 Let $T^* = mc(T, F)$. If M is another transition relation such that $T^* \subseteq M \subseteq T$, then $mc(M, F) = T^*$. (see Figure 5.)

This is a powerful result. Think of T as the full transition relation for a machine. As usual, T^* is the output of the model checker when T is the input. Proposition 13 tells us that we can use as input to the model checker any transition relation M that is contained in T and contains T^*, and still get T^* as output. Intuitively, this follows since the model checker only removes transitions from the input.

6 Automatic Reduction

Our goal is to apply the model checker to a *system* of interacting machines. However, to use the model checker, we need a single transition relation representing the composition of the component machines. If we take the composition of the full component machines, there is a danger that the size of the resulting BDD will be prohibitively large. To avoid this danger, we wish to apply reductions to the component machines before forming the composition.

Consider a system of n interacting machines, $A_1, A_2, \ldots, A_n$. The global state of the system is $X = [x_1, \ldots, x_n]$. Each machine A_i has present state variable x_i and next state variable x_i', and takes as input $X_{\neq i} = [x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n]$, the present state variables of the other machines. We assume that all external inputs to the system have been smoothed out. Thus, we represent A_i by the transition relation $A_i(x_i, X_{\neq i}, x_i') = A_i(X, x_i')$.

As suggested above, the naive method to verify a formula F on the system is to apply the model checker to the complete product

$$M = \prod_{i=1}^{n} A_i$$

```
function $cmc$(array[$A_i$], $type$, $f$, $g$) {

1     for $i = 1$ to $n$ {      /* project $f$ on $x_i$ and run $mc$ on $A_i$ */
2         $A_i^* = mc(A_i, type, S_{X_{\neq i}} f(X), S_{X_{\neq i}} g(X))$;
3     }

4     $R = \prod (S_{x_i'} \cdot A_i^*)$;   /* reducing term */
5     for $i = 1$ to $n$ {      /* find lower bound of onset of $A_i$ */
6         $A_i^{*\prime} = rdt(A_i^*, R)$;
7     }

8     for $i = 1$ to $n$ {   /* use don't cares to minimize BDD for $A_i$ */
9         $\hat{A}_i = min\_bdd(A_i, A_i^{*\prime})$;
10    }

11    $\hat{M} = mc(\prod \hat{A}_i, type, f(X), g(X))$;   /* run $mc$ on reduced product */
12    $Q(X) = S_{X',I} \hat{M}$;      /* states that satisfy $F$ */
13    return $Q(X)$;
  }
```

Figure 6: Compositional Model Checker

We denote the output of this procedure as M^*, that is, $M^* = mc(M, F)$.

We seek to exploit the power of Proposition 13 by finding a transition relation $\hat{M}$ such that $M^* \subseteq \hat{M} \subseteq M$. Furthermore, we want to choose an $\hat{M}$ with a small BDD representation. To achieve this, we reduce each A_i as much as possible to yield $\hat{A}_i$, and then take the product of the $\hat{A}_i$ to produce $\hat{M}$.

6.1 Compositional Model Checker

We have developed a procedure *cmc* (Compositional Model Checker) to check a simple (non-nested) CTL formula F on a system of machines. The inputs to *cmc* are F and an array of the component machines *array[A_i]*. The output is Q, the set of states of the system that satisfy F. A CTL formula F is actually a structure composed of a *type* specifying the type of formula ($\exists G$, $\exists[\ R\ ]$, $\exists X$, etc.) and two Boolean functions $f(X)$ and $g(X)$ on the states of the total system. Note that g will be ignored for all types but $\exists[\ R\ ]$. For example, if we have $F = \exists G(x_1 + x_2)$, we use $type = \exists G$, $f = x_1 + x_2$, $g = 0$, and call $Q = cmc(array[A_i], type, f, g)$.

The *cmc* procedure consists of four phases (see Figure 6). In phase 1 (lines 1-3), the model checker is used to extract a subset of each component machine that satisfies the formula F when F is projected onto each component. Phase 2 (lines 4-7) further reduces each machine by finding a smaller subset of each machine that covers the entire system. Phase 3 (lines 8-10) uses the don't care transitions derived for each machine to find a cover for each machine with a small BDD representation. Finally, phase 4 (lines 11-13) applies the model checker to the composition of the reduced machines. Below, we describe each phase in detail, and then prove the correctness of this procedure.

6.1.1 Phase 1

For a formula F with proposition $f(X)$ to hold at a state $[x_1 = b_1, x_2 = b_2, \ldots, x_n = b_n]$ of the product machine, the projection of F onto each machine A_i, that is F with proposition $f(x_i) = S_{X_{\neq i}} f(X)$, must hold at state $(x_i = b_i)$ of A_i. To find the states of A_i which satisfy F, we simply apply the model checker to the single machine A_i using $f(x_i)$ as the proposition for F to yield the output A_i^*.

In applying the model checker to A_i, we ignore the inputs to A_i from the other machines A_j in the system. This can be thought of as replacing each A_j by an abstract version of A_j which can non-deterministically produce all the outputs which A_j can produce. This is equivalent to the COSPAN concept of "freeing" A_j [6]. Thus, the interaction between A_i and the other machines is totally disregarded in this phase. The behavior of A_i, when it is part of the larger system, is contained by the behavior of A_i when it is independent of the other machines. Therefore any state that satisfies F in the whole system will still satisfy F when projected onto A_i. Conversely, if a state $(x_i = b_i)$ in A_i fails to satisfy the projection of F onto A_i, then no state of the product machine with an i-th component equal to b_i will satisfy F.

