Approximate Reachability with BDDs using Overlapping Projections *

Shankar G. Govindaraju[1] , David L. Dill[1], Alan J. Hu[2], and Mark A. Horowitz[1]

[1] Computer Systems Laboratory
Stanford University
Stanford, CA 94305, USA

[2] Department of Computer Science
University of British Columbia
Vancouver, B.C, Canada V6T 1Z4

Abstract

Approximate reachability techniques trade off accuracy with the capacity to deal with bigger designs. Cho et al [3] proposed approximate FSM traversal algorithms over a *partition* of the set of state bits. In this paper we generalize it by allowing *projections* onto a collection of nondisjoint subsets of the state variables. We establish the advantage of having *overlapping projections* and present a new *multiple constrain* function for BDDs, to compute efficiently the approximate image during symbolic forward propagation using overlapping projections. We demonstrate the effectiveness of this new algorithm by applying it to several control modules from the I/O unit in the Stanford FLASH Multiprocessor. We also present our results on the larger ISCAS 89 benchmarks.

1 Introduction

Binary Decision Diagrams (BDDs) [1] have enabled formal verification to tackle larger hardware designs than before. However for many large design examples, even the most sophisticated BDD-based verification methods cannot produce exact results because of BDD-size blowup. One alternative is to trade accuracy for BDD size requirements, by using approximate verification algorithms.

Computing the set of reachable states from an initial set is a basic component of many verification algorithms and has other applications as well. An overapproximated reachable set can be viewed as an underapproximated *don't care* set, which has applications in the synthesis domain. It can be used to simplify symbolic model checking efforts, by preventing the model checking algorithms from exploring unreachable states.

*This work was supported by DARPA contracts DABT63-94-C-0054 and DABT63-96-C-0097. The content of this paper does not necessarily reflect the position of the policy of the Government and no official endorsement should be inferred.

1.1 Related Work

Cho et al [3] proposed approximate algorithms to do symbolic forward propagation. Their basic idea was to partition the set of state bits into *mutually disjoint* subsets, and then do a symbolic forward propagation on each individual subset. The individual subsets can be viewed as submachines which have in some ways been torn from other submachines. The original problem is thus reduced to doing the exact symbolic forward propagation over smaller submachines. This induces extra degrees of freedom for the submachines, and hence yields an overapproximation of the reachable state space.

They also propose different variants of the approximated symbolic forward propagation algorithm: MBM (Machine By Machine) and FBF (Frame By Frame) which basically differ in the way they model the interaction among the various submachines. FBF allows interactions among the submachines at each time frame of a least fixed point routine, and hence allows for tighter don't care *sequences* to constrain the other submachines. MBM on the other hand allows interaction only after a complete least fixed point has been computed for a submachine. As a result the sequencing information is lost when trying to constrain the other submachines. They further propose two variants of the FBF scheme, RFBF (Reached Frame By Frame) and TFBF (To Frame By Frame), which again differ in the constraint set posed to the various sub machines during the course of the least fixed point routine. Cho et al [3, 4] also propose heuristics on how to partition the set of state bits.

1.2 Contribution

In this paper, we improve on the approximate symbolic reachability analysis of Cho et al [3] by allowing for *overlapping projections*. We establish the need for overlapping projections, and propose a new *multiple constrain* function for BDDs, which allows us to compute efficiently the image of an *implicit* conjunction of BDDs with possibly overlapping support, using Boolean function vectors. We apply our algorithm on a real, large design and show its relative superiority over the FBF algorithms. Of course our scheme cannot be less accurate than the FBF method, since overlapping projections include disjoint partitions as a special case.

2 Background

We analyze synchronous hardware, given as a Mealy machine $M = \langle x, y, q_0, \mathbf{n} \rangle$, 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 $[x \to \mathcal{B}]$, where $\mathcal{B} = \{0,1\}$. The initial state $q_0 \in [x \to \mathcal{B}]$. The next state function is $\mathbf{n} : [x \to \mathcal{B}] \times [y \to \mathcal{B}] \to [x \to \mathcal{B}]$.

