Iterative Abstraction-based CTL Model Checking

Jae-Young Jang
Motorola Inc.,
7700 W. Parmer Lane
Austin, TX 78729
Jae.Jang@motorola.com

In-Ho Moon   Gary D. Hachtel
University of Colorado at Boulder
ECEN Campus Box 425
Boulder, CO 80309
{In.Moon,Gary.Hachtel}@colorado.edu

Abstract

A paradigm for automatic approximation/refinement in conservative ***CTL*** *model checking is presented. The approximations are used to verify a given formula conservatively by computing upper and lower bounds to the set of satisfying states at each sub-formula. These approximations attempt to perform conservative verification with the least possible number of BDD variables and BDD nodes. We present new forms of operational graphs to avoid limitations associated with previously used operational graphs. Three new techniques for efficient automatic refinement of approximate system are presented. These methods make it easier to find the locality. We also present a new type of don't cares (Approximate Satisfying Don't Cares) that can make model checking more efficient in time and space. On average, an order of magnitude speedup was achieved.*

1. Introduction

The success of formal verification in detecting incorrect designs has been proven over the last decade. However, the limitation on the size of verifiable problems continue to be a serious drawback. Therefore, for most practical, industrial strength designs, abstraction, either manual or automatic, is required to make formal verification tractable. There has been extensive research on this topic. Some of works that were foundation of this research are summarized below.

In [6], Kurshan described an abstraction paradigm called "localization reduction" in the context of ω-regular language containment based on reducing the parts of a system that are *irrelevant* with respect to the property being verified. Another approach using both (but not at the same time) over and under approximations was considered in [7] based on *State Space Decomposition* [3]. If the result is not conclusive due to abstraction, further refinement is done by adding the best subsystem to each of the remaining subsystems. In the worst case, the number of decision procedure calls is quadratic in the number of subsystems in the system. This procedure is limited to ∀**CTL** formulae. In [12], a more sophisticated, operation-based refinement was proposed. The given CTL formula is parsed, and the initial error states (initial states shown not to satisfy the given CTL formula) are treated as a "goal set." Using the parse tree of the formula, called an operational graph, the goal set is systematically distributed for refinement to the operations which induced the approximation in the first place. This procedure requires exact image computation to propagate goal set. Another new method that uses upper- and lower-approximations at the same time was introduced in [8]. But, they did not explore all the benefits from previously computed satisfying states (explained in Section 5) and their refinement method is only based on dependency relation with structural depth.

2. Approximation

The physical interpretation of the given system is a finite state machine (FSM) model. Here the symbol X represents the external input to the system. We operate in an encoded world, so $S = \{0,1\}^n$ and $X = \{0,1\}^q$, and $s \in S$ denotes a n-vector of state variable value assignments. To distinguish a present state $s \in S$ from a next state $s \in S$ t is used to express a state after one clock cycle from a state s. The triple s,x,t can be viewed as a member of the system transition relation T, or equivalently as a labeled edge of the *State Transition Graph*. Thus there are n latches, and $T = \prod_{i=1}^{n} T_i(s,x,t_i)$ is the system transition relation. This transition relation can be implemented in many ways. In this research, BDDs are used to manipulate a set and a relation including $T(s,x,t_i)$. In general, BDDs operations are exponential in the number of variables and polynomial in the size of BDDs nodes.

Model checking of a **CTL** [4] property f is based on a states evaluation function, $\mathbf{SAT}(f)$ —all states in $\mathbf{SAT}(f)$ satisfy f [10]. After computation of $\mathbf{SAT}(f)$, the initial states, S_0, are compared to $\mathbf{SAT}(f)$ and verification is completed with the following test:

$$S_0 \subseteq \mathbf{SAT}(f) \Leftrightarrow f is true. \quad (1)$$

Many researchers have focused on approximate evaluation of the set of satisfying states. We shall refer to two such approximate evaluating functions: $\mathbf{SAT}^{-}(f)$, and $\mathbf{SAT}^{+}(f)$:

$$\mathbf{SAT}^{-}(f) \subseteq \mathbf{SAT}(f) \subseteq \mathbf{SAT}^{+}(f).$$

When we derive exact verification results from these approximate states evaluations, we call it *Conservative Verification* (*Positive Conservatism*: $S_0 \subseteq \mathbf{SAT}^{-}(f) \Rightarrow S_0 \subseteq \mathbf{SAT}(f)$, *Negative Conservatism*: $S_0 \not\subseteq \mathbf{SAT}^{+}(f) \Rightarrow S_0 \not\subseteq \mathbf{SAT}(f)$).