Thus, the purpose of this phase is to identify transitions of A_i which do not belong to paths which are specified by F. These "bad" transitions are removed from A_i by the model checker to yield A_i^*. Actually, these transitions are don't cares; if we do not remove them now, then they will be implicitly removed in phase 4 when the model checker is applied to the reduced product.

It should be emphasized that this is the only phase where we apply "abstraction" techniques. In this case, we apply a trivial form of abstraction: in simplifying A_i, each of the other machines is replaced by a one state machine that non-deterministically selects all possible outputs that the full component machine can produce.

As an example of phase 1, consider a system of two machines A and B, and a formula $F = \exists G f$ where f is a function of the present state of A. The projection of f onto A is still f, and thus $mc(A, F)$ extracts the infinite paths of A where f always holds. On the other hand, the projection of f onto B is the tautology, and thus $mc(B, F)$ simply extracts all the infinite paths of B. This makes sense since an infinite path in $A \cdot B$ is the product of infinite paths in A and B.

6.1.2 Phase 2

The product $\dot{M} = \prod A_i^*$ contains all the transitions specified by a formula F. As we show below, if the model checker is applied to $\dot{M}$, the output will be M^*. However, there are still more transitions that we can remove from each A_i^* without affecting the overall computation. Consider a machine A_i^*. In the space of $X \times X'$, $A_i^* \supseteq \dot{M}$. If this containment is strict, then there will be some transitions of A_i^* that "disappear" when A_i^* is composed with the other machines. This occurs when the present state-input pair of a transition in A_i^* has an empty intersection with all other present state-input pairs of the other machines.

We can identify the transitions of A_i^* that will disappear in the product $\dot{M}$ without *explicitly* forming the product. A_i^* *is* interested in the present state variables of the other machines, because these are the inputs to A_i^*; and A_i^* *is* interested in the inputs to the other machines because these determine the possible transitions in M. On the other hand, machine A_i^* is *not* interested in the next state variables of the other machines. Thus, to remove the transitions of A_i^* that will disappear, we smooth out the next state variables from the other machines, and then take the product of these reduced machines with A_i^*. The result of this operation is $A_i^{*\prime}$.

Proposition 14 Consider a system of interacting machines composed of $A_1, A_2, \ldots, A_n$. Let

$$R(X) = \prod_j (S_{x'_j} A_j)$$

Then $A_i^{*\prime} = rdt(A_i, R) = A_i \cdot R$ is the smallest subset of A_i which contains $\prod_j A_j$ and is independent of $X'_{\neq i}$.

In lines 4-7 of *cmc*, we apply Proposition 14 with the A_j of Proposition 14 replaced by the A_i^* of *cmc*.

$A_i^{*\prime}$ is a lower bound on the onset of A_i which satisfies the conditions in Proposition 14. That is, it contains the minimum number of transitions of A_i needed to represent the relevant behavior of A_i, with respect to F, in the larger system. As shown below, the model checker applied to the product of the $A_i^{*\prime}$ still gives us M^*.

To reiterate, there is no "abstraction" taking place in phase 2. We are simply identifying transitions of the component machines that do not survive when the product of the machines is taken.

What have we achieved at this point? *If* we were using an *explicit* representation for the transition relations, then the $A_i^{*\prime}$ would be exactly what we desire. Namely, by minimizing the number of transitions in each component machine, we would be minimizing the size of their corresponding representations. Consequently, we would have minimized the size of the relevant part of the product machine needed to verify F.

However, we are not using an explicit representation. Instead, we are using BDDs, an *implicit* representation, where the size of a BDD representing a relation does not directly correspond to the number of elements in the relation. Is all lost? No. We can think of the difference $A_i - A_i^{*\prime}$ as a don't care set, and try to find a function between A_i and $A_i^{*\prime}$ with a small BDD representation. Thus, phases 1 and 2 can be seen as an attempt to maximize the size of the don't care set of each component machine.

6.1.3 Phase 3

From phase 2, we have the minimum onset $A_i^{*\prime}$ for each machine A_i. Also, we have the maximum onset, namely A_i. The hypothetical procedure *min_bdd* in phase 3 finds a function between $A_i^{*\prime}$ and A_i with a minimum BDD representation. Such a procedure for BDDs is analogous to ESPRESSO for the sum of products representation.

The function *min_bdd* in line 9 is implemented using the *projection cofactor*. The projection cofactor, termed the generalized cofactor in Touati *et. al.* [3], is a heuristic to find a cover with a small BDD for an incompletely specified function. In the following discussion, let A be any machine A_i. For our application, we apply the projection cofactor to each machine, using the don't care set

$A - A^{*'}$ derived from phase 2. Thus, we cofactor the minimum onset $A^{*'}$ by the care set $\overline{A} + A^{*'}$ to yield $\hat{A}$. That is,

$$\hat{A} = A^{*'}_{(A^{*'}+\overline{A})}$$

Actually, since $A^{*'} \subseteq A$, $\hat{A}$ can also be calculated as

$$\hat{A} = A_{(A^{*'}+\overline{A})}$$

We must keep in mind that the projection cofactor is just a heuristic. According to Touati, "...in most cases, the BDD representation of f_c is smaller than the BDD representation of f." If a better method is found to find a small cover of an incompletely specified function, then we can apply the method directly in our technique in place of the call to *min_bdd*.

