Counterexample-Guided Choice of Projections in Approximate Symbolic Model Checking *

Shankar G. Govindaraju David L. Dill
Computer Systems Laboratory, Stanford University, Stanford, CA 94305
{shankar@encore, dill@cs}.stanford.edu

Abstract

BDD-based symbolic techniques of approximate reachability analysis based on decomposing the circuit into a collection of overlapping sub-machines (also referred to as overlapping projections*) have been recently proposed. Computing a superset of the reachable states in this fashion is susceptible to false negatives. Searching for real counterexamples in such an approximate space is liable to failure. In this paper, the "hybridization effect" induced by the choice of projections is identified as the cause for the failure. A heuristic based on Hamming Distance is proposed to improve the choice of projections, that reduces the hybridization effect and facilitates either a genuine counterexample or proof of the property. The ideas are evaluated on a real large design example from the PCI Interface unit in the MAGIC chip of the Stanford FLASH Multiprocessor.*

1 Introduction and Related Work

One of the key desirable features of model checkers is their ability to generate counterexamples automatically, which can directly aid debugging of the design. However approximate model checking techniques suffer from the problem that it is not always feasible to map a counterexample generated in the approximate space into a valid counterexample in the real design. The approximated design may have extra degrees of freedom, allowing certain transitions not possible in the real design. Analysis of the cause of failure of the counterexample in the approximated space, can highlight what information is lost in the approximation process, and hints on refining the approximation appropriately can then be given. This basic idea has been used by various researchers. Kurshan [8] used it in the context of verification of timed automata, Balarin *et al* [1] use it to check for language containment. Clarke *et al* [5] explore this same basic idea in verification using abstraction functions for different variables in a SMV program. This paper explores the same basic idea in the context of approximation by overlapping projections [6].

*This work was supported by GSRC contract SA2206-23106PG-2.

2 Background: Approximation by Projections

Mealy machines are used to model synchronous hardware design examples, and for our purposes, it is a 4 tuple, $M = \langle x, y, q_0, \mathbf{n} \rangle$, where $x = \{x_1, \ldots, x_k\}$ is the set of state variables, and y is the set of input signals. The set of states is given by $[x \rightarrow \mathcal{B}]$, where $\mathcal{B} = \{0,1\}$. The initial state $q_0 \in [x \rightarrow \mathcal{B}]$. The next state function is $\mathbf{n} : [x \rightarrow \mathcal{B}] \times [y \rightarrow \mathcal{B}] \rightarrow [x \rightarrow \mathcal{B}]$. Using BDDs for the functions $\mathbf{n}$ and a BDD $R(x)$ for the set of states R, the image $(Im(R(x), \mathbf{n}(x,y)))$ and preimage $(Pre(R(x), \mathbf{n}(x,y)))$ of the set R under $\mathbf{n}$ can be computed.

Let $\mathbf{w} = (w_1, \ldots, w_p)$ be a collection of not necessarily disjoint subsets of x. The projection operator α projects a predicate $R(x)$ onto the various w_j's, *i.e.* $\alpha(R(x)) = (\alpha_1(R), \ldots, \alpha_p(R))$, where $\alpha_j(R(x)) = \exists(x - w_j).R(x)$. The associated concretization operator γ conjuncts the collection of projections. Under this scheme of approximation, for any set of states R, the implication $R \rightarrow \gamma(\alpha(R))$ holds. More details on this scheme of approximation can be obtained from [6, 7].

Let Im_{ap} (the subscript *ap* denotes "approximate") return the projected version of the image of an implicit conjunction of BDDs, and let Pre_{ap} return the projected version of the preimage of an implicit conjunction of BDDs.

$$Im_{ap}(\mathbf{R}, \mathbf{n}) = \alpha(Im(\gamma(\mathbf{R}), \mathbf{n}(x,y)))$$
$$Pre_{ap}(\mathbf{R}, \mathbf{n}) = \alpha(Pre(\gamma(\mathbf{R}), \mathbf{n}(x,y)))$$

Efficient algorithms to implement Im_{ap} and Pre_{ap} proposed in [6, 7] are used here. Using Im_{ap}, an overapproximation, $\gamma(FwdReach_{ap}(q_0))$, of the reachable states for a machine M is computed. Using Pre_{ap}, an overapproximation, $\gamma(BackReach_{ap}(\bar{g}))$, of the set of states in M that can reach the set of states $\bar{g}$ is computed as shown below. (Given two equally sized tuples, their *join* ($\sqcup$) is a pairwise disjunction, and their *meet* ($\sqcap$) is a pairwise conjunction of elements of the tuples.)