In our applications, sets can be viewed as predicates, since we can form the characteristic function corresponding to a set. BDDs can be used to represent predicates and manipulate them [2]. For example, let $R(x)$ be a predicate with support in x, we can compute the image of R under $\mathbf{n}$ as

$$Im(R(x), \mathbf{n}(x, y)) = \lambda x'.\exists x, y.(x' = \mathbf{n}(x, y)) \wedge R(x).$$

Im produces a predicate with support x', which is 1 *iff* x' is in the image of R under $\mathbf{n}$. The set of reachable states in M can be computed by a least fixpoint iteration [2]

$$Reach(M) = \text{ lfp } R.\lambda x.(q_0(x) \vee Im(R(x), \mathbf{n}(x, y))).$$

2.1 Approximation by Projections

Let $\mathbf{w} = (w_1, \ldots, w_p)$ be a collection of not necessarily disjoint subsets of x. Each subset will be referred to as a *block*. We define the operator $\alpha_j(R)$ which projects a predicate $R(x)$ onto the variables in w_j. Intuitively, $\alpha_j(R)$ represents a set of Boolean vectors that agree for the variables in w_j with some Boolean vector satisfying R. Let z consist of all of the Boolean variables in x that are *not* in w_j, then we can define α_j as

$$\alpha_j(R(x)) = \lambda x.\exists z.R(x).$$

From the explanation above, it should be clear that the set of Boolean vectors satisfying R is a subset of those satisfying $\alpha_j(R)$. This can be written using logical implication as $R \to \alpha_j(R)$. The approximation operator α projects a predicate $R(x)$ onto the various w_j's and returns the tuple,

$$\alpha(R(x)) = (\alpha_1(R), \ldots, \alpha_p(R)).$$

The concretization operator γ conjoins the collection of projections:

$$\gamma(R_1, \ldots, R_p) = \bigwedge_{j=1}^{p} R_j.$$

Lemma 1 *Given a collection of subsets* $(w_1, \ldots, w_p)$ *and a predicate* $R(x)$, $R \to \gamma(\alpha(R))$.

The proof for this lemma is simple since $R \to \alpha_j(R)$ for all j. Thus projecting a predicate R onto a collection of subsets, and then concretizing the projections by γ results in an overapproximation.

Let $\mathbf{R} = (R_1, \ldots, R_p)$ and $\mathbf{S} = (S_1, \ldots, S_p)$ be two equally sized tuples. We define the *join* operator between $\mathbf{R}$ and $\mathbf{S}$ as follows:

$$(R_1, \ldots, R_p) \sqcup (S_1, \ldots, S_p) = (R_1 \vee S_1, \ldots, R_p \vee S_p)$$

Note that $\gamma(\mathbf{R}) \cup \gamma(\mathbf{S}) \subseteq \gamma(\mathbf{R} \sqcup \mathbf{S})$. Hence the join operator is an approximation of set union.

The operator α allows us to represent a big BDD with support in x by a tuple of potentially smaller BDDs with limited support, at the cost of loss of accuracy. In contrast, γ can potentially result in a bigger BDD with bigger support, hence we would like to avoid computing $\gamma(R_1, \ldots, R_p)$ explicitly. We therefore need to compute the image of an implicit conjunction and return the result as an implicit conjunction of the elements of a tuple. Let Im_{ap} return the projected version of the image of an *implicit* conjunction of BDDs.

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

Using Im_{ap}, we can compute an overapproximation, $Reach_{ap}(M)$, of the reachable states for a machine M as follows:

$$Reach_{ap}(M) = \text{ lfp } \mathbf{R}.(\alpha(q_0) \sqcup Im_{ap}(\mathbf{R}, \mathbf{n}))$$

Note that the least fixpoint routine above starts with $\mathbf{R} = (0, \ldots, 0)$, and finally after reaching convergence, it returns a tuple $\mathbf{R}$ to $Reach_{ap}(M)$.The overapproximate reachable states set is the *implicit* conjunction $\gamma(Reach_{ap}(M))$.