6.1.4 Phase 4 and Proof of Correctness

The final phase of the procedure *cmc* is to apply the model checker to the product of the $\hat{A}_i$ and the original formula F, to determine the states of the product machine that satisfy F. Even though many transitions of each component machine have been stripped away by phases 1 and 2, the model checker output of line 11 gives us the same result as if we had taken the naive approach and applied the model checker to the full product machine. This result is stated in the following theorem and proved rigorously in the appendix.

Theorem 15 Consider a system of interacting machines composed of $A_1, A_2, \ldots, A_n$. Let F be any CTL formula. Then

$$cmc(\prod A_i, F) = cmc(\prod \hat{A}_i, F).$$

6.2 Nested Formulas

As mentioned in section 4, CTL formulas can be nested, that is the propositions can either be explicit sets of states or formulas to be computed. For example, the following CTL formula

$$F = \forall G(req \rightarrow \forall F(ack + reset))$$

is a 4-level nested formula (see Figure 7.)

Computing nested CTL formulas in our compositional approach is handled in exactly the same way it is handled in other techniques where the full product machine is computed [2] [4]. We traverse the formula from the bottom up (from the leaves to the root.) At each level of nesting we compute one or more simple CTL formulas whose propositions are either given or computed from the previous level. In the example above, the atomic propositions $req(X)$, $ack(X)$, and $reset(X)$ are given as BDDs. The formula is then verified as follows.

Let $M = array[A_i]$. The formula $F = \forall G(req \rightarrow \forall F(ack + reset))$ is satisfied by the set $Q(X)$ of states of M given by

$$\begin{aligned} Q_0(X) &= cmc(M, ack(X) + reset(X)) && (1)\\ Q_1(X) &= cmc(M, \forall F Q_0(X)) && (2)\\ Q_2(X) &= cmc(M, req(X) \rightarrow Q_1(X)) && (3)\\ Q(X) &= cmc(M, \forall G Q_2(X)) && (4) \end{aligned}$$

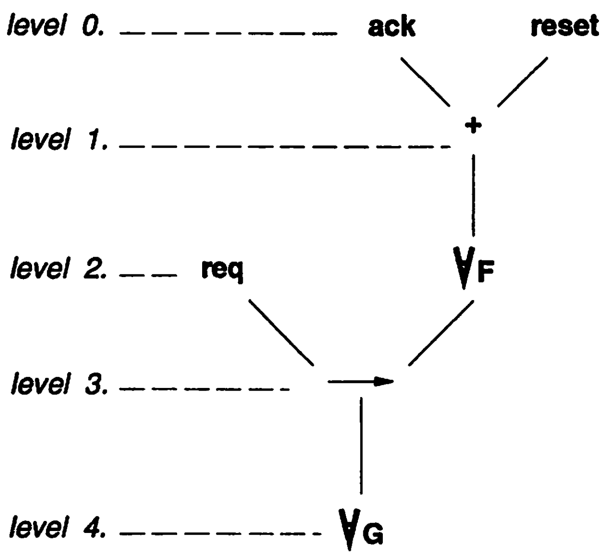

Figure 7: Nested CTL Formula as a Tree

To compute an arbitrary nested formula F, we embed the *cmc* function, which verifies a simple CTL formula, in a recursive procedure *rcmc* (Recursive Compositional Model Checker) that explores the tree of the formula F from the leaves to the root (see Figure 8.)

7 Example

The simple example in this section shows how a system of two interacting machines is verified by our compositional technique. The main purpose of the example is to demonstrate how we handle a property defined on multiple machines.

The two components A and B are

$A(x_1, x_2, y_1, y_2, x_1', x_2')$

x_1x_2	y_1y_2	$x_1'x_2'$	$y_1'y_2'$
01	00	10	−−
01	01	11	−−
01	10	11	−−
01	11	01	−−
10	0−	01	−−
10	10	01	−−
10	10	10	−−
10	11	11	−−
11	0−	11	−−
11	1−	10	−−

$B(x_1, x_2, y_1, y_2, y_1', y_2')$

x_1x_2	y_1y_2	$x_1'x_2'$	$y_1'y_2'$
−−	00	−−	11
11	00	−−	10
−0	10	−−	11
−1	10	−−	00
10	11	−−	11
0−	11	−−	10
11	11	−−	10

```
function $rcmc$(array[$A_i$], $type$, $f$, $g$) {

    if ($NESTED(f)$) {      /* if $f$ is nested, apply $Rcmc$ */
        $f = rcmc$(array[$A_i$], $f.type$, $f.f$, $f.g$);
    }

    if ($NESTED(g)$) {      /* if $g$ is nested, apply $Rcmc$ */
        $f = rcmc$(array[$A_i$], $g.type$, $g.f$, $g.g$);
    }

    $Q = cmc$(array[$A_i$], $type$, $f$, $g$);  /* run $cmc$ on root formula */
    return $Q$;
}
```

Figure 8: Recursive Compositional Model Checker

The product machine is

$A \cdot B$

x_1x_2	y_1y_2	$x_1'x_2'$	$y_1'y_2'$
01	00	10	11
01	10	11	00
01	11	01	10
10	00	01	11
10	10	01	11
10	10	10	11
10	11	11	11
11	00	11	11
11	00	11	10
11	10	10	00
11	11	10	10