$$FwdReach_{ap}(q_0) = \text{ lfp } \mathbf{R}.(\alpha(q_0) \sqcup Im_{ap}(\mathbf{R}, \mathbf{n}))$$
$$BackReach_{ap}(\bar{g}) = \text{ lfp } \mathbf{R}.(\alpha(\bar{g}) \sqcup Pre_{ap}(\mathbf{R}, \mathbf{n}))$$

where *lfp* is a least fixed point iteration [3] which starts with $\mathbf{R} = (0, \ldots, 0)$, and on each iteration *joins* the

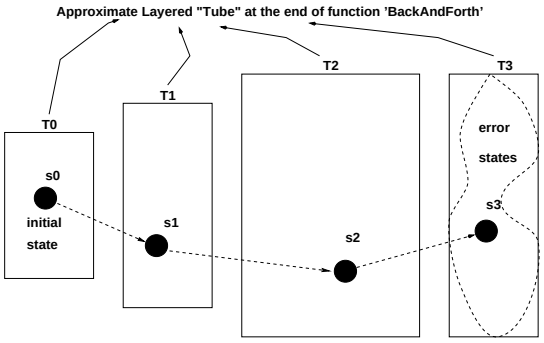

Figure 1. Counterexamples in Final Tube

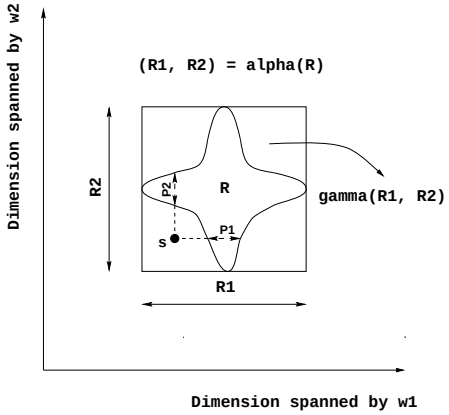

Figure 2. Projections induce Hybridization

current approximate set with the approximate successor set. An overapproximation of the states that lie on a path from the initial state q_0 to a state *not* satisfying a user-specified property g is computed by repeated forward and backwards approximate passes.

function *BackAndForth* (g)
$\mathbf{R_f} \leftarrow (0, \ldots, 0)$
$\mathbf{R_b} \leftarrow (1, \ldots, 1)$
while $(\mathbf{R_f} \neq \mathbf{R_b})$ **do**
  $\mathbf{R_f} \leftarrow$ *lfp* $\mathbf{R}.(\alpha(q_0) \sqcup (Im_{ap}(\mathbf{R}, \mathbf{n}) \sqcap \mathbf{R_b}))$
  if $(\gamma(\mathbf{R_f}) \rightarrow g)$ **return** *"no errors"*
  $\mathbf{R_b} \leftarrow$ *lfp* $\mathbf{R}.(\alpha(\bar{g}) \sqcup (Pre_{ap}(\mathbf{R}, \mathbf{n}) \sqcap \mathbf{R_f}))$
  if $(\gamma(\mathbf{R_b}) \wedge q_0 = 0)$ **return** *"no errors"*
endwhile
return $\mathbf{R_f}$

3 Counterexamples

If *BackAndForth* fails to rule out an error, it is useful to check whether there is an actual error by generating an example path from q_0 to a state that does not satisfy g. This both confirms the existence of an error and provides debugging information to the user.

Starting from the error states, the algorithm computes approximate preimages (intersected with the set of states seen in the previous pass) and stores the preimages obtained at the various iterations of the fixpoint algorithm in a stack. Let $T_0, T_1, \ldots, T_m$ (where T_m intersects with the error states, and T_0 intersects the initial states) be the final contents of the stack. The contents of the stack represent an approximate *tube* through which all counterexamples (if any) pass.

A *single* state, s_0, is chosen from the intersection $q_0 \wedge T_0$, and the exact image of s_0 is computed. If the image of s_0 intersects with T_1, a single state s_1 is chosen from the intersection and the process is repeated (figure 1). This may either generate a valid counterexample or be *stuck* in some state s_j in layer T_j (where *stuck* in s_j means the image of s_j does not intersect with T_{j+1} at all, implying T_{j+1} is approximately reachable from s_j but not exactly reachable from s_j).

