Tearing Based Automatic Abstraction for CTL Model Checking*

Woohyuk Lee Abelardo Pardo Jae-Young Jang Gary Hachtel Fabio Somenzi
University of Colorado
ECEN Campus Box 425
Boulder, CO, 80309

Abstract

In this paper we present the *tearing* paradigm as a way to automatically abstract behavior to obtain upper and lower bound approximations of a reactive system. We present algorithms that exploit the bounds to perform conservative ECTL and ACTL model checking. We also give an algorithm for false negative (or false positive) resolution for verification based on a theory of a lattice of approximations. We show that there exists a bipartition of the lattice set based on positive versus negative verification results. Our resolution methods are based on determining a pseudo-optimal shortest path from a given, possibly coarse but tractable approximation, to a nearest point on the contour separating one set of the bipartition from the other.

1 Introduction

We describe an approach for approximation-based formal verification of sequential systems. We obtain, automatically, upper (lower) bound approximations which allow conservative verification for ACTL (ECTL) formulas[1]. Thus, our tearing methods directly address the inherent computational difficulty which limits formal verification of practical circuit to those with on the order of a hundred latches or fewer. In contrast, the methods of [4] were able to perform redundancy removal on circuits with on the order of a thousand latches. In this paper we take first step toward establishing the practicality of applying such methods to problems in conservative CTL model checking.

In the sequel we will manipulate BDD representations of transition systems defined as $M = (S, X, T, S_0, O)$ where S is a set of states, X is a set of input combinations, T is an encoded partitioned transition relation $T(s, x, t) \subseteq S \times X \times S$, given in product form $T(s, x, t) = \prod_{i=1}^{m} T_i(s, x, \vec{t_i})$ and S_0 is a set of initial states. We operate in an encoded world, so $S = \{0,1\}^n$, $X = \{0,1\}^q$, $s \in S$ denotes an n-vector of state variable value assignments, and $\vec{t_i}$ represents a subvector of variable assignments. In a circuit context, the x_i can be called primary inputs, and the s_j latch variables. O is a similarly defined output domain.

Further, we assume that the block sub-relations T_i have disjoint next state variable support. That is, the $T_i(s, x, \vec{t_i})$ depend only on a local sub-vector $\vec{t_i}$, representing value assignments to only a subset of the next state variables t. The present state, primary input, and next state variable support of a function or relation such as T_i will be denoted as $\sigma_s(T_i)$, $\sigma_x(T_i)$, and $\sigma_t(T_i)$ respectively. We used the heuristic proposed in [4] to obtain the partitioned transition relation. The heuristic was based on a directed graph $G_L = (\{v_1, \ldots, v_n\}, E_L)$, which we call the *latch graph*, defined for the special case of the minimum partition ($m = n$). In this graph there is an edge from nodes v_i to v_j if and only if $s_j \in \sigma_s(T_i)$.

Definition 1.1 *Assume $s_j \in \sigma_s(T_i(s,x))$. When we replace the block $T_i(s,x)$ by its upper bound $T^{\lceil s_j \rceil}(s,x) = \exists_{s_j} T_i(s,x)$ or its lower bound $T^{\lfloor s_j \rfloor}(s,x) = \forall_{s_j} T_i(s,x)$, we say that we "tear" the connection from v_i to v_j. A circuit is "torn" by performing these operations on some subset of the intra-block connections determined by the partitioning step.*

Note tearing is directional, since T_j and T_k may still depend on s_2. Thus tearing is more fine-grained than previous approaches to decomposition [1, 6] which are based on "distributing the abstraction" ("en masse" tearing of all intra-block communication). The upper bound approximation is related to that of [4], but differs in that here the circuit is decomposed, rather than the state space. The lower bound approximation is related to that of [5], but differs in that here we do not use the "selection set" heuristic, and the abstraction is formed at the encoded (circuit with latches) level of abstraction, rather than at the abstract state space level.

We use two other tearing-like approximations as well. First, we use "block sub-setting" ($T \subseteq \prod_I T_i, \quad I \subseteq \{1, \ldots, m\}$) in the partitioned transition relation, as in [6, 1]. Second, we approximate state sets ($C(s)$) by both existential (upper bound: $C(s) \subseteq \exists_I(C(s))$) and universal (lower bound: $\forall_I(C(s)) \subseteq C(s)$) abstractions. The first abstraction is directly related to tearing connections in the physical system being verified. The second follows the spirit of tearing when dealing with the BDDs that arise in the verification process.