We want to verify the formula

$$F(x,y) = \exists[(x_1x_2 + x_1\bar{x}_2y_1\bar{y}_2)U(x_1\bar{x}_2(\bar{y}_1\bar{y}_2 + y_1y_2))]$$

The projection of F onto A is

$$F^A(x) = \exists[(x_1x_2 + x_1\bar{x}_2)U(x_1\bar{x}_2)]$$

The projection of F onto B is

$$F^B(y) = \exists[1U(\bar{y}_1\bar{y}_2 + y_1y_2)]$$

We verify the two components obtaining the following results

A^*

x_1x_2	y_1y_2	$x'_1x'_2$	$y'_1y'_2$
10	10	10	--
11	0-	11	--
11	1-	10	--

B^*

x_1x_2	y_1y_2	$x'_1x'_2$	$y'_1y'_2$
--	00	--	11
11	00	--	10
-0	10	--	11
-1	10	--	00
10	11	--	11
0-	11	--	10
11	11	--	10

Finally, we check $(A^* \cdot B^*)$ and obtain

$(A^* \cdot B^*)^* = (A \cdot B)^*$

x_1x_2	y_1y_2	$x'_1x'_2$	$y'_1y'_2$
10	10	10	11
10	11	11	11
11	00	11	11
11	00	11	10
11	10	10	00
11	11	10	10

Figure 9 illustrates the transformations on the two FSMs and on the product machine.

8 Future Work

8.1 Theoretical

Below are some theoretical issues which deserve future attention.

1. We could use the *rdt* reduction on the initial components of the system we are to verify. Would this enhance the performance of the fixed point computations?

2. Can we apply FSM state minimization algorithms (like Hopcroft's) to achieve further degrees of reduction?

3. Are there further property dependent reductions that can be made? Proposition 14 seems to rule out any further property independent reductions, such as a "ping-pong" procedure.

4. Is it effective to repartition the transition relations to better suit a system to the verification of a given property or type of property.

5. Are there reductions that can be made just knowing the type of CTL formula, without knowing the specific propositions in the formula? For example, we can verify the formula $\exists G\ true$ with the tautology in place of the proposition f and find the set of infinite paths in a system.

6. What are the time and space complexities of our methods? The model checking algorithms are linear in the number of transitions. However, building a BDD can be exponential in the number of nodes.

7. We can consider extending our approach to other verification logics like ω-regular properties and ECTL. Can these formalisms fit in our compositional scheme in a natural way, or do we need to change something?

8. Coudert talks about the difference between backward traversal and forward traversal of transition relations. We need to understand this difference and see if we observe it in real examples. If so, can we formulate the model checker to use forward traversal?

8.2 Experimental

1. Develop a "real-world" example of a complex system of interacting finite state machines.

2. Develop a software system implementing our ideas. The single machine model checker is finished. We need to complete the compositional part.

3. Develop a counterexample facility to help the designer understand the results returned by the verification system.

Only by developing a working system will we know if our techniques produce practical benefits.

9 Conclusions

Our spring '91 290H project [8] planted the seed for our current work. In that project, we formulated a technique to verify a simple CTL formula with propositions on one machine, in a system of two machines. Besides these limitations on the properties that we could verify and on the number of machines, the technique could not handle nested formulas. We left open the following questions:

- Can we prove the correctness of our technique?
- Can we verify properties defined on more than one machine?
- Can we verify arbitrary (nested) formulas?
- Can we write a program implementing our technique?
- Can we demonstrate that this technique gives useful results in some meaningful benchmark?

Significant progress has been made this semester. We have overcome all the limitations we had and we have given a positive answer to all but one of the questions above. The only (and very important!) aspect we still we need to work out is the practical utility of our technique. To do this, we need to find some real examples that show some serious improvement by using this compositional technique.

In our spring report, we were particularly pessimistic about verifying nested formulas. As it turned out, this issue reduced to the problem of verifying formulas with propositions on multiple machines. This problem was solved by projecting the propositions on the component machines, reducing each machine independently, and then verifying the original formula on the reduced product. In this way, each level of nesting is handled in a natural way and the computation of a nested formula reduces to a straightforward recursive procedure that traverses the formula from the leaves to the root. With this technique, we can handle arbitrary CTL formulas, such as

$$F = \forall G(req \rightarrow \forall F(ack + reset)).$$

References

[1] R. E. Bryant. "Graph-based algorithms for boolean function manipulation." IEEE Trans. Comput., C-35(8), 1986.

[2] E. M. Clarke, E. A. Emerson, and P. Sistla. "Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications." ACM Trans. Prog. Lang. Syst., 8(2):244-263, 1986.

[3] H. J. Touati, Hamid Savoj, B. Lin, R. K. Brayton, and Alberto Sangiovanni-Vincentelli. "Implicit State Enumeration of Finite State Machines using BDDs." In *ICCAD'90*, November, 1990.

[4] J. Burch, E. Clarke, K. McMillan, David Dill. "Sequential circuit verification using symbolic model checking", DAC 1990.

[5] E. M. Clarke, D. E. Long, K. L. McMillan, "Compositional Model Checking," Proc. of the 4th IEEE Symposium on Logic in Computer Science, June, 1989, Asilomar, CA.

[6] Z. Har'El, R. P. Kurshan, "COSPAN User's Guide," AT&T Bell Laboratories, 1987.