Theorem 1 *For a given Mealy machine M,*

$$Reach(M) \quad \to \quad \gamma(Reach_{ap}(M)) \tag{1}$$

The proof relies on the observation that during computation of $Reach_{ap}(M)$, the image at *every* iteration of the least fixpoint routine is an overapproximation. The formal proof is omitted.

3 Overlapping Projections

3.1 Motivation for Overlaps

Overlapping projections can capture limited interactions among state machines while keeping the sizes of the BDDs under control. We discuss some common scenarios where this happens in this subsection. In contrast, disjoint partitions can only capture interactions among a set of state machines by including all of them in a single projection, which often leads to large variable subsets, that cause BDD size blowup.

Often, two rather big state machines have a small interface, which can be captured by adding extra blocks to our collection $\mathbf{w}$, that merely include the bits through which the two machines communicate. Design modules usually have a *master* FSM that communicates with a number of other *slave* FSMs. A collection of overlapping subsets allows us to capture the master-slave behavior by having blocks, where the master is paired with each of its slaves in *different* blocks. We can further capture the correlation between various FSMs by having small blocks with pairs of FSMs in them.

In each of the these cases, disjoint partitions require larger sized blocks to capture the same property. Thus overlapping subsets allow us to hit intermediate points in the *memory space* vs *strength of invariant* tradeoff curve, with disjoint partitions on one extreme and exact reachability on the other.

3.2 Multiple Constrained Image

The key step in symbolic forward propagation algorithms is image computation.

$$Im_{ap}(\mathbf{R}, \mathbf{n}) = (S_1, \ldots, S_p) = \alpha(Im(\gamma(\mathbf{R}), \mathbf{n}(x, y)))$$

We would like to be able to compute the S_j's separately, without computing $Im(\gamma(\mathbf{R}), \mathbf{n})$. Clearly S_j can only depend on the next state functions of the variables appearing in the j^{th} block, w_j in $\mathbf{w}$. In our implementation, $\mathbf{n}(x, y)$ is represented as a set of predicates $\{n_i(x, y) \mid 1 \leq i \leq k\}$, where each predicate determines the value of a bit in the next state. Let $\alpha_j(\mathbf{n})$ be the subset of predicates determining the next state for the bits in w_j. Clearly, $S_j = Im(\gamma(\mathbf{R}), \alpha_j(\mathbf{n}))$

To avoid unnecessary BDD blowup, we want to avoid the explicit conjunction $\gamma(\mathbf{R})$. S_j can be computed, by forming the next state *relation* for block w_j and using early quantification [2, 8]. However this did not work when we tried it on our larger examples. Instead Coudert and Madre [5, 8] have shown how to compute the image of a Boolean function vector, using the *generalized cofactor* (also called *constrain*) operator ($\downarrow$). $(f \downarrow g)(x)$ has the same value as $f(x)$ when $g(x)$ holds, and usually results in a smaller BDD than f. Generalized cofactor allows us to cofactor a function, f, with respect to another function, g, and reduces to ordinary cofactor when g is a single variable. So $f \downarrow x_i$ is the cofactor of f when $x_i = 1$.