All $\mathbf{SAT}(f)$ computations can be based on one primitive operation called $\mathbf{PRE}(C(s))$ which computes predecessors of states in $C(s)$:

$$\mathbf{PRE}(C(s)) = \exists_{x,t}[\prod_{i\in I} T(s,x,t_i)\cdot C(t)]. \quad (2)$$

Same index $i \in I$ are associated with s_i, t_i, and $T_i(s,x,t_i)$ and T_i is used instead of $T_i(s,x,t_i)$ interchangeably. Of the many ways to implement upper-bound or lower-bound of Equation 2, we shall consider only the following methods defined in Definition 2.1 and 2.2, where $T_i^{-} \subseteq T_i \subseteq T_i^{+}$. With these methods, lower-bound and upper-bound preimage computations are defined as shown below. Here, $C^*(t)$ is computed from $C(t)$ according to the approximation method being used.

$$\begin{aligned}\mathbf{PRE}^{+}(C(t)) &= \exists_{x,t}[T^{+}(s,x,t)\cdot C^*(t)],\\ \mathbf{PRE}^{-}(C(t)) &= \exists_{x,t}[T^{-}(s,x,t)\cdot C^*(t)].\end{aligned}$$

Definition 2.1 (Upper-bound Transition Relation)

$$T^{+}(s,x,t) = \prod_{i\in I^A} T_i(s,x,t_i)\cdot \prod_{i\in I^R} T_i^{+}(s,x,t_i) \quad (3)$$

Here, $I^A \subseteq I, I^R = I - I^A$.

(U1) $T_i^{+} = \mathbf{BddSupersetting}(T_i)$, $C^*(t) = C(t)$.

(U2) $T_i^{+} = 1$, $C^*(t) = \exists_{t^R} C(t)$, $t^R = \{t_i \mid i \in I^R\}$.

U1 refers to making T_i^{+} from T_i with generic BDDs supersetting techniques explained in [13]. These methods produce a new set that is close to the original set in terms of on-set minterm count, where the new BDDs should be much smaller in size. **U2** refers to the *Block Tearing Method* [1, 9, 7], where T_i^{+} is the tautology. All t^R variables are not appeared during **PRE** computation.

Definition 2.2 (Lower-bound Transition Relation)

$$T^{-}(s,x,t) = \prod_{i\in I^A} T_i(s,x,t_i)\cdot \prod_{i\in I^R} T_i^{-}(s,x,t_i) \quad (4)$$

(L1) $T_i^{-} = \mathbf{BddSubsetting}(T_i)$, $C^*(t) = C(t)$.

(L2) $T_i^{-} = 1$, $C^*(t) = \forall_{t^R} C(t)$, $t^A = \{t_i \mid i \in I^A\}$.

The **L1** method is same as **U1** but the complement of the given BDDs are supersetted. The **L2** method was used in [7] to disprove a $\forall$**CTL** formula more efficiently.

All approximation methods explained in this section are easier than exact one in a sense that either the number of variables is reduced or the size of BDDs are reduced during **PRE** computation.

3. New Operational Graphs

The notion of an *Labeled Operational Graph* was first defined in [12]. There a *Polarity* was associated with each node in operational graph. The polarity of a node of the operation graph tells whether over- (+) or under-approximation (-) should be computed at that node in order to have a positive conservative answer at the top node. However, it considered only the case of '-' polarity at the top node, therefore, only *Positive Conservatism* was considered in that work. We now define a *Dioecious Operational Graph* (**DOG**), which allows either positive or negative polarity at the top node. We shall refer to a graph with a required '+'('-') polarity at the top as a *Positive (Negative) Dioecious Operational Graph,* **pDOG** (**nDOG**) respectably. Then *the Labeled Operational Graph* in [12] is same as **nDOG**. Note that a model checker requires a **pDOG** in order to deal with *Negative Conservatism.*