Although these approximations are not entirely new, we appear to be the first to combine them in approximation-based verification using systematic false negative/positive resolution techniques.

Figure 1 illustrates the effect of tearing on a simple two-block system, whose transition relation $T = T_1 T_2$ is defined by $T_1 = (t_1 \equiv (s_1 s_2))$ and $T_2 = (t_2 \equiv (s_1 + s_2))$. We tear the connection from of s_2 from the relation T_2 to the relation T_1, thus leading to $T_1^{\lceil s_2 \rceil} = \exists_{s_2} T_1 = s_1 + \bar{t}_1$, and $T_1^{\lfloor s_2 \rfloor} = (\forall_{s_2} T_1 = \bar{s}_1 \bar{t}_1)$. that connection has in the graph structure defined by the system. Note that when functional dependencies are eliminated by abstraction, we remain in

*This work is supported in part by NSF/DARPA grant MIP-94-22268 and SRC contract 95-DJ-560.

[1] ACTL (ECTL) formulas are CTL with only universal (existential) quantifiers and negation only at the atomic proposition level.

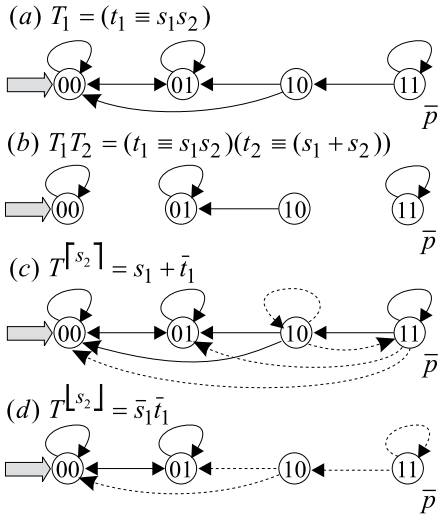

Figure 1: Upper and lower bound tearing.

(a)), T_1T_2 (Part (b)), $T_1^{\lceil s_2 \rceil}$ (Part (c)), and $T_1^{\lfloor s_2 \rfloor}$ (Part (d)). In Part (c), the four edges of $T_1^{\lceil s_2 \rceil}$ which are not in T_1, are shown solid, while other "pseudo-edges" are shown dashed. In Part (d) the right, the dashed edges which are in T_1, but not in $T_1^{\lfloor s_2 \rfloor}$ show what lower bound tearing eliminates.

Note that $T_1^{\lceil s_2 \rceil}$ is an upper bound approximation to $T = T_1 \cdot T_2$. However, $T_1^{\lfloor s_2 \rfloor} = (\forall_{s_2} T_1 = \overline{s}_1 \overline{t}_1)$ is *not* a lower bound approximation for $T = T_1 \cdot T_2$ (it has solid edges not present in T_1T_2). Nevertheless, $T_1^{\lfloor s_2 \rfloor} \cdot T_2$ is a lower bound approximation to $T = T_1 \cdot T_2$, and so is $T_1^{\lfloor s_2 \rfloor} \cdot T_2^{\lfloor s_1 \rfloor}$.

For the simple case of Figure 1, we can use the upper bound variable tearing approximation to efficiently demonstrate a false negative for $\mathbf{AG}p$. For an initial state set $I = \{00\}$, and the error set $\overline{p} = s_1 s_2$, there are no paths from I to $\overline{p}$ despite the pseudo-edges induced by the approximation. Dual cases exist where lower bound approximation may be similarly exploited.

The experimental results reported in this paper have been obtained using VIS [2] to which we have added the following modules: (a) a modified Partition module; (b) an Adaptive CTL Model Checker, based on Upper and Lower Bound Tearing Approximations; (d) Machine by Machine and Frame by Frame fixed point algorithms (as in [4]), modified for backward (as opposed to forward) traversal and lower (as well as upper) bounding.