Coudert and Madre [5] show that $Im(\gamma(\mathbf{R}), \alpha_j(\mathbf{n}))$ $= Im(1, \alpha_j(\mathbf{n}) \downarrow \gamma(\mathbf{R})$. To avoid computing the large BDD for $\gamma(\mathbf{R})$, it is tempting to compute $\alpha_j(\mathbf{n}) \downarrow R_1 \downarrow R_2 \ldots \downarrow R_p$. This works well if the supports of R_i's are disjoint. (McMillan has shown [7] that **if** g and h have independent support, **then** $f \downarrow (g \wedge h) = (f \downarrow g) \downarrow h)$. However since we have overlapping subsets, the naive method is incorrect. For example, consider $f = wxy, g = w$ and $h = \bar{w} \vee wxy$. $f \downarrow (g \wedge h) = 1$, and so $Im(g \wedge h, f) = \{1\}$. However $(f \downarrow g) \downarrow h = w \vee \bar{w}xy$, and $Im(1, (f \downarrow g) \downarrow h) = \{0, 1\}$ (we used the variable order $w < x < y$ for this example).

Instead, for overlapping projections, we propose the following method of *multiple constrain.* Let $(z_1, \ldots, z_p)$ be dummy state bits with corresponding next state functions $(R_1, \ldots, R_p)$. The multiple constrain method relies on the following key observation

$$\begin{aligned} &Im(\gamma(R_1, \ldots, R_p), \alpha_j(\mathbf{n})) = \\ &\quad Im(1, [\alpha_j(\mathbf{n}), R_1, \ldots, R_p]) \downarrow z_1 \downarrow z_2 \ldots \downarrow z_p \end{aligned}$$

In words, we first extend the Boolean function vector $\alpha_j(\mathbf{n})$ with $(R_1, \ldots, R_p)$, and compute the range of the extended vector to get the set of next states. Every point in the range of $[\alpha_j(\mathbf{n}), R_1, \ldots, R_p]$ will be "tagged" with the dummy variables z_i, which keep track of which of the R_i's were satisfied in the present state. The required image is the part of the range where all the dummy bits $(z_1, \ldots, z_p)$ are 1, where all the R_i's were satisfied by the present state. Selecting the cofactors where $z_1 = z_2 = \ldots = z_p = 1$ finds the BDD for the relevant part of the range while eliminating the dummy z_i variables. In our example, our *multiple constrain* would compute $(f \downarrow h) \downarrow (g \downarrow h) = 1$. Hence we get $Im(1, (f \downarrow h) \downarrow (g \downarrow h)) = \{1\}$ which matches $Im(f, g \wedge h)$.

We can optimize on the usual recursive co-domain partitioning algorithm [5], by avoiding computing the parts of the range that will be discarded. Hence, we start with $[\alpha_j(\mathbf{n}), R_1, \ldots, R_p]$, constrain the vector of predicates, $[\alpha_j(\mathbf{n}), R_1, \ldots, R_{p-1}]$, with R_p, then constrain the resulting $[\alpha_j(\mathbf{n}), R_1, \ldots, R_{p-2}]$, with the constrained R_{p-1} and so on till we constrain the resulting $\alpha_j(\mathbf{n})$ with the final R_1. Thereafter we can do the standard recursive image computation given by Coudert and Madre [5].

function $Im_{mc}((R_1, \ldots, R_p), (n_1, \ldots, n_m))$
$\mathbf{v} \leftarrow [n_1, \ldots, n_m, R_1, \ldots, R_p]$
for *j=p* **down to** *1* **by** *1* **do**
$\quad\mathbf{v} \leftarrow \mathbf{v} \downarrow v[m + j]$
endfor
return $Im(1, \{v[1], \ldots, v[m]\})$

Our least fixpoint routine starts with $\mathbf{R}$: $(0, \ldots, 0)$ and computes the tuple $Reach_{ap}$ as,

$$\text{lfp } \mathbf{R}.(\alpha(q_0) \sqcup (Im_{mc}(\mathbf{R}, \alpha_1(\mathbf{n})), \ldots, Im_{mc}(\mathbf{R}, \alpha_p(\mathbf{n}))$$

Our algorithm most closely resembles the RFBF algorithm proposed by Cho et al [3], but differs in that we allow for overlapping projections and compute the image for each block with our new Im_{mc} operator. It is also straightforward to do MBM, TFBF, TMBM [3] traversals using overlapping projections.