Since **pDOG** or **nDOG** has a one-side advantage (e.g., f with **nDOG** is good to conservatively verify f TRUE, but not for verifying f FALSE), we should exploit the advantage of both cases. If **nDOG** and **pDOG** are merged, a more effective approximate verification approach can be achieved without causing too much overhead. This operational graph shall be called the *Monoecious Operational Graph* (**MOG**). Every node v in **MOG** has $\mathbf{SAT}^{+}(v)$ and $\mathbf{SAT}^{-}(v)$ at the same time. With these two approximate satisfying states, dual conservatism can be applied on a unified labeled operational graph. The other benefit of **MOG** will be explained in Section 5. In Figure 1, three different operational graphs of $\mathbf{AG}(\mathbf{EF}q)$ are illustrated. First, formula f should be translated into $\exists$**CTL** form by using equivalence rules explained in [10]. For example, $\mathbf{AG}p = \neg\mathbf{EU}(True, \neg p)$ and $\mathbf{EF}q = \mathbf{EU}(True, q)$. The polarity of a given node v in **DOG** is determined by counting the number of $\neg$ nodes in a path from the top node to v. When even number of $\neg$ nodes in this path, node v gets the same polarity as the top node. $\mathbf{SAT}^{+}(v)$ or/and $\mathbf{SAT}^{-}(v)$ is computed according to a polarity of v in a operational graph in depth first order. It is guaranteed that an approximate satisfying set at the top node is correct by using this method [12].

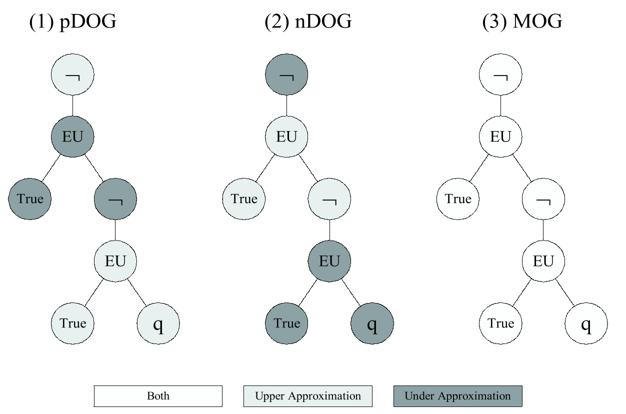

Figure 1. Different Operational Graphs for $\mathbf{AG}(\mathbf{EF}q)$

4. Iterative Model Checking and Refinement

In approximate **CTL** based model checking, the main operations are $\mathbf{SAT}^{+}(f)$ and $\mathbf{SAT}^{-}(f)$ computations. We compute $\mathbf{SAT}_0^{+}(f)$ or $\mathbf{SAT}_0^{-}(f)$ on the operational graph that we want to use first. If no conservative answer is found, $\mathbf{SAT}_1^{+}(f)$ or $\mathbf{SAT}_1^{-}(f)$ is computed with a refined system, where $\mathbf{SAT}_0^{+}(f) \supseteq \mathbf{SAT}_1^{+}(f)$ and $\mathbf{SAT}_0^{-}(f) \subseteq \mathbf{SAT}_1^{-}(f)$. This process is continued until we prove the given formula or run out of gas. We call this type of verification *Iterative Model Checking* and the index i in $\mathbf{SAT}_i^{+}(f)$ is called *iteration index*. As the size of a circuit grows, it is more difficult to reason about it as a whole. So, most properties that we want to verify are written in a modular manner. In other words, a formula may require only part of a system to be verified by model checking. This is called *Locality*. In this sense, refinement is important because the main goal of refinement is how to find out the locality of a formula with less effort.

In Equations 3 and 4, the index set I is divided into two sets, I^A and I^R. Then a transition sub-relation T_i for a state variable t_i whose index i is in I^R is approximated to T_i^{+} or T_i^{-}. The refinement is simply to replace T_i^{+} or T_i^{-} with T_i. Then, our problems are how to define I^A and how to increase I^A by adding I^{Δ}, a subset of I^R. First, the constant *bound* is the size of I^{Δ} and defined by the user. The first index set I_0^{Δ} is computed as following. Let I_d be a set of indices of latches shown in a given formula. If $\mid I_d \mid \leq bound$, $I_A = I_0^{\Delta} = I_d$. Otherwise *bound* indices are chosen out of I_d randomly. If the verification result is inconclusive, $I^A = I^A + I_1^{\Delta}$ and so on. To make refinement efficient, I_i^{Δ} should be computed in a way that I_i^{Δ} is more important to verify a given formula than any other combination of *bound* elements in I^R. To compute I_i^{Δ} for $i > 0$, the following two methods are used.