We have added three novel contributions to the usual framework of conservative CTL model checking. First, we address the problem of false negative (or false positive) resolution by developing a theory of a lattice of approximations. When a false negative (or positive) is present, we show that there exists a bipartition of the lattice set based on positive versus negative verification results. Our resolution methods are based on determining a pseudo-optimal shortest path from a given, possibly coarse but tractable approximation to a nearest point on the contour separating one set of the bipar-

grain of tearing approximation than previously employed. To this end we propose a paradigm called "Tearing on Demand" in which we try to maintain, for as long as possible (to the point where BDD blow-up occurs) the usual exact model checking algorithm. We then resort to tearing approximations only when the size of the intermediate BDD heap is too large. Third, we show how the "Machine by Machine" and "Frame by Frame" traversal algorithms developed in [4] can be easily adapted to the more general verification framework model checking, as well as language containment.

2 Theory

Given a system M and its abstraction M_A, we need to establish a relation between the possible behavior of M and the behavior of M_A. We will use the concept of *simulation* as proposed in [8, 7] in the context of deterministic programs. We will show that tearing produces approximations that simulate the original system.

Definition 2.1 *Let us consider two systems M_1 and M_2 with the same output domain O. We say that a state s_1 is simulated by state s_2 ($s_1 \prec s_2$) if the output values of both states are equal and every successor of s_1 is simulated by a successor of s_2. We say that M_1 is simulated by M_2, denoted by $M_1 \prec M_2$, if for every path in M_1 there exists a path in M_2 such that the simulation relation holds state-wise.*

Given the transition sub-relation $T_i(s, x, \vec{t}_i)$, variable tearing consists of tearing any variable $v_j \in \sigma_s(T_i) \cup \sigma_t(T_i)$ creating a new conjugate variable $\tilde{v}_j$, called a Pseudo-Primary Input (PPI). The PPI $\tilde{v}_j$ is a "free" copy of v_j. If this PPI is then removed by existential abstraction, we get $T_i^{\lceil s_j \rceil}$, whose STG has the same states as T_i, but a superset of its edges. If we similarly replace T_i by $T_i^{\lfloor s_j \rfloor}$, the new STG will have a subset of the original edges. By the definition, there exists a simulation relation such that $T \prec T^{\lceil s_j \rceil}$, and $T^{\lfloor s_j \rfloor} \prec T$.

Suppose we apply upper bound tearing on variables $v_1, \ldots, v_k$, and the corresponding torn systems are denoted $T^1, \ldots, T^k$. Then, each torn system is an upper bound approximation of the previous, so $T \prec T^1 \prec \ldots, T^{k-1} \prec T^k$. Symmetric reasoning applies to the universal abstraction. In either case the final torn system is independent of the order of tearing. Therefore, the set of upper and lower bound tearing approximations constitutes a lattice. This is shown for $k = 3$ in Figure 2, where we have added a third subsystem $T_3 = (t_3 \equiv s_1 s_2 s_3)$. Here $T^{\lceil 123 \rceil}$ represents the upper bound obtained by all 3 of the connections indicated by arrows at the top of the figure. Similarly, $T^{\lceil 12 \rceil}$ is the result of tearing only the first two connections, and so on. A strategy for the refinement of a tearing abstraction may be formulated on such a lattice. Suppose our tearing based model checker is asked to check the formula $\mathbf{AG}p$. In the present case, we Initially tear three variables and start with the maximal approximation $T^{\lceil 123 \rceil}$. If a FALSE answer is returned, since the upper bound approximation only gives true positives, this putative negative result needs to be confirmed. The idea is to start with the maximum upper bound $T^{\lceil 123 \rceil}$ and pseudo-optimally search for the shortest path to the contour

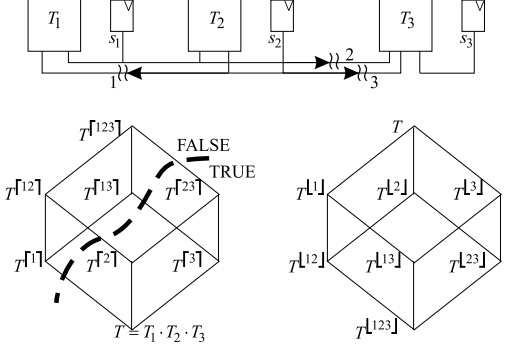

Figure 2: Lattice of upper and lower approximations.

separating TRUE and FALSE.