[7] J. E. Hopcroft, "An *nlogn* Algorithm for Minimizing the States in a Finite Automaton," In *The Theory of Machines and Computation,* New York: Academic Press, pp. 189-196, 1971.

[8] M. Chiodo, K. Kodandapani, T. Shiple, "New Ideas on Compositional Model Checking," Term Project, EE290H, Spring 1991.

A Set Properties

In the following proofs we make use of some general properties of sets.

Let A, B and C be subsets of S.

1. $A \cdot B \subseteq A$
2. If $A \subseteq C$ and $B \subseteq C$, then $A + B \subseteq C$
3. If $A \subseteq B$, then $AC \subseteq BC$
4. If $A \cdot C \subseteq B$, then $A \cdot C \subseteq B \cdot C$
5. If $A \subseteq B \subseteq A$, then $A = B$

B Proofs

Proposition 7 CTL formulas 3 through 8 can be expressed in terms of a Boolean combination of the base CTL formulas and propositions of the state variables.
Proof (by construction)

4. $\exists[fUg] = \exists[fRg] + g$
5. $\forall Xf = \overline{\exists X\overline{f}}$
6. $\exists Ff = \exists[trueRf] + f$
7. $\forall Ff = \overline{\exists G\overline{f}}$
8. $\forall Gf = \overline{\exists F\overline{f}}$
9. $\forall[fUg] = \overline{(\exists[(\overline{g})R(\overline{f} \cdot \overline{g})] + (\overline{f} \cdot \overline{g}))} \cdot \overline{\exists G\overline{g}}$. ∎

We seek to prove that the output T^* of the model checker is contained in the input T. Each of Lemmas 16, 17 and 18 establish the principal result needed for each of the 3 base CTL formulas, respectively, and then Proposition 12 states the property we seek.

Lemma 16 Let $T^* = mc(T, \exists Xf)$. Then $T^* \subseteq T$.
Proof By construction, T^* is a restriction of T. The lemma follows from the first set property above. ∎

Lemma 17 In the model checker computation $mc(T, \exists Gf)$, at each iteration of the greatest fixed point computation, $T_n \subseteq T$.
Proof (by induction)
Basis step: T_0 is a restriction of T,and thus $T_0 \subseteq T$.
Induction hypothesis: Suppose $T_n \subseteq T$ for some $n \geq 0$.
Induction step: Recall from Definition 10 that

$$T_{n+1} = T_n \cdot (S_i S_{x'}(T_n))_{x \to x'}$$

T_{n+1} is a restriction of T_n, and thus $T_{n+1} \subseteq T_n$. Then by the induction hypothesis, $T_{n+1} \subseteq T$. ∎

Lemma 18 In the model checker computation $mc(T, \exists[fRg])$, at each iteration of the least fixed point computation, $T_n \subseteq T$.
Proof (by induction)
Basis step: Since T_0 is a restriction of T, then $T_0 \subseteq T$.
Induction hypothesis: Suppose $T_n \subseteq T$ for some $n \geq 0$.
Induction step: Recall from Definition 11 that

$$\begin{aligned} \tilde{T} &= T \cdot f(x) \cdot f(x') \\ T_{n+1} &= \tilde{T} \cdot (S_i S_{x'}(T_n))_{x \to x'} + T_n \end{aligned}$$

$\tilde{T} \cdot (S_{x'}(T_n))_{x \to x'} \subseteq T$, since $\tilde{T} \subseteq T$. Also, by the induction hypothesis, $T_n \subseteq T$. Thus, by the second set property, $T_{n+1} \subseteq T$. ∎

Proposition 12 Let $T^* = mc(T, F)$. Then $T^* \subseteq T$, for any base CTL formula F.
Proof (by cases)
Case $F = \exists X f$: This is just Lemma 16.
Case $F = \exists G f$: From Lemma 17, $T_n \subseteq T$ holds for any n, and thus at the fixed point, it holds for $T_n = T^*$.
Case $F = \exists[fRg]$: From Lemma 18, $T_n \subseteq T$ holds for any n, and thus at the fixed point, it holds for $T_n = T^*$. ∎
Proposition 13 Let $T^* = mc(T, F)$. If M is another transition relation such that $T^* \subseteq M \subseteq T$, then $mc(M, F) = T^*$.
Proof (by cases)
Case $F = \exists X f$:

$$\begin{array}{rcccll} T^* & \subseteq & M & \subseteq & T & \\ f(x') \cdot T^* & \subseteq & f(x') \cdot M & \subseteq & f(x') \cdot T & \text{(Set Prop. 4)} \\ T^* & \subseteq & mc(M, F) & \subseteq & T^* & \text{(Definition 9)} \\ T^* & = & mc(M, F) & & & \text{(Set Prop. 5)} \end{array}$$

Case $F = \exists G f$:
By definition 10, T^* is the union of all $\pi_{G(f)}$-paths in T such that for all $\langle x, x' \rangle \subseteq \pi_{G(f)}$, $f(x) \cdot f(x') = 1$. That is, $T^* = \bigcup \pi_{G(f)}, \pi_{G(f)} \subseteq T$. Likewise, let $M^* = mc(M, F) = \bigcup \pi_{G(f)}, \pi_{G(f)} \subseteq M$. By the hypothesis, $T^* \subseteq M$. Since $mc(M, F)$ extracts all the $\pi_{G(f)}$-paths from M, it follows that $T^* \subseteq M^*$ (see figure).