4.1. Latch Affinity Refinement

In [3], various relations between two latches were defined and used to partition the state space. Those relations are used here to compute efficient schedule to refine the approximate system. Those latch relations include connectivity, agreement, and affinity. The connectivity is computed based on whether a next state function depends on the other state variable. When two state variables have mutual dependency, their connectivity is 1. In case of uni-directional dependency, connectivity is 0.5 and 0 when there is no dependency. The aggrement shows how similar a next state function is to the other and the complement of the other. Minterm fraction of Exclusive OR of these two next state functions shows the similarity.Finally, the affinity is a convex combination of these two factors. Latch affinity is the factor that shows how two state variables are structurally dependent and how their behaviors are similar. Let σ_{ij} be the affinity between s_i and s_j. After computing I_0^{Δ}, the relational affinity σ_j^* for each $j \in I - I_0^{\Delta}$ is computed as below.

$$\sigma_j^* = \sum_{i \in I_0^{\Delta}} \sigma_{ij} \tag{5}$$

This new σ_j^* is nothing but a summation of affinities between s_i in I_0^{Δ} and s_j in I^R. If σ_j^* is bigger than σ_k^*, it implies s_j has stronger affinity to the state variables in the formula than s_k. According to the relational affinity, we compute each I_i^{Δ}. This method is based on direct relations to the given formula.

4.2. Edge Traversal Refinement

Upper-bound approximation creates pseudo edges which are not in the exact system. Main idea of this algorithm is to determine which transition sub-relation has more impact to delete them. Not every pseudo edges are considered, but the pseudo edges that are traversed from the initial states are targeted. First, a subset of initial states which have not been proved to satisfy the formula are propagated by pre-order depth first search in operational graph. This propagation, called *Forward Edge Traversal* is a series of upper-bound image computations bounded by $\mathbf{SAT}^{+}$ computed already. Each approximate image computation yields an edge envelope $ee(s,x,t)$ which is a set of exact and pseudo edges. This edge envelope is conjoined with the negation of each transition sub-relation which is approximated during verification. This conjoined term is $ee(s,x,t) \cdot \overline{T_i(s,x,t_i)}$. The result is stored as $Kill_i(s,x,t)$ and this is the set of edges to be removed if the approximated $T_i^{+}(s,x,t)$ is replaced with exact $T_i(s,x,t)$. If any of $Kill_i(s,x,t)$ is not empty during this propagation, the system is refined in a way that the transition sub-relation with the biggest minterm count of $\mid Kill_j(s,x,t) \mid$ is refined. When no pseudo transition is found and the traverse reaches to bottom of the formula, then post-order propagation of refining procedure, called resolution propagation, where an un-verified set of states of a sub-formula during edge traversal is bi-partitioned into a set which satisfies the sub-formula and the other set where sub-formula is verified false. This second propagation verified the given formula. This refinement method is only applicable when all nodes $v = \{\mathbf{EX}g, \mathbf{EG}g, \mathbf{EU}(g,h)\}$ in operational graph have $\mathbf{SAT}^+(v)$, because the edge envelope $ee(s,x,t)$ is computed in $\mathbf{SAT}^+(v)$. The advantages of this method are (1) refinement is guided by approximate satisfying states and (2) the impact can be predicted before the system is actually refined.

4.3. Example

Now, a simple example is presented to illustrate the algorithms described in this section. The system models a 4-bit modulo 10 counter. States are encoded with variables $counter[3:0]$. The initial state is State 7 meaning '$counter[3:0] = 0111$'.

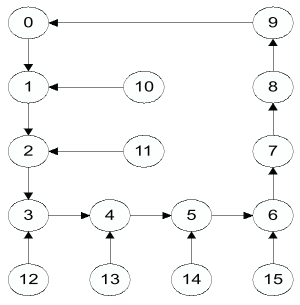

Figure 2. STGs of a 4-bits modulo 10 counter

The specification that we want to check is $f = \mathbf{AG}(\neg(counter[3:0] = 1010))$. The partial exact state transition graph is in (1) of Figure 4. Since state 10 does not have any predecessor, this formula is true.

Let us now find out how approximation and refinement work. The 'Universal quantification Method (L2)' explained in Definition 2.2 is used for lower bound computation and the 'Existential quantification Method (U2)' in Definition 2.1 is for upper bound computation. Let us start with an initial approximate system which includes $counter[3:2]$. This approximation results in a new state transition graph (4) in Figure 4. The approximate system should be refined because no conservative answer can be derived from above results. First, let us apply *Latch Affinity Refinement* method. With this method, $\sigma_0^* = 0.5$ (relational affinity to $counter[0]$) and $\sigma_1^* = 1.0$ (relational affinity to $counter[1]$) as in Figure 3. Detail computation methods can be found in [3].