3.3 Choice of Collection of Subsets

Our scheme for choosing the collection of subsets is presently manual. First, we find the FSMs by inspecting the HDL source (we had access to the RTL description for our design examples). For each state bit x_i we compute a score by counting the number of predicates $n_j(x, y)$ it supports. To each machine, a score is assigned which is the sum of the scores of its state bits. The two machines (M_1, M_2) with the highest scores are identified as *master* FSMs. If the state bits of machines M_i and M_j support the bits of the *master* machine M_1 in their next state predicates, then M_i and M_j are *slaves* of M_1. The different slave machines for each of the master FSMs are identified. We then form blocks by pairing the master FSMs with their slaves. Thus, in this case, we would add the blocks (M_1, M_i) and (M_1, M_j) to the collection of subsets.

Often some FSMs are very small. The corresponding small blocks can then be aggregated with other blocks without running into intermediate image BDD size explosion. The converse problem is some FSM, M_i may have large state registers, resulting in big blocks. If so, we try to prune these blocks by exploiting the *small interface* phenomenon, described in Section 3.1. We also add a block with the master FSMs,to capture the *correlation* between the FSMs. We ensure that no block w_i in the collection $\mathbf{w}$ is a proper subset of another block $w_j \in \mathbf{w}$, since this would clearly be wasteful.

4 FLASH Design Example

The Stanford FLASH (Flexible Architecture for SHared memory) multiprocessor [6] efficiently integrates support for cache coherent shared memory and high performance message passing. Each node in FLASH contains a microprocessor (MIPS R10000), a portion of the machine's global memory, a port to the interconnection network, an I/O interface and a custom node controller called MAGIC (Memory And General Interface Connect). MAGIC handles all communication both within the node and among nodes, using a embedded superscalar, dual issue RISC processor core. We focus on the *control logic* in the I/O unit, since bugs more often than not reside in the control logic rather than the datapath. (The MAGIC chip design description has a rather clean division between the control and datapath). Table 1 gives a brief description of the various control modules in the I/O unit.

Table 1: Control Modules in I/O unit in FLASH

Module	State Bits	Input Bits
IOInboxQCtl	23	8
ReqDecode	37	27
ReqService	41	58
IOMiscBusCtl	44	18
PciInterface	88	55

5 Experiments

We built a LISP interface to David Long's BDD package. Our approximate algorithm returns a superset of the reachable states, which is also an invariant of the design. To quantify the size of the superset, we compute the satisfying fraction of the the superset. (Please refer to the appendix for the algorithm that was used to compute an upper bound on the satisfying fraction). Since projection induces an overapproximation, the smaller the satisfying fraction, the stronger the invariant.

We preset the maximum number of BDD nodes (BDD Node Limit) for each experiment. Given our partitioning heuristics, we try to get the strongest invariant using overlapping projections. We compare our results with the disjoint partition schemes. The same variable ordering was used for both the schemes. *Node Count* keeps track of the highest number of nodes that existed at a time during the experiment. The *Iter* column lists the number of iterations needed to reach the fixpoint. The last column under the heading *Relative* is the ratio between the satisfying fraction with disjoint partitions and the satisfying fraction with overlapping projections. Thus, higher the figures in the *Relative* column, the stronger is the invariant with overlapping projections.

Table 2a: *IOInboxQCtl Invariant Strengths:* Note that to improve upon the invariant with satisfying fraction 5.004883e-03, in the case of disjoint partitions, the BDD node count had to jump from 28,254 to 76,630. which is a 2.71 times increase in the BDD node count. The last entry under disjoint partitions was computed with all the state variables in a single block, which clearly gives the strongest possible invariant. Overlapping projections produces the strongest invariant at much lower node count.