On the other hand, also by the hypothesis, $M \subseteq T$. Thus, all $\pi_{G(f)}$-paths in M must exist in T; that is, $M^* \subseteq T^*$. Since $M^* \supseteq T^*$ and $M^* \subseteq T^*$, it follows that $M^* = T^*$ (see figure).
Alternate Proof of case $F = \exists G f$:
Let $p = f(x) \cdot f(x')$. Let $\tilde{T} = T \cdot p$. Likewise, let $\tilde{M} = M \cdot p$. The proof proceeds by the following series of implications:

$$\begin{array}{rll} & T^* \subseteq M \subseteq T & (1) \\ \Rightarrow & T^* \subseteq \tilde{M} \subseteq \tilde{T} & (2) \\ \Rightarrow & mc(\tilde{M}, F) = T^* & (3) \\ \Rightarrow & mc(M, F) = T^* & (4) \end{array}$$

(1) $\Rightarrow$ (2): From Definition 10 and by a simple inductive argument, $T^* \cdot p = T^*$. Thus, intersecting the terms in (1) with p gives (2).
(2) $\Rightarrow$ (3): Recall that by Definition 10, $T^* = \bigcup \pi_{G(f)}, \pi_{G(f)} \subseteq T$ and $\tilde{M}^* = \bigcup \pi_{G(f)}, \pi_{G(f)} \subseteq \tilde{M}$. For sake of contradiction, suppose that $T^* \neq \tilde{M}^* = mc(\tilde{M}, F)$. Then one of the following must be true:

1. $\exists \pi_{G(f)} \subseteq \tilde{M}^*$ such that $\pi_{G(f)} \not\subseteq T^*$. If $\pi_{G(f)}$ is in $\tilde{M}^*$, then it must exist in $\tilde{M}$. On the other hand, if $\pi_{G(f)}$ is not in T^*, then it must not be in $\tilde{T}$, since the model checker extracts from $\tilde{T}$ all infinite paths. Since $\pi_{G(f)}$ is in $\tilde{M}$ but not in $\tilde{T}$, then $\tilde{M} \not\subseteq \tilde{T}$, which contradicts (2).

2. $\exists \pi_{G(f)} \subseteq T^*$ such that $\pi_{G(f)} \not\subseteq \tilde{M}^*$. If $\pi_{G(f)}$ is not in $\tilde{M}^*$, then it must not be in $\tilde{M}$, since the model checker extracts from $\tilde{M}$ all infinite paths. Since $\pi_{G(f)}$ is in T^* but not in $\tilde{M}$, then $T^* \not\subseteq \tilde{M}$, which contradicts (2).

In both cases, we derive a contradiction, and thus, $T^* = \tilde{M}^*$.
(3) $\Rightarrow$ (4): From Definition 10, the first step in the computation $mc(M, F)$ is to remove transitions not in p from M, to yield $\tilde{M}$. Thus, the computation produces the same result whether we start from M or from $\tilde{M}$.
Case $F = \exists[fRg]$:
The proofs proceeds exactly as that for the case $\exists Gf$, except that "$\pi_{G(f)}$-paths" is replaced by "$\pi_{R(f,g)}$-paths". ∎
Proposition 14 Consider a system of interacting machines composed of $A_1, A_2, \ldots, A_n$. Let

$$R(X) = \prod (S_{x'_j} A_j)$$

Then $A'_i = rdt(A_i, R) = A_i \cdot R$ is the smallest subset of A_i which is independent of $X'_{\neq i}$ and contains $\prod A_j$.
Proof Clearly, the smallest subset of A_i that contains $\prod A_j$ is $\prod A_j$. However, $\prod A_j$ is not independent of $X'_{\neq i}$. By smoothing $\prod A_j$ with respect to $X'_{\neq i}$, that is $S_{X'_{\neq i}}(\prod A_j)$, we obtain the smallest subset of A_i which contains $\prod A_j$ and is independent of $X'_{\neq i}$ (by the properties of the smoothing operator). Since A_i is already independent of $X'_{\neq i}$, we have

$$\begin{aligned} S_{X'_{\neq i}}(\prod A_j) &= (S_{X'_{\neq i}}(\prod_{j \neq i} A_j)) \cdot A_i \\ &= (\prod_{j \neq i} (S_{x'_j} A_j)) \cdot A_i \cdot S_{x'_i} A_i \\ &= (\prod (S_{x'_j} A_j)) \cdot A_i \\ &= R \cdot A_i \end{aligned}$$

Lastly, $A'_i \subseteq A_i$ since A'_i is a restriction of A_i. ∎

Lemma 19 Consider a system of two interacting finite state machines A and B. Let F be a simple CTL formula with propositions defined on both A and B, and let $A^* = mc(A, F)$. Then

$$A^* \supseteq (A \cdot B)^*$$

Proof (by cases)
Case $F = \exists X f$:
Using Definition 9,

$$A^* = A \cdot S_{y'} f(x', y') \supseteq A \cdot B \cdot f(x', y') = (A \cdot B)^*$$