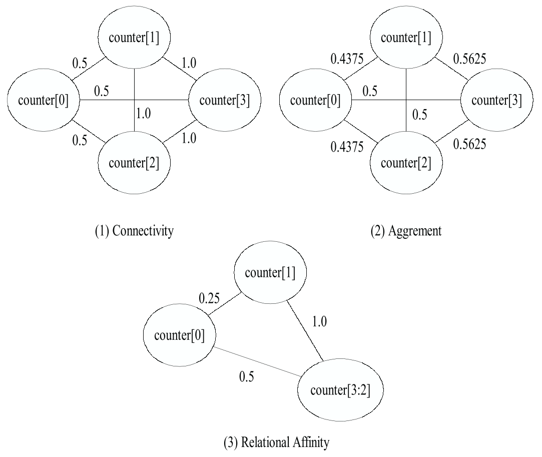

Figure 3. Relational Affinity graph of a 4-bits modulo 10 counter

Based on this factor, $counter[1]$ is selected to refine the approximate system. The refined STGs (2) is changed from (4) in Figure 4. There is no reason to further refine because State 10 does not have preimage now and the formula f is verified true by positive conservatism.

Now, *Edge Traversal Refinement* method is applied. Forward edge traversal computes the first $ee(s,t) = \{(7,8),(7,10),(7,11)\}$. There are two pseudo edges (7,10) and (7,11) in $ee(s,t)$. Notice that these two edge contribute to make the formula f false because there are two paths, (7,10) and (7,11,2,3,4,5,6,7,8,10) which lead the initial state to State 10. The set of pseudo edges that can be killed by $counter[1]$ and $counter[0]$ are:

$$\begin{aligned} kill_{counter[1]} &= \{(7,10),(7,11)\} \\ kill_{counter[0]} &= \{(7,11)\} \end{aligned}$$

This method also chooses $counter[1]$ rather than $counter[0]$ because $\mid kill_{counter[1]} \mid = 2$ is bigger than $\mid kill_{counter[0]} \mid = 1$.

Both methods successfully refine and verify the formula f with three latches. If $counter[0]$ is selected, all latches must be included to verify the formula f. This is because the edge (7,10) is not killed by adding $counter[0]$ only.

5. Approximate Satisfying Don't Care

Reachability analysis of a system has a significant role in model checking. A set of reachable states is used to make transition relation smaller by RESTRICT [5] operation. This minimized transition relation is smaller than original one because unreachable states are used as *don't care states*. This type of *don't cares* is called *Reachability Don't Care*(**RDC**). In [11], a superset of exact reachable

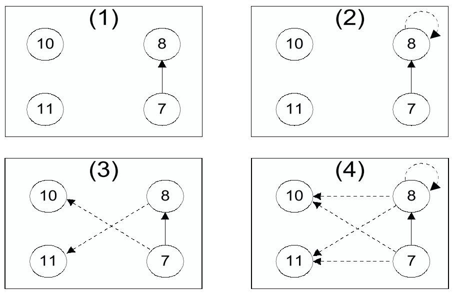

Figure 4. Upper-bound approximate STGs of a modulo 10 counter with different T^+. Only part of the whole STG is shown. A dashed line is a pseudo edge. (1) $T = T_3T_2T_1T_0$, (2) $T^+ = T_3T_2T_1$, (3) $T^+ = T_3T_2T_0$, (4) $T^+ = T_3T_2$.

states is computed and used in the same way. This is called *Approximate Reachability Don't Care (**ARDC**)*.