States like s_j where the counterexample gets stuck represent *bogus* states, since paths from the initial states to these states in layer T_j cannot be extended to form a complete counterexample. It is useful to analyze what information is lost in the approximation scheme that allows such *bogus* states to creep into the *tube*. Hints can then be provided on how to improve the choice of projections and create more accurate tubes with fewer bogus states.

4 Hybridization

The geometric intuition of the approximation induced by projections is illustrated in figure 2. For simplicity, assume there are only two subsets in our choice of subsets. The irregular shape in figure 2 represents the exact set R, and the outer box represents the set of states obtained after projecting through α, then concretizing through γ. This allows bogus states like s in figure 2 to creep into the approximation tube. For the given choice of subsets, there is some loss of correlation between the state variables in different subsets. In particular, bogus states like s do not have to agree with some real state in R across all the w_1 and w_2 bits, but instead merely need to agree with some real states in R on w_1 bits, and with some *other* real states in R on w_2 bits. This leads to the notion of *hybridization*.

Definition 1 *Let s_1 and s_2 be two states from $[x \rightarrow \mathcal{B}]$. Given a collection of subsets $w = (w_1, w_2)$, the states s_1 and s_2 are said to* hybridize *a state s,* i.e., *$s \in hybrid(s_1, s_2)$ if the following conditions hold:(1) $s \neq s_1$ and $s \neq s_2$, (2) $\alpha_1(s) = \alpha_1(s_1)$ and (3) $\alpha_2(s) = \alpha_2(s_2)$.*

In words, $s \in hybrid(s_1, s_2)$ holds relative to the choice of subsets $w = (w_1, w_2)$ if s agrees with s_1 on the w_1 bits and s agrees with s_2 on the w_2 bits. From figure 2 note that every state from P_1 would hybridize with every state from P_2 to allow bogus state s to creep in.

Let s_1 and s_2 be two states from P_1 and P_2 respectively. Since s_1, s_2 and s are single states, they have a unique assignment to all the state variables in x. For ease of exposition, consider a design with six state variables $x = \{a, b, c, d, e, f\}$, $R = b \vee e$, $w_1 = \{a, b, c, d\}$

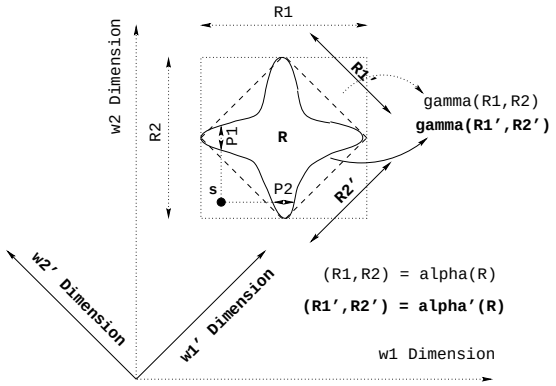

Figure 3. Refinement by the heuristic

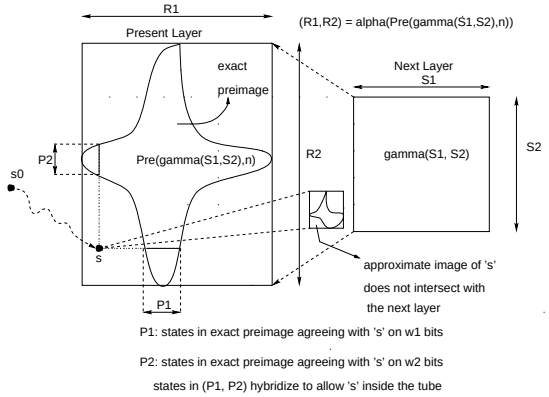

Figure 4. Case 1: Remove Bogus States

and $w_2 = \{c, d, e, f\}$. Let the bit vectors $s = 101101$, $s_1 = 101111$ and $s_2 = 111101$ represent these single states (the leftmost bit in the vector refers to a and the rightmost refers to f). Note that $s_1 \in R$, $s_2 \in R$, but $s \notin R$ as required from figure 2. Also s and s_1 differ in the assignment to variable e, whereas s and s_2 differ in the assignment to variable b.