The original block may be reconstructed from the "torn" one by "stitching", the dual of tearing. If the PPIs were eliminated by abstraction, this is done by replacing the approximated subrelations by the originals. Else we may stitch by simply composing the PPIs with their conjugates: $T_i \leftarrow \exists_{\tilde{v}_j} \widetilde{T}(s, x, \vec{t_i}, \tilde{v}_j) \cdot (\tilde{v}_j \equiv v_j)$. Here the term $(\tilde{v}_j \equiv v_j)$ is called the "stitching constraint".

Two separate approximation mechanisms come under the aegis of "block tearing". First, block tearing includes block sub-setting — the practice of including some blocks and ignoring others. This was studied by Long [6]. Second, block tearing includes tearing all connections from block i to block j. Both of these aspect were studied by Balarin et al. in [1]. Since we also use both, we review them briefly below.

Block sub-setting rests on the following observation. Let $C \subseteq \{1, \ldots, n\}$, and let T_C be the transition relation obtained by ignoring blocks T_j, $j \notin C$. That is, $T_C = \Pi_{i \in C} T_i(s, x, \vec{t_i})$.

Since $T \subseteq T_C$ we have $M \prec M_C$, that is, M_C simulates M and each such approximation is defined by which of the 2^n possible subsets C is chosen, we have by Stone's theorem that the set of approximations forms a lattice, with the tautologous transition relation as the maximal element and the original transition relation as the minimal element.

Group tearing consists of tearing all the connections from one block of the partitioned transition relation to another. Note that tearing of this type may be unidirectional or bidirectional. The bidirectional type of tearing abstracts completely the connections from block i to j and vice-versa. This kind of tearing allows the existential quantification in the image and preimage computation to distribute over the abstracted blocks. In the case of the upper approximation:

$$\exists_s(C(s) \cdot \exists_x \textstyle\prod_i T_i(s, x, t_i)) \subseteq \prod_i(\exists_s C(s) \cdot \exists_x T_i(s, x, t_i)),$$
$$\exists_t(C(t) \cdot \exists_x \textstyle\prod_i T_i(s, x, t_i)) \subseteq \prod_i(\exists_t C(t) \cdot \exists_x T_i(s, x, t_i)).$$

The first identity was used in [1]. The second is just the backward version of the same thing.

Similar types of approximations were used by Burch to simplify the BDD representation of multipliers [3], and by Cho, et al [4] to decompose the reachable state set of a sequential circuit into a more compact Cartesian product form.

a hierarchy of simulation relations, we now state the relationship between simulation relations and conservative CTL model checking. In the following ACTL and ECTL refer to the CTL fragment restricted to universal and existential quantification respectively and negation at the level of atomic predicates [6].

Theorem 2.1 *Let M be a system with relation T and M^U obtained by upper bound tearing, so that $M \prec M^U$. Then $\forall f \in ACTL$, $M^U \models f \Rightarrow M \models f$. Dually, if M^L is obtained by lower bound tearing, so that $M^L \prec M$, then $\forall f \in ECTL$, $M^L \models f \Rightarrow M \models f$.*

In addition to approximating transition relation, we may approximate the state sets computed during verification. Model checking procedures rely on fixed point computations, in which we compute the preimage of a given constraining set $C(t)$. We next consider the approximation of the set $C(t)$, which will be based on the observation that the preimage function is monotonic. Therefore, if we overestimate the restricting set $C(t)$ we will obtain an over approximation of its preimage. This type of approximation is orthogonal to block and variable tearing and therefore we may choose to use only one of them or both at the same stage of the computation.

3 Conservative Verification

We have discussed above methods based on approximating the transition relation. Next we consider a way to obtain lower bound of exact preimage by using universal abstraction of the constraint set $C(t)$. We first decompose the constraint set $C^0(\vec{t})$ into a disjunctive list $\sum_i C_i^0(\vec{t_i})$. This is done by universal abstraction of variables $\vec{t_j}$ where $i \neq j$. Since we have performed the universal abstraction of $C^0(t)$, we are guaranteed to have $\sum_i C_i^0(\vec{t_i}) \subseteq C^0(\vec{t})$.