The benefit of *Iterative Verification* is to use $\overline{\mathbf{SAT}_i^+(v)}$ as don't cares to minimize the transition relation and reduce the cost of computing the refined $\mathbf{SAT}_{i+1}^-(v)$ or $\mathbf{SAT}_{i+1}^+(v)$. $\overline{\mathbf{SAT}^+(v)}$ is called *Approximate Satisfying Don't Care*(**ASDC**). This is possible because the refined system yields tighter satisfying states than previous level of approximate system for a given sub-formula v as below.

$$\mathbf{SAT}^- i(v) \subseteq \mathbf{SAT}_{i+1}^-(v) \subseteq \mathbf{SAT}_{i+1}^+(v) \subseteq \mathbf{SAT}_i^+(v) \qquad (6)$$

In addition to the extra don't cares, the fixpoint computation can be simplified by the following observation. When we compute the greatest fixpoint computation $\mathbf{SAT}_{i+1}^+\mathbf{EG}(p)$ as below, initially y is tautology.

$$\mathbf{SAT}_{i+1}^+(\mathbf{EG}(p)) = \nu y.(\mathbf{SAT}_{i+1}^+(p) \wedge \mathbf{PRE}^+(y)) \qquad (7)$$

But, $y = \mathbf{SAT}_i^+(\mathbf{EG}(p)))$ can be used as the first computation and reduce some redundant steps because of 6. Also, when we compute the least fixed point of $\mathbf{EF}(p)$, $\mathbf{SAT}_i^-(\mathbf{EF}(p))$ is used instead of a empty set by the same reason. Based on these two observations, upper-bound approximate satisfying set enables us to make upper and lower transition relations smaller and both upper- and lower-bound approximate satisfying sets reduce the number of iterations with tighter starting point. **MOG** enjoys these two benefits because every node in the operational graph has both approximations. Any node with only negative polarity in **DOG** can not use **ASDC**. Generally speaking, **MOG**'s overhead to compute both $\mathbf{SAT}^+$ and $\mathbf{SAT}^-$ requires more computing resources than **DOG** uses. But, as the approximate system gets refined close to the exact system, **MOG**'s overhead is compensated with the reduction of the transition relation by **ASDC**. For instance, Example 7 in Table 6 is the case where **pDOG** used 63.9MB memory while **MOG** used 42.9MB only. Figure 5 shows how upper-bound satisfying states are used as care states to minimize transition relations in this Example. The average size of approximate transition relation minimized by **ASDC** is 20.08% of one without **ASDC**.

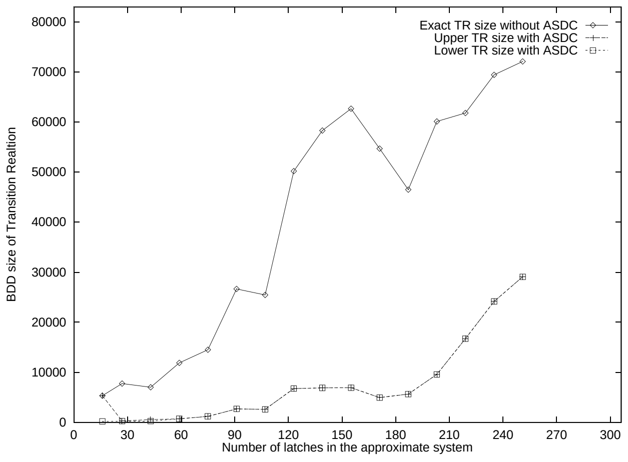

Figure 5. Minimization of $T(s,x,t)$ by ASDC.

6. Experimental Results

We have implemented all the algorithms in this paper in VIS-1.3 (called IMC). All experiments were ran on two different machines. The first one is a Pentium II(400MHz) system with 512K internal L2 cache and 1GB main memory. The other is a Pentium II(233MHz) system with 256K internal L2 cache and 256MB main memory. VIS-1.3 [2] was used as a main platform and CUDD-2.3.0 [14] was used. Here, RM: Refinement Method (S: Latch Affinity Method, D: Edge Traversal), OG: Operational Graph (M: MOG, P: pDOG, N: nDOG), FF: Number of latches in a circuit, RFF: Number of latches reduced by formula dependency, and VFF: $|I^A|$. The results of IMC are compared to the ones from exact model checker (MC). Not every combination of lower-bound methods, upper-bound methods, and refinement methods were tested. But, many examples were chosen from a small circuit to a big one to demonstrate that incremental verification is not restricted to some class of the circuits. Example 4 through 12 were the best results among **nDOG** , **pDOG** and **MOG** . **MOG** was the best in 6 examples out 9. Example 3 is a case where **ASDC** helps model checking even though there is no locality and the approximate system should be increased to the exact system. For all 12 cases, average memory usage of IMC is less than 30.0% of MC (CPU Time is about 15%).