The key idea is that by looking at the bits that disagree in s and s_1 (s_2), and adding them to the w_1 (w_2) subset, the bogus state s can be eliminated from the approximation. Consider adding to w_1 the bit positions where s_1 and s differ, *i.e.* $w_1' = w_1 \cup \{e\}$. Analogously the new $w_2' = w_2 \cup \{b\}$ is formed by adding to w_2 the bit positions where s_2 and s differ. Relative to this new choice of subsets $w' = (w_1', w_2')$, the bogus state s cannot appear in the present layer because of hybridization. This is because the possible states that could hybridize to give place to the state s, namely $P_1' = \alpha_1'(s) \wedge R$ reduces to 0 and $P_2' = \alpha_2'(s) \wedge R$ reduces to 0. The geometric interpretation of the refinement induced by this heuristic is in figure 3.

The size of the individual subsets in w' increases (and hence the size of BDDs in *BackAndForth*) in this process. To ensure incremental growth of individual subsets, states s_1, s_2 are so chosen from P_1, P_2 respectively, such that they have the smallest *Hamming distance* from s.[1] This will incrementally lead to one or more iterations of augmenting the subsets that will rule out the bogus state s. Formally, the algorithm is:

function *ImproveProj* $((P_1, P_2), s, (w_1, w_2))$
  Choose $s_1 \in P_1$ *s.t.* $\mid s_1 - s \mid$ *is small*
  Choose $s_2 \in P_2$ *s.t.* $\mid s_2 - s \mid$ *is small*
  $w_1' = w_1 \cup$ *bits where* (s, s_1) *differ*
  $w_2' = w_2 \cup$ *bits where* (s, s_2) *differ*
return $w' = (w_1', w_2')$

[1] The *Hamming Distance* between two states p and q, denoted by $\mid p - q \mid$, is defined as the number of bit positions where p and q differ. Finding a state from a set of states that has minimum Hamming distance from some other reference state can be computed efficiently by an algorithm [11] whose complexity is linear in the size of the BDD representing the set.

This heuristic eventually leads to the *shortest possible counterexample*. In contrast to structural methods [4] of choosing the collection of subsets, this is an automatic method of choosing subsets *relative to a property* that needs to be proven. In the general case of more than two subsets in the collection, $w = (w_1, \ldots, w_p)$, the subset w_i is improved relative to the next subset in the collection, *i.e.* $w_{(i+1) \ mod \ p}$.

5 Hamming Distance Heuristic

It is useful to distinguish between two different kinds of bogus states. Suppose the counterexample generation method is stuck at a state s in some layer (R_1, R_2). Depending on whether the approximate image of s intersects with the next layer or not, there are two possible scenarios:

• *Case 1 - Hybridization in present layer:* In other words, even the approximate image of s does not intersect with the next layer (figure 4). Surely the state s does not belong to the exact preimage of the next layer (or a one step transition would have been possible), but gets included as the preimage gets approximated. As in figure 4, P_1 (P_2) is the set of states in the exact preimage of $\gamma(S_1, S_2)$ that agree with s on w_1 (w_2) bits. Every state in P_1 will hybridize with every state in P_2 to produce the bogus state s. The algorithm *ImproveProj* is invoked with arguments $((P_1, P_2), s, (w_1, w_2))$ to obtain an improved choice of projections.

• *Case 2 - Hybridization in next layer:* In other words, the approximate image of s intersects with the next layer (figure 5). As in figure 5, let q be a state in $\gamma(Im_{ap}(s, \mathbf{n}) \sqcap (S_1, S_2))$. Now P_1 (P_2) is the set of states in exact image of s that agree with q on w_1 (w_2) bits. Every state in P_1 will hybridize with every state in P_2 to produce the bogus state q. As in the previous case, the algorithm *ImproveProj* is invoked with arguments $((P_1, P_2), q, (w_1, w_2))$ to obtain an improved choice of projections.