The next step is to compute the preimage of each C_i^0 with the *exact* transition relation. Since C_i^0 only depends on $\vec{t_i}$ and by definition, T_j depends on $\vec{t_j}$, and not on $\vec{t_i}$, $i \neq j$, we can distribute existential abstraction while computing the preimage of each C_i^0. We keep the resulting set $C_i^1(\vec{s})$ in the form of conjunctive list. When we apply this lower bound preimage computation to fixed point computation, we universally abstract the variables s_j where $j \neq i$ from the resulting set $C_i^1(\vec{s})$ which is now in the form of conjunctive list. Notice that, since the universal abstraction distributes over conjunction, we never compute the entire transition relation.

We now consider the fixed point computation using the upper and lower bound preimage computation described in previous section. We refer to the set of states that satisfies a given formula as the satisfying set. If we use the upper bound preimage computation to conservatively check ACTL formulas, we obtain a lower bound of the exact satisfying set. Similarly, if we use the lower bound preimage computation to model check ACTL formulas, the resulting approximate satisfying set will be a superset of the exact satisfying set. The ensuing refinement step may be viewed as tightening the upper/lower bounds of the satisfying set. We will assume that the partitioned transition relation has been previously computed. Algorithm APPROX_MC_UPPER of Figure 3 verifies a

```
procedure APPROX_MC_UPPER(T^1, ..., T^m, I(s), p(t)){
1    T^opt = 1; J = {1, ..., m}; I_v(s) = I(s)
2    while( ¬ (Mem out) ∧ ¬ (Time out) )
3        foreach ( j ∈ J ) {
4            T^cur = T^opt · T^j
5            F^j(s) = EVAL(T^cur, p);
6            if((¬F^j(s) ∩ I_v(s)) = ∅) return(TRUE)
7            I_v(s) = I_v(s) ∩ ¬F^j(s);
8            k* = SMALLEST{|(¬F^j(s) ∩ I(s))|}
         }
9        J = J − {k*}; T^opt = T^opt · T^k*
10       I(s) = I_v(s)
     }
}
```

Figure 3: Algorithm to prove a true positive of an ACTL formula using upper bound approximation.

true positive of a given ACTL formula using the upper bound approximation. The first step is to evaluate the formula with every single subsystem in a given order. The EVAL() procedure in line 5 is identical to the evaluation procedure of conventional model checking algorithms. The resulting set $F^j(\vec{s})$ is a lower bound of the exact satisfying set. For each $F^j(\vec{s})$, we check whether the formula evaluates to TRUE (line 6). Notice that the algorithm uses a monotonically decreasing subset I_v for these checks. Each time we compute $F^j(\vec{s})$, we reduce the cardinality of the current $I_v(\vec{s})$ by intersecting it with $F^j(\vec{s})$ as shown in line 7. The algorithm succeeds if it occurs that $I_v(\vec{s})$ is contained in the lower bound of the exact satisfying set. In this case, the algorithm exits and returns TRUE. When the algorithm fails to prove true positive with a single subsystem, we update the best subsystem that produces the smallest $|\neg F^j(\vec{s}) \cap I(\vec{s})|$ and proceed by refining the transition relation. For each refinement we check again whether the formula evaluates to TRUE. The algorithm iterates this process until it runs out of memory or runs over certain time bound.