No	Circuit	FF	Formula	T/F	RFF	RM	OG	IMC			MC	
								VFF	MB	TIME	MB	TIME
1	PDC	61	AG(p $\rightarrow$ AF q)	T	61	D	N	58	6.9	23	26.3	1751
2	Example1	115	AG(p $\vee$ AF q)	T	115	D	N	6	5.8	12	14.4	1905
3	Ethernet	118	AG(p $\rightarrow$ AFq)	T	86	D	N	86	13.5	978	20.6	2506
4	CPS	231	AG(EF p)	F	231	S	M	4	9.5	14	19.4	956
5	Hw_Top	356	AG(EF p)	F	356	S	P	12	14.0	64	30.7	182
6			AG(p $\rightarrow$ EF q)	T	277	S	N	62	22.0	1409	90.0	3533
7			AG(EF p)	F	306	S	M	251	42.9	299	248.4	27600
8	TrainFlat	866	AG(p $\rightarrow$ EF q)	T	583	S	N	37	18.4	536	269.7	1781
9			AG(p $\rightarrow$ EF q)	T	609	S	M	16	17.6	207	292.1	1846
10			AG(EF p)	F	609	S	M	16	17.7	203	322.3	4339
11	Avq	3705	AG(p $\rightarrow$ EF q)	T	2715	S	M	3	162.4	668	321.6	1932
12			EF p	T	2772	S	M	11	163.7	922	$>$616.6	$>$86400

Table 1. Experimental results.

7. Conclusions

In approximate model checking, there are two main goals. The first one is efficiency to find out the locality of a formula. The impact of this is significant because the number of BDD variables can be reduced by property locality. The second is how easier approximate computation is than exact one. In this paper, efficient refinement algorithms were introduced for the first goal. The upper-bound and lower-bound methods along with ASDC were proposed to achieve the second target. It is true that there is a formula that does not have any locality. For this kind of formula, the approximation method seems to be useless. But, thanks to **ASDC**, the verification with the last iteration index may be easier than the exact verification when **ASDC** is not empty. The impact of all these new ideas was demonstrated with big examples. Order of magnitude speedup was achieved with 70% reduction in memory consumption.

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] H. Cho, G. D. Hachtel, E. Macii, M. Poncino, and F. Somenzi. A structural approach to state space decomposition for approximate reachability analysis. In *Proceedings of the International Conference on Computer Design*, pages 236–239, Cambridge, MA, Oct. 1994.

[4] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings Workshop on Logics of Programs*, pages 52–71, Berlin, 1981. Springer-Verlag. LNCS 131.

[5] O. Coudert, C. Berthet, and J. C. Madre. Verification of sequential machines using boolean functional vectors. In L. Claesen, editor, *Proceedings IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, Leuven, Belgium, Nov. 1989.

[6] R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes.* Princeton University Press, Princeton, NJ, 1994.

[7] W. Lee, A. Pardo, J. Jang, G. Hachtel, and F. Somenzi. Tearing based abstraction for CTL model checking. In *Proceedings of the International Conference on Computer-Aided Design*, pages 76–81, San Jose, CA, Nov. 1996.

[8] J. Lind-Nielsen and H. R. Anderson. Stepwise ctl model checking of state/event systems. In N. Halbwachs and D. Peled, editors, *Eleventh Conference on Computer Aided Verification (CAV'99)*. Springer-Verlag, Berlin, 1999.

[9] D. E. Long. *Model Checking, Abstraction, and Compositional Verification.* PhD thesis, Carnegie-Mellon University, July 1993.

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

[11] I.-H. Moon, J.-Y. Jang, G. D. Hachtel, F. Somenzi, C. Pixley, and J. Yuan. Approximate reachability don't cares for CTL model checking. In *Proceedings of the International Conference on Computer-Aided Design*, pages 351–358, San Jose, CA, Nov. 1998.

[12] A. Pardo and G. D. Hachtel. Automatic abstraction techniques for propositional μ-calculus model checking. In O. Grumberg, editor, *Ninth Conference on Computer Aided Verification (CAV'97)*, pages 12–23. Springer-Verlag, Berlin, 1997. LNCS 1254.

[13] K. Ravi, K. L. McMillan, T. R. Shiple, and F. Somenzi. Approximation and decomposition of decision diagrams. In *Proceedings of the Design Automation Conference*, pages 445–450, San Francisco, CA, June 1998.

[14] F. Somenzi. *CUDD: CU Decision Diagram Package*. ftp://vlsi.colorado.edu/pub/.