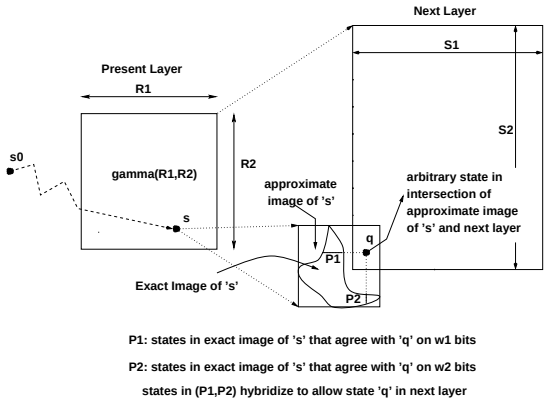

Figure 5. Case 2: Remove Bogus States

5.1 Computation of P_1 and P_2

• **Case 1**, P_1 and P_2 can be computed without computing the exact preimage $Pre(\gamma(\mathbf{S}), \mathbf{n})$. To compute P_1, all the next state functions are constrained with $\alpha_1(s)$, and then passed to the Pre_{ap} algorithm described in [7]. The other arguments passed to the Pre_{ap} algorithm are $\mathbf{S}$, $\mathbf{I}$, $x - w_1$. (where $\mathbf{S} = (S_1, S_2)$ represents the next layer and $\mathbf{I}$ represents the states generated by previous approximate forward reachability pass). In the cases where the set $\{x - w_1\}$ is too big inducing many recursive subproblems in Pre_{ap} algorithm, the recursive algorithm stops as soon as it finds some state in P_1, and in essence computes an underapproximation of P_1. This entails possibly augmenting the subsets with a few more bits than the optimal minimum, but our experiments show that this is not a problem. The Hamming Distance causes size increments in the 1-17 range and is well-behaved to ensure that the collection of subsets becomes coarser incrementally.

• **Case 2**, computing P_1 and P_2 is easy. Since s is a single state, computing the exact image, $Im(s, \mathbf{n})$ is not a problem. P_1 can then be computed by explicitly computing $\alpha_1(q) \wedge Im(s, \mathbf{n})$.

6 Experimental Results

The method was evaluated on the PCI Interface section of the I/O unit from the MAGIC chip, a custom node controller in the Stanford FLASH multiprocessor [9]. Earlier efforts [7] to verify this resulted in many invalid counterexamples because of lack of environment models to model the legal inputs from the PCI bus.

Recently Shimizu *et al* [10] released a formal specification of the PCI Bus protocol. The monitor-style specification of the protocol by Shimizu *et al* [10] enabled connecting the I/O section of the MAGIC chip with the monitors (see Figure 6). The monitors snoop the transactions on the PCI bus and generate signals $ocorrect_i$, for $1 \leq i \leq 66$, to ensure that only legal inputs from the PCI bus go into the I/O unit. At the same time, the monitors snoop the output signals from the I/O unit to ensure that the outputs obey the PCI bus protocol. These output checking signals form the signals $mcorrect_i$, for $1 \leq i \leq 66$, as in figure 6.

The inputs are constrained to keep all the *ocorrects* high. While doing so, if some $mcorrect_i$ goes to 0, then the PCI unit has violated the PCI spec. An example of a property from the *mcorrects* is: *irdy* cannot be asserted by the I/O unit in a cycle if the bus was *idle* in the previous cycle.

Proving properties on the final model: The design size of the resulting verification example was 429 state variables and 85 inputs. The initial choice of subsets was based on the heuristic reported in [6]. State variables encoding the state of the same state machine were kept in a single subset. Thereafter, the Hamming distance heuristic improved the choice of subsets. Eventually, the method completed the proof of 64 properties. The remaining two properties could not be proved (nor could counterexamples be generated for them).

The details are in table 1. The column labeled $\#Proj$ refers to the number of subsets. As the size of the individual subsets becomes larger, some subsets that are totally subsumed by other subsets are removed. The column labeled *Avg.* gives the average size of the subsets, and *Max.* reports the size of the biggest subset. (The 66 *ocorrects* are not included in any of the subsets in w, and instead used as constraints. Further each of the 66 *mcorrects* are included in single sized subsets, which are not counted for the numbers in the $\#Proj$ column.)

The *Size of Tube* (measured in terms of satisfying fraction) is an upper bound on the number of states in the final *tube* inside which all counterexamples must lie. As the subsets become coarser, the size of the tube becomes smaller, as expected. *Nodes* reports the peak number of BDD nodes alive during the experiment, and *Time* reports the time in seconds. The column *Proved* reports the number of *mcorrects* proved after that experiment, and *HD Range* gives data on how much the size of some subsets increased during the experiment. Note that the size of the subsets grew marginally (by 17 bits in the largest case) compared to the size of the final model (which had 429 state variables), even as it enabled the proof of 64 out of 66 properties.

Counterexample generation: The complete FLASH I/O unit is a very large design with nearly 2400 state variables. To ease the task of verification, initially only the core control portion of the I/O unit was included in the model. Proof of many of the *mcorrects* initially failed and counterexamples for the failed properties were generated. The Hamming Distance heuristic helped in the generation of such counterexamples.