If this upper bound algorithm fails to prove the true positive, we abandon this strategy and try to prove the true negative using the lower bound algorithm of Figure 4. Unlike the upper bound algorithm, we first decompose the constraint set $p(\vec{t})$ into a disjunctive list of $\sum_j p_j(t_j)$ as shown in line 4 (as discussed above). If there exists any initial state that is not contained in the upper bound of the exact satisfying set, we know that the original ACTL formula is false (lines 9,10). In this lower bound algorithm we have shown the additional refinement step which is similar to the "Machine by Machine" method applied to forward reachability analysis in [4]. By $FANOUT(T_j)$ we denote those subsystems whose transition relation depends on the present state variable s_j. By $FANIN(T_j)$ we denote those subsystems whose present state variable is a member of the support of T_j. Every time we encounter a smaller satisfying set F^j, we re-schedule the subsystems that are fanouts of T_j. These systems are evaluated once again with the smaller care states, $F_{FANIN(T^j)}$ which is the conjunction of satisfying sets of components until there is no new scheduled subsystem.

```
procedure APPROX_MC_LOWER(T^1, ..., T^m, I(s), p(t)){
     /* F = satisfying set I = initial states */
1    T^opt = 1; J = {1, ..., m};
2    foreach(j ∈ J) {
         schedule_j = 1; F^j_old = 1;
4        q_j = support(T_j) − {s_j, t_j}; p_j(t_j) = ∀q_j p(t);
     }
5    while( ¬ (Mem out) ∧ ¬ (Time out))
6        while ( ∃_i ( schedule_i = 1 ) ) {
7            foreach ( j ∈ J ) {
8                T^cur = T^opt · T^j;
9                F^j(s_j) = EVAL_LOWERBOUND(T, T^cur, p_j(t_j),
                                    q_j, F_FANIN(T^j));
10               if ((¬F^j ∩ I) ≠ ∅) return(FALSE)
11               if (F^j ⊂ F^j_old) }
12                   foreach (z ∈ FANOUT(T^j)) {
                         schedule_z = 1 } }
13               schedule_j = 0; F^j_old = F^j
14               k* = SMALLEST{|F^j|} } }
15       J = J − {k*}; T^opt = T^opt · T^k* }
}
```

Figure 4: Algorithm to prove a true negative of an ACTL formula using lower bound approximation.

4 Creating Approximations on demand

In this section we describe the approach in which the introduction of an approximation is delayed until there is a constraint in the model checking procedure. We recall from Section 2 the computation of the preimage of a restricting set $C(t)$. The technique presented in this section detects when intermediate preimage BDDs become too large. In this case the intermediate result is approximated.

The relational product is computation by the following iteration. Let us assume we have n sub-relations, and that the order of quantification is given. Let the initial restricting set be $C(t)$, and q_i represent the set of variables that are quantified in the i^{th} iterate.

$$\begin{aligned} PreImage_0(C(t)) &= C(t) \\ PreImage_{i+1}(C(t)) &= \exists q_i \, (T_i(s,t) \cdot PreImage_i(C(t))) \\ PreImage(C(t)) &= PreImage_n(C(t)) \end{aligned}$$

Suppose that both $C(t)$ and $PreImage(C(t))$ may have a compact representation in terms of BDDs. The problem with the above approach is that even in this favorable case, the intermediate results produced by $PreImage_i(C(t))$ may require excessive memory. When this occurs, certain variables in the support of the intermediate set have yet to be quantified. In order to reduce the size of the representation, our algorithm will explore in advance the sets q_{i+1} through q_n. It then pre-quantifies (ahead of schedule) these variables one by one, until the size of the representation has been reduced by

if an lower bound approximation is required, else existential quantification is used.

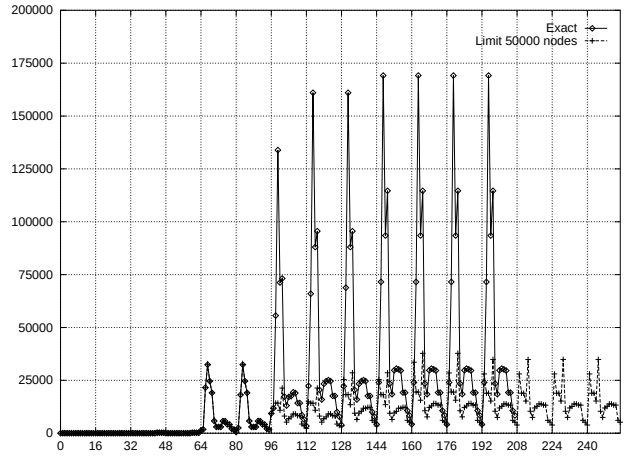

Figure 5: Bdd size during fix point computation.

Figure 5 shows the size of the intermediate results during a fixed point computation with the ISCAS'89 circuit s1423. The vertical divisions in the horizontal axis mark the end of every preimage computation. In the case of the exact method, the verification of this formula required 13 preimage computation, at the end, the formula was proved true but the memory requirements reached 175000 BDD nodes. The approximate result instead remains below the 50000 limit and still obtains the exact result. The approximation of temporary results also creates the need for more iterations until the fixed point is reached, in this particular case, this number is 16, but since the operands have a more compact representation, the execution time was 3120 CPU seconds for the exact case and 408 seconds for the approximate case (the machine configuration is explained in Section 5).