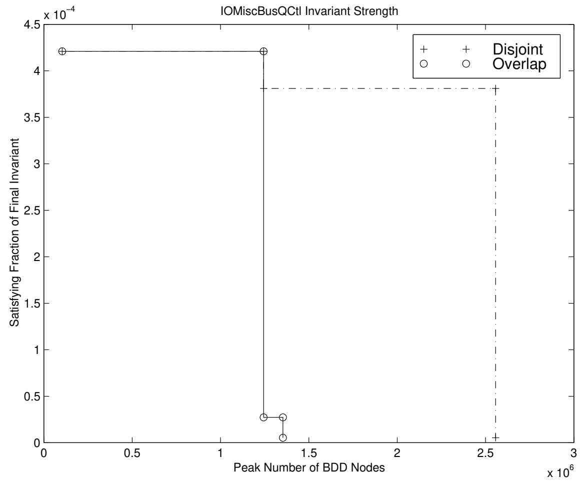

Figure 1: IOMiscBusCtl: Projections vs Partitions

Table 2b: *ReqDecode Invariant Strengths:* For node limits of 1.0 million and 1.5 million, our algorithm with overlapping projections yields stronger invariants (by a factor of 1.252 and 1.562 respectively). Further, for the node limit of 1.0 million, our algorithm with overlapping projections uses lesser number of BDD nodes compared to the FBF runs with disjoint partitions, for a stronger invariant.

Table 2c: *ReqService Invariant Strengths:* Here, with disjoint partitions, the node count penalty goes up from 407,728 to 11,007,330 (a factor of 27) before we see any improvement in the strength of the invariant. The last entry under disjoint partitions was computed with all state variables in a single block, which extracts the strongest invariant. Note that the same invariant is extracted by our overlapping projections scheme at much lower node count penalty (1,995,304 nodes vs 11,007,330 nodes, which is lower by a factor of 5.517).

Table 2d: *IOMiscBusCtl Invariant Strengths:* Note that with node limits of 1.5 million and 2.0 million, our algorithm with overlapping projections yields significantly stronger invariants (by a factor of 13.973 and 71.329 respectively), at only incremental extra cost in terms of BDD node count. In Figure 1, we plot the satisfying fraction of the final invariant *vs* the peak number of BDD nodes. We see that the solid curve for overlapping projections is considerable below the other curve for disjoint partitions, indicating that overlapping projections give stronger invariants with lower BDD node counts compared to disjoint partitions.

Table 2e: *PciInterface Invariant Strengths:* The final invariant with overlapping partitions is much stronger (factor of 5.932) than that obtained with disjoint partitions. Note that even as the node limit is raised from 2 million to 25 million, there is no improvement in the disjoint partition case.

6 ISCAS Benchmarks

We have also tried our algorithm on the ISCAS 89 benchmark suite. We use the partitions used by Cho et al [3] to identify the FSMs in the design. We chose the variable subsets by adding small overlaps to some of their blocks. We are unable to report comparative figures for s35932 because we could not procure the partitions used by Cho et al for s35932. In the case of of s5378, our version of s5378 had 179 flip flops as opposed to the 164 flip flops in the one used by Cho et al. We report our results on the remaining benchmarks. The numbers under the *Disjoint Partitions* column correspond to the results we got by running TMBM algorithm [3] (s13207, s15850, s38584) and RFBF algorithm (s1238, s1423) on the partitions used by Cho et al [3]
Table 3: *ISCAS-89 Invariant Strengths:* Note that we report orders of magnitude improvement in the strength of the invariant for s13207 and s38584. The numbers in Table 3 under overlapping projections are *upper bound estimates* of the satisfying fraction of the final invariant. Thus, the invariant with overlapping projections is stronger, *at least* by a factor equal to the figures under the *Relative* column. (TMBM algorithm starts off as TFBF and switches to MBM after a few iterations. Since we are using TMBM algorithm for some circuits, the *Iter* column in Table 3, lists the number of iterations of doing TFBF + the number of iterations in the outer greatest fixpoint of MBM).