Case $F = \exists G f$:
The proof is by induction on the number of iterations in the greatest fixed point computation.
Basis step: $A_0 = A \cdot S_y f(x, y) \cdot S_{y'} f(x', y') \supseteq A \cdot B \cdot f(x, y) \cdot f(x', y') = (A \cdot B)_0$.
Induction hypothesis: Suppose $(A \cdot B)_n \subseteq A_n$ for some $n \geq 0$.
Induction step: Recall from Definition 10 that

$$A_{n+1} = A_n \cdot [S_{y,x'} A_n]_{x \rightarrow x'}$$

Since A_n is independent of y', we can smooth A_n with respect to y' without any effect. Furthermore, since $(S_y S_{x'} A_n)$ is independent of y, we can substitute y' for y without any effect. Thus,

$$A_{n+1} = A_n \cdot [S_{y,x',y'} A_n]_{x \rightarrow x', y \rightarrow y'}$$

Using the induction hypothesis and substituting $(A \cdot B)_n$ for A_n

$$\begin{aligned} A_{n+1} &\supseteq (A \cdot B)_n \cdot [S_{y,x',y'} (A \cdot B)_n]_{x \rightarrow x', y \rightarrow y'} \\ &\supseteq (A \cdot B)_n \cdot [S_{x',y'} (A \cdot B)_n]_{x \rightarrow x', y \rightarrow y'} \end{aligned}$$

But, by Definition 10

$$(A \cdot B)_n \cdot [S_{x',y'} (A \cdot B)_n]_{x \rightarrow x', y \rightarrow y'} = (A \cdot B)_{n+1}$$

Thus, $A_{n+1} \supseteq (A \cdot B)_{n+1}$. Since the greatest fixed point computation must terminate, there exists an n such that $A_n = A^*$ and $(A \cdot B)_n = (A \cdot B)^*$, then $(A \cdot B)^* \subseteq A^*$.
Case $F = \exists G[f R g]$:
The proof is by induction on the number of iterations in the least fixed point computation.
Basis step: $(A \cdot B)_0 = A \cdot B \cdot f(x, y) \cdot g(x', y') \subseteq A \cdot S_y f(x, y) \cdot S_{y'} g(x, y)) = A_0$
Induction hypothesis: Suppose $(A \cdot B)_n \subseteq A_n$ for some $n \geq 0$
Induction step: Recall from definition 11 that

$$A_{n+1} = A \cdot S_y f(x, y) \cdot S_{y'} g(x', y') [S_{y,x'} A_n]_{x \rightarrow x'} + A_n$$

Since A_n is independent of y', we can smooth A_n with respect to y' without any effect. Furthermore, since $(S_y S_{x'} A_n)$ is independent of y, we can substitute y' for y without any effect. Thus,

$$A_{n+1} = A \cdot S_y f(x, y) \cdot S_{y'} g(x', y') [S_{y,x',y'} A_n]_{x \rightarrow x', y \rightarrow y'} + A_n$$

Using the induction hypothesis and substituting $(A \cdot B)_n$ for A_n

$$\begin{aligned} A_{n+1} &\supseteq A \cdot S_y f(x, y) \cdot S_{y'} g(x', y') [S_{y,x'} (A \cdot B)_n]_{x \rightarrow x', y \rightarrow y'} + (A \cdot B)_n \\ &\supseteq A \cdot B \cdot f(x, y) \cdot g(x', y') \cdot [S_{x',y'} (A \cdot B)_n]_{y \rightarrow y', x \rightarrow x'} + (A \cdot B)_n \end{aligned}$$

But, by Definition 11

$$A \cdot B \cdot f(x, y) \cdot g(x', y') \cdot [S_{x',y'} (A \cdot B)_n]_{y \rightarrow y', x \rightarrow x'} + (A \cdot B)_n = (A \cdot B)_{n+1}$$

Thus, $A_{n+1} \supseteq (A \cdot B)_{n+1}$. Since the least fixed point computation must terminate, there exists an n such that $A_n = A^*$ and $(A \cdot B)_n = (A \cdot B)^*$, then $(A \cdot B)^* \subseteq A^*$. ∎

Proposition 20 Consider a system of interacting finite state machines $A_1, A_2, \ldots, A_n$. Let F be a simple CTL formula, and let $A^* = mc(A, F)$. Then

$$(\prod A_i)^* \subseteq \prod A_i^*$$

Proof (by induction)
Basis step: By Lemma 19, $(A_1 \cdot A_2)^* \subseteq A_1^*$ and $(A_1 \cdot A_2)^* \subseteq A_2^*$, and thus $(A_1 \cdot A_2)^* \subseteq A_1^* \cdot A_2^*$.
Induction hypothesis: Suppose $(\prod^k A_i)^* \subseteq \prod^k A_i^*$ for some $k \geq 0$
Induction step: First consider that $(\prod^{k+1} A_i)^* = (A_{k+1} \cdot \prod^k A_i)^*$. By observing that $\prod^k A_i$ is itself a transition relation, and applying lemma 19, we have $(A_{k+1} \cdot \prod^k A_i)^* \subseteq A_{k+1}^* \cdot (\prod^k A_i)^*$. Then, applying the induction hypothesis we have $A_{k+1}^* \cdot (\prod^k A_i)^* \subseteq A_{k+1}^* \cdot \prod^k A_i^* = \prod^{k+1} A_i^*$ ∎

Proposition 21 Let $A_i' = A_i \cdot R$, where $R(X) = \prod(S_{x_j'} A_j)$. Then

$$\prod A_i = \prod A_i'$$

Proof
$\prod A_i' = \prod(A_i \cdot R) = (\prod A_i) \cdot R = (\prod A_i) \cdot \prod(S_{x_i'} A_i) = \prod(A_i \cdot S_{x_i'} A_i) = \prod A_i$ ∎