5 Experimental Results

Experiments were run on a Pentium Pro(200MHz) system with 256K internal L2 cache and 128M of main memory. Our approximation based model checker was built on top of VIS release version 1.1 [2], which was used to perform the corresponding exact computations where possible. Our experimental results are given in Table 1.

Figure 6 (a) shows how the upper bound on the reachable states of the bdlc circuit changes as we refine the approximate system. This circuit has $2.85 \cdot 10^{45}$ reachable states. After initial partitioning, each subsystem contains about ten latches. An upper bound approximation obtained with a subset of just six subsystems gives a fairly tight upper bound of just $4.05 \cdot 10^{45}$ states. Figure 6 (b), for the CTL formula $AX(t_0 = 1)$ (here t_0 is an arbitrary next state variable) shows how the upper and lower bounds of the satisfying set changes as we refine the approximation of the bdlc transition relation. Note the discontinuity as the number of subsystems passes from seven to eight. As we perform the seventh refinement, the cardinality of the satisfying set obtained by upper bound with the next subsystem, we observed another small leap of two order of magnitude. Such discontinuities are typical, and reflect sensitivity of the system to granular approximation.

In our benchmark set, latch counts ranged from 40 to 597. Circuit s15850, one of the ISCAS'89 benchmark circuits, has 14 primary inputs, and 87 primary outputs. For the first experiment on this circuit, we formulated an arbitrary ACTL formula $AG(p)$ where p is an arbitrary set of states, whose characteristic function depends on about 40% of the next state variables. The result is given in the first row of the s15850 subtable. The lower bound algorithm proved the true negative with a single subsystem, which contained 10 latches. The second experiment (second row of subtable) was similar to the first, but the support of p was decreased to about 20% of the next state variables. The third experiment is identical to the first, except that the formula is changed to $AF(p)$. The last experiment with this circuit verified the formula $AG(TRUE)$. As indicated by these results, for circuits like s15850, it is easy to find formulas which the exact method fails but the approximate method succeeds. We found some situations in which both exact and approximate methods proved the given formula, like the second case presented in Table 1, where a lower bound was used. However, in all cases, our approximate method was significantly faster and required significantly less memory. We did not find any case in which the exact method succeeded while our approximate method failed.

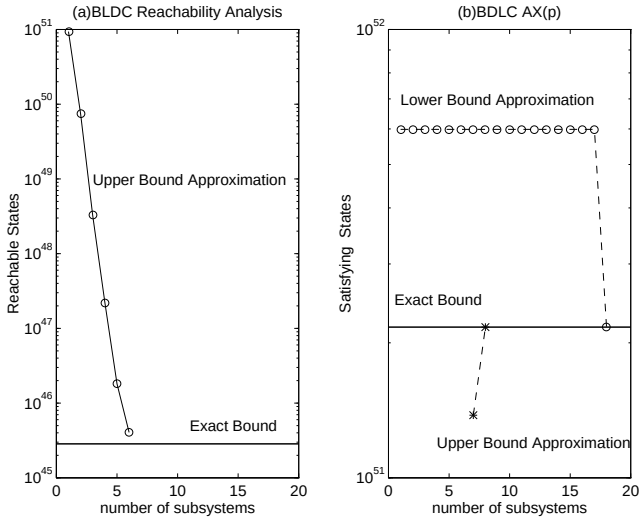

Figure 6: Evolution of upper and lower bounds

Next we present the experiments with the bdlc circuit. The formula we used for the first example is $AG(p)$ where p is an arbitrary set of states with all next state variables in its support. The exact method exceeded the time bound. However, the lower bound algorithm proved the true negative with seven subsystems. The approximation obtained with these seven subsystems contains 64 latches.