7 Conclusions

In this paper we have proposed a new approximation scheme that enables us to do symbolic forward reachability analysis over an overlapping projection of the set of state bits. The approximation scheme results in tighter overapproximations compared to earlier schemes based on disjoint partitions. Our experiments show that a small amount of appropriately chosen overlaps in a given projection can substantially improve the quality of the overapproximation. Overlapping projections allow us to hit intermediate points in the *memory space* vs *strength of invariant* tradeoff curve, with disjoint partitions on one extreme and exact reachability on the other.

We have also proposed a new *multiple constrain* function for BDDs, that enables us to compute efficiently the image of an *implicit* conjunction of BDDs with possibly overlapping support, using Boolean function vectors.

We further need to look at automatic methods for choosing the collection of subsets, from gate level descriptions.

8 Acknowledgments

We thank David Long for his quick responses for BDD package support. We thank Enrico Macii for sending us the partitions that were used by Cho et al in their paper [3]. We further thank Jules Bergmann for helping us use his tool, *vex*, as a front end Verilog parser for the FLASH benchmarks.

9 Appendix

9.1 Approximating Sat_Fr of Superset

Given $\mathbf{S} : (S_1, \ldots, S_p)$, corresponding to the collection $\mathbf{w} : (w_1, \ldots, w_p)$, we want an upper bound on *sat_fr* of $\gamma(\mathbf{S})$. If elements of $\mathbf{S}$ have mutually disjoint support, we could compute *sat_fr* exactly as $\prod_{i=1}^{p} sat\_fr(S_i)$. However here, S_i's may have overlapping support. Our greedy algorithm computes *sat_fr* of a superset of $\gamma(\mathbf{S})$, by using the fact, $\exists x.(a \wedge b) \subseteq (\exists x.a) \wedge (\exists x.b)$.

A set Z is used to keep track of which variables to hide existentially before computing *sat_fr* of each block. At every step the BDD S_m, with the lowest *sat_fr* (after hiding existentially variables in Z from S_m), is picked. Its *sat_fr* is cumulatively multiplied to f, and variables in w_m are added to set Z.

$Z \leftarrow \emptyset$; $f \leftarrow 1.0$
for *j=1* **up to** p **by** *1* **do**
  find m, s.t. $\forall i.(sat\_fr(\exists Z.S_m) \leq sat\_fr(\exists Z.S_i))$
  $f \leftarrow f \times sat\_fr(\exists Z.S_m)$
  $Z \leftarrow Z \cup w_m$
endfor
return f

An alternative approach, Monte Carlo simulation, appears to be ineffective because of the extreme sparseness of the state space covered by $\gamma(\mathbf{S})$.

References

[1] Bryant, R. E., "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, Vol. C-35, No. 8, pp. 677-691, August 1986.

[2] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D, L, and Hwang, L. J., "Symbolic Model Checking: 10^{20} States and Beyond," *Proceedings of the Conference in Logic in Computer Science*, pp. 428-439, 1990.

[3] Cho, H., Hachtel, G., Macii, E., Pleisser, B., and Somenzi, F., "Algorithms for Approximate FSM Traversal Based on State Space Decomposition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 15, No. 12, pp. 1465-1478, December 1996.

[4] Cho, H., Hachtel, G., Macii, E., Poncino, M., and Somenzi, F., "Automatic State Space Decomposition for Approximate FSM Traversal Based on Circuit Analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 15, No. 12, pp. 1451-1464, December 1996.

[5] Coudert, O., and Madre, J. C., "A Unified Framework for the Formal Verification of Sequential Circuits," *IEEE International Conference on Computer-Aided Design*, pp. 126-129, 1990.

[6] Kuskin, J., Ofelt, D., Heinrich, M., Heinlein, J., Simoni, R., Gharachorloo, K., Chapin, J., Nakahira, D., Baxter, J., Horowitz, M., Gupta, A., Rosenblum, M., and Hennessy, J., "The Stanford FLASH Multiprocessor," *Proceedings of the 21st International Symposium on Computer Architecture*, pp. 301-313, April 1994.