The designer would inspect the counterexample, and indicate which part of the I/O unit needed to be added to the model to rule out the counterexample. Typically this is because in the process of cutting out parts of the I/O unit, many internal signals are modeled as non-deterministic inputs in the model. Relevant logic

Table 1. Proving safety properties on the final model

Iter	# Proj	Avg.	Max	Size of Tube	Nodes	Time	Proved	HD Range
1	65	5.06	10	3.646131e-22	298842	2843	11/66	-
2	53	6.35	15	3.207865e-27	307152	3931	11/66	1-8
3	42	8.76	20	9.166265e-30	437237	6496	12/66	4-15
4	41	9.16	29	2.796728e-55	198830	1343	61/66	7-17
5	41	9.17	29	8.739775e-57	128488	977	64/66	1-5
6	41	9.32	29	2.184944e-57	126222	979	64/66	1-3

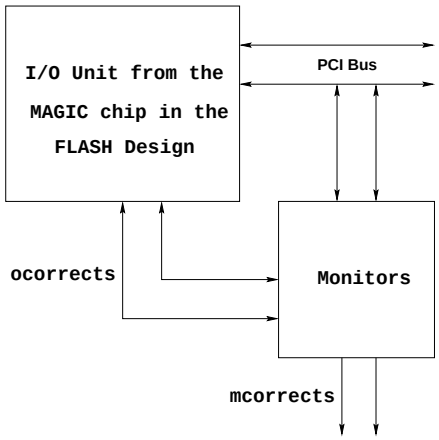

Figure 6. PCI Design Example

from the I/O unit that drove such signals were added to the model and the verification exercise was repeated. In our experience, the Hamming Distance heuristic is helpful in not only improving the choice of subsets to enable proof of a property, but also in generating actual counterexamples (relative to the model) and giving information on which other parts of the I/O unit need to be added to the model. The final model had 429 state variables and 85 input variables.

Example of hints provided by the heuristic: The monitors and I/O unit of the MAGIC chip, both maintain some equivalent internal state to record the past transactions on the PCI bus. The initial choice of subsets left the correspondingly equivalent internal state variables in different subsets. Many of the properties were not provable because of the hybridization effect induced by these equivalent internal state variables not being correlated at all times. The Hamming distance heuristic was able to automatically correct this. We conjecture that most or all of the monitor's state is functionally dependent on state in the I/O unit implementation. This is an interesting avenue for future research.

Discussion and Future Work: The appeal of the ideas in this paper is the automatic failure analysis and automatic modification of the subsets to address the failure. Even though the robustness of the heuristic is evident from the experimental results, there are some avenues for further improvement. Some backtracking could be employed to look for other candidate states in a given layer. This would offset the randomness associated in the present selection of a single state from the exact image to move to the next layer.

Instead of immediately looking for ways to improve the choice of subsets, more effort can be put on searching for counterexamples more exhaustively in the current approximation tube. In particular, methods of computing underapproximations of images and preimages with overlapping projections as the underlying approximation scheme would greatly facilitate this and would fit very well in the overall verification methodology. We are exploring ideas along these lines.

References

[1] Balarin, F. and Sangiovanni-Vincentelli, A. L., "An iterative approach to language containment," *CAV*, pp. 29-40, 1993.

[2] Bryant, R. E., "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, Vol. C-35, No. 8, pp. 677-691, August 1986.

[3] Burch, J. R. *et al.*, "Symbolic model checking: 10^{20} states and beyond," *LICS*, pp. 428-439, 1990.

[4] Cho, H. *et al.*, "Automatic state space decomposition for approximate FSM traversal based on circuit analysis," *IEEE-TCAD*, Vol. 15, No. 12, pp. 1451-1464, December 1996.

[5] Clarke, E. *et al.*, "Counterexample-guided abstraction refinement," *CAV*, pp. 154-169, July 2000.

[6] Govindaraju, G. S., Dill, D. L., Hu, A. J, and Horowitz, M. A., "Approximate reachability with BDDs using overlapping projections," *DAC*, pp. 451-456, 1998.

[7] Govindaraju, G. S. and Dill, D. L., "Verification by approximate forward and backward reachability," *ICCAD*, pp. 366-370, 1998.

[8] Kurshan, R. P., "Timing verification by successive approximation," US Patent US05483470.

[9] Kuskin, J., *et al.*, "The Stanford FLASH multiprocessor," *ISCA*, pp. 301-313, April 1994.

[10] Shimizu, K., Dill, D., and Hu, A. J., "Monitor based formal specification of PCI," *(To appear)*.

[11] Yang, C. H., and Dill, D., "Validation with guided search of the state space," *DAC*, pp. 599-604, 1998.