The formula for the second example is $AF(I(s))$. Notice that in this case, with a single subsystem, the upper bound algorithm came to within a factor of 30 of the exact satisfying

	Partition		Formula		Time		Max BDD Size		| Satisfying States |	
	FFs	Subsystems	Exact	Approx	Exact	Approx	Exact	Approx	Exact	Approx
s15850	597	60	?	F at 1	>18000	599	>8430	1	?	0
			F	F at 1	1349	247	1703	1	0	0
			?	T at 1	>18000	103	>10092	717	?	3.59e179
			?	T at 1	>18000	48	?	1	?	5.18e179
bdlc	172	18	?	F at 7	>18000	719	>91578	16	?	5.74e51
			T	T at 1	32	2	51	22	8.56e46	2.85e45
			T	T at 1	4047	2	106763	1	5.98e51	5.98e51
s1423	74	8	T	T at 1	170	1	3617	3	1.63e22	1.41e22
abp	40	10	T	T at 4	1	2	21	6	69480	69120
			F	F at 9	1	6	54	49	1158	29268

Table 1: Tearing results

set. The third example is again $AG(TRUE)$, and this time a single subsystem comes to within 1%.

The next circuit we used is the ISCAS'89 benchmark circuit s1423. Although it has only 74 latches, this circuit is not traversable and it is known to be a "difficult" circuit. The formula we used is $AF((G726{=}0 + G729{=}1))$. Again this experiment shows how closely the upper bound to the satisfying set approximates the exact satisfying set when just a single subsystem (around 9 latches) is used.

The last circuit contains 40 latches. It is a generic arbiter controller included in the VIS distribution. Comparison with the exact computation was made with two complex ACTL formulas. With only four subsystems, the approximate method came very close to the exact satisfying set, and proved the formula TRUE. For the second formula, nine subsystems were needed to prove the formula FALSE with the lower bound algorithm.

6 Conclusions

We have presented a *tearing* paradigm for conservative model checking of ECTL and ACTL formulae. We addressed the problem of false negative (or false positive) resolution for model checking on a selection of medium to large examples, which were impossible for traditional exact methods. Although we were successful, we desire to focus future effort on still larger examples and more realistic CTL formulae.

We also presented some promising but preliminary results on a separate paradigm, which we called "Approximation on Demand". These techniques aim to avoid the memory requirements in the relational product computation, and they provide a finer control on the bounds produced by the approximation in the system.

We presented a theory of a lattice of approximations, which arises in variable tearing and block tearing. Our resolution methods are based on determining a locally-optimal shortest path from a given, possibly coarse but tractable approximation to a nearest point on the contour separating one set of the bipartition from the other.

Experimental results showed that in many cases, the exact method could not verify the given formula while the approximate method succeeded with a relatively economical approximation, based on a small number of subsystems.

Acknowledgments: We acknowledge help from Rajeev Ranjan, Tom Shiple, Gitanjali Swamy, and Robert Brayton.

References

[1] F. Balarin and A. L. Sangiovanni-Vincentelli. An iterative approach to language containment. In C. Courcoubetis, editor, *Fifth Conference on Computer Aided Verification (CAV '93)*. Springer-Verlag, Berlin, 1993. LNCS 697.

[2] R. K. Brayton et al. VIS: A system for verification and synthesis. In T. Henzinger and R. Alur, editors, *Eigth Conference on Computer Aided Verification (CAV'96)*, pages 428–432. Springer-Verlag, Rutgers University, 1996. LNCS 1102.

[3] J. R. Burch. Using BDDs to verify multipliers. In *Proceedings of the Design Automation Conference*, pages 408–412, San Francisco, CA, June 1991.

[4] H. Cho and F. Somenzi. Sequential logic optimization based on state space decomposition. In *Proceedings of the European Conference on Design Automation*, pages 200–204, Paris, France, February 1993.

[5] P. Kelb, D. Dams, and R. Gerth. Practical symbolic model checking of the full μ-calculus using compositional abstractions. Technical Report 95-31, Department of Computing Science, Eindhoven University of Technology, 1995.

[6] D. E. Long, A. Browne, E. M. Clarke, S. Jha, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. In D. L. Dill, editor, *Sixth Conference on Computer Aided Verification (CAV'94)*, pages 338–350. Springer-Verlag, Berlin, 1994. LNCS 818.

[7] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, MA, 1994.

[8] R. Milner. An algebraic definition of simulation between programs. *Proc. 2nd Int. Joint Conf. on Artificial Intelligence*, pages 481–489, 1971.