[7] McMillan, K. L., "A Conjunctively Decomposed Boolean Representation for Symbolic Model Checking," *In Proceedings of Computer-Aided Verification*, pp. 13-25, 1996.

[8] Touati, H. J., Savoj, H., Lin, B., Brayton, R. K., and Sangiovanni-Vincentelli, A., "Implicit State Enumeration of Finite State Machines using BDDs," *IEEE International Conference on Computer-Aided Design*, pp. 130-133, 1990.

Table 2: FLASH Design Example Results

Node Limit	Disjoint Partitions			Overlapping Projections			Relative
	Sat. Fr.	Iter	Node Count	Sat. Fr.	Iter	Node Count	
30000	5.004883e-03	20	28254	5.004883e-03	20	28254	1.000
60000	"	"	"	4.943967e-03	20	53740	1.012
70000	"	"	"	3.967404e-03	20	64462	1.262
80000	3.967404e-03	20	76630	3.967404e-03	20	64462	1.000
Node Limit	Disjoint Partitions			Overlapping Projections			Relative
	Sat. Fr.	Iter	Node Count	Sat. Fr.	Iter	Node Count	
50000	2.184883e-05	20	33408	2.184883e-05	20	33408	1.000
200000	2.107944e-05	20	134536	1.979293e-05	20	171448	1.065
1000000	1.274049e-05	33	980968	1.017604e-05	20	608726	1.252
1500000	"	"	"	8.156174e-06	20	1195109	1.562
2500000	3.168825e-06	25	2032890	3.168825e-06	25	2032890	1.000
Node Limit	Disjoint Partitions			Overlapping Projections			Relative
	Sat. Fr.	Iter	Node Count	Sat. Fr.	Iter	Node Count	
50000	1.658440e-02	34	23662	1.658440e-02	34	23662	1.000
500000	1.351655e-03	44	407728	1.053363e-03	37	470642	1.283
750000	"	"	"	1.039535e-03	40	537578	1.300
1800000	"	"	"	1.039460e-03	40	1776965	1.300
2000000	"	"	"	1.036219e-03	44	1995305	1.304
12000000	1.036219e-03	44	11007330	"	"	"	1.000
Node Limit	Disjoint Partitions			Overlapping Projections			Relative
	Sat. Fr.	Iter	Node Count	Sat. Fr.	Iter	Node Count	
1000000	4.210770e-04	4	104135	4.210770e-04	4	104135	1.000
1500000	3.810450e-04	4	1173863	2.726912e-05	4	1244294	13.973
2000000	"	"	"	5.342066e-06	4	1353024	71.329
3000000	5.342066e-06	4	2556733	"	"	"	1.000
Node Limit	Disjoint Partitions			Overlapping Projections			Relative
	Sat. Fr.	Iter	Node Count	Sat. Fr.	Iter	Node Count	
500000	3.574305e-03	21	283186	1.168328e-04	50	228390	30.593
1000000	1.041354e-05	55	598257	1.041354e-05	55	598257	1.000
2000000	9.463498e-07	71	1616055	6.311379e-07	71	1293062	1.499
25000000	"	"	"	1.595220e-07	71	20851528	5.932

Table 3: ISCAS 89 Benchmark Results

Circuit	Disjoint Partitions			Overlapping Projections			Relative
	Sat. Fr.	Iter	Node Count	Sat. Fr.	Iter	Node Count	
s1238	7.036972e-02	3	59360	6.320953e-03	4	73849	11.133
s1423	2.985005e-03	37	310461	2.193374e-03	248	1032286	1.361
s13207	3.421447e-106	10+6	161447	1.136200e-115	10+5	198779	3.3208e+08
s15850	5.840135e-102	10+5	271093	3.937940e-102	10+4	336048	1.483
s38584	6.494194e-41	10+2	646258	5.764063e-57	10+5	1853461	8.876e+15