Theorem 15 Consider a system of interacting machines $A_1, A_2, \ldots, A_n$. Let F be any simple CTL formula. Then

$$mc(\prod A_i, F) = mc(\prod \hat{A}_i, F)$$

Proof By Proposition 13, to prove the theorem we must show that $(\prod A_i)^* \subseteq \prod \hat{A}_i \subseteq \prod A_i$. This can be done by proving the following three steps:

(1)	$(\prod A_i)^* \subseteq \prod A_i^*$	(By Proposition 20)
(2)	$\prod A_i^* = \prod A_i^{*'}$	(By Proposition 21)
(3)	$\prod A_i^{*'} \subseteq \prod \hat{A}_i \subseteq \prod A_i$	

To prove step (3), recall that $\hat{A}$ is the *projection cofactor* of $A^{*'}$ with respect to $(\overline{A} + A^{*'})$, that is $\hat{A} = A^{*'}_{(\overline{A}+A^{*'})}$. From the general properties of the projection cofactor, for each A_i we have $A_i^{*'} \subseteq \hat{A}_i \subseteq A_i$. Combining the A_i machines, we have $\prod A_i^{*'} \subseteq \prod \hat{A}_i \subseteq \prod A_i$ ∎

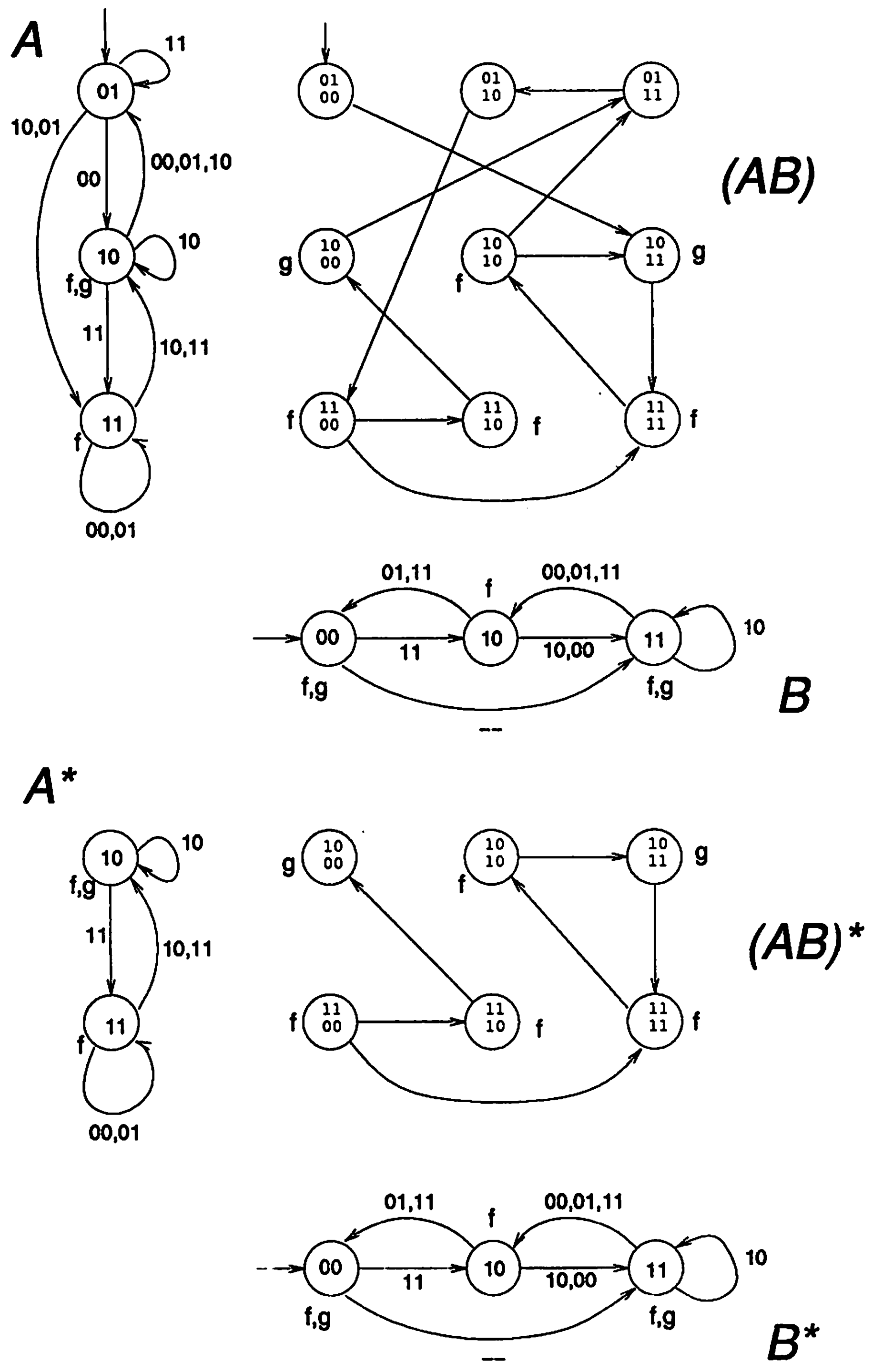

Figure 9: Two machine example