Symbolic Reachability Analysis of Large Finite State Machines Using Don't Cares

Youpyo Hong and Peter A. Beerel
Department of Electrical Engineering - Systems
University of Southern California, Los Angeles, CA 90089

Abstract

Reachability analysis of finite state machines is essential to many computer-aided design applications. This paper presents novel techniques to improve both approximate and exact reachability analysis using don't cares. First, we propose an iterative approximate reachability analysis technique in which don't care sets derived from previous iterations are used to improve the approximation in subsequent iterations. Second, we propose new techniques to use the final approximation to enhance the capability and efficiency of exact reachability analysis. Experimental results show that our techniques can improve traditional reachability analysis significantly. Moreover, we believe that most state-of-the-art reachability analysis techniques can also be improved by including our techniques.

1 Introduction

Symbolic reachability analysis techniques using binary decision diagrams (BDD's) [1] were introduced by Coudert et al. [4], and have been shown to be able to analyze much larger FSMs than was possible using explicit state techniques which process one state at a time [4, 5, 6]. Nevertheless, symbolic techniques cannot handle some large FSMs because they either require too much memory or are computationally too expensive.

Various techniques have been developed to enhance the capability and efficiency of symbolic reachability analysis. One class of the techniques is motivated by the observation that the size of a BDD depends greatly on the ordering of the BDD variables [1]. Many heuristics have been developed to find a good *initial* variable ordering for BDDs representing the transition relation of FSMs [2]. The initial variable ordering, however, is often not a good ordering for BDDs created during reachability analysis, so techniques to dynamically reorder the variables (e.g. [3]) are widely used [11, 17, 18].

Numerous other techniques to analyze large FSMs have been developed. For example, Ravi et al. [17] proposed a mixed breadth-first and depth-first traversal to utilize subsets of states that are representable with small BDDs. This technique can reduce the size of the BDDs significantly but may increase the number of iterations to complete reachability analysis. Cabodi et al. [18] and Narayan et al. [19] proposed to *partition* state sets to avoid the problem of too many iterations. In addition, Cabodi et al. [20] presented an approach that extends the application of disjunctive partitioned transition relations from asynchronous circuits to synchronous ones and utilizes iterative squaring. Their approach is effective when the FSM analysis requires many iterations.

The performance of reachability analysis may also be improved by applying BDD minimization using don't cares (DCs) [4,12] to reduce the size of the BDDs. For example, Coudert et al. [4] proposed to reduce the *frontier set* BDD which is used as a state space search frontier at each iteration. In addition, Brayton et al. [12] presented techniques to reduce the size of the *transition relation BDDs* (*TR*) which represent the possible state transitions of FSMs. They proposed to use transitions originating from non-frontier states and transitions to known reachable states as DC transitions. The major disadvantage of this approach is that the minimization of the transition relation BDD needs to be repeated at each iteration times because the frontier set and consequently the DC set is constantly changing.

Moreover, Ranjan et al. [11] discussed techniques to reduce the size of TRs using DCs derived from overapproximated reachable states [13]. The main advantage of this approach is that the minimization needs to be performed only once because the DC set is fixed for the entire reachability analysis. However, surprisingly, the BDD minimization led to larger BDDs in their experiments and performance improvements were not reported in their work.

In this paper, we present new techniques to make use of the overapproximated reachable states to improve both approximate reachability analysis and exact reachability analysis. Novel aspects of our approach include:

- A *clustered DC BDD* constructed from partitioned DC BDDs to maintain high BDD minimization ratio with manageable DC BDD size.
- *Iterative* approximate reachability analysis in which DCs computed in previous iterations are used to improve the quality of the approximation computed in subsequent iterations.
- A heuristic to *minimize the support set size of TRs* to improve the early quantification schedule in exact reachability analysis.

Our experimental results demonstrate the effectiveness of the new techniques.

After describing background material in Section 2, we present our techniques to improve standard reachability analysis using DCs in Section 3. Section 4 reports our experimental results and Section 5 presents our conclusions.

2 Background

In this paper, we consider synchronous sequential circuits and we omit the standard input/output specification of the FSM description to focus on the behavior of the state variables. To describe the state transitions we use two sets of variables: present variables, denoted by x, and next state variables, denoted by y. The next state function δ for a FSM with n state bits consists of δ_i's for $1 \le i \le n$, referred to as a *transition function vector* [4], where δ_i represents the next state logic for i^{th} state bit. A *transition relation* [4], denoted by *TR,* defines all possible current/next state pairs using the conjunction of the transition function vectors as follows.

$$TR(x, y) = \prod_{i=1}^{k} (y_i \equiv \delta_i(x)) = \prod_{i=1}^{k} TF_i.(x, y).$$

To represent the transition relations, symbolic techniques use BDDs [4]. Because a single BDD representing the TR can be prohibitively large, techniques to represent the TR using a list of BDDs representing TF_i's, referred to as a *partitioned transition relation*, have been proposed [7]. Later, an automatic way of partitioning the TF_i's into groups and using the conjunction of TF_i's in each group, called a *cluster*, to reduce the number of partitions was developed [11]. In this paper, we represent the i^{th} cluster by TR_i. Then the TR can be implicitly represented using clusters as follows.

$$TR = \prod_{i=1}^{n} TR_i(x, y),$$

where n is the number of clusters.

The states, *To,* reachable from a set of states, *From,* in at most one step can be obtained from the BDD representing the TR by performing existential quantification and conjunction as follows.

$$To(y) = From(y) \cup \exists_{x_i \in x} [From(x) \cdot TR(x, y)],$$

where $\exists$ represents existential quantification. When a partitioned transition relation is used,

$$To(y) = From(y) \cup \exists_{x_i \in x} [From(x) \cdot TR_1(x, y) \cdot \ldots \cdot TR_n(x, y)].$$

Here, the conjunctions are performed iteratively, forming many *intermediate results*.

If there exist a set of variables that some clusters do not depend on, the existential quantification can be distributed over the partial products. We can quantify out such variables from the intermediate results before the entire multiplication is finished, leading to smaller intermediate result BDDs [6]. This technique is called *early quantification*.

3 Symbolic Traversal Using Don't Cares

The basic premise of this paper is that approximate reachability analysis can be used to quickly compute many states that are guaranteed to be unreachable from initial states. For example, Cho et al. [13] presented efficient approximate reachability analysis techniques that produce a superset of the reachable states. The complement of this set represents a subset of unreachable states. The key idea that any transition that originates from an unreachable state is a DC because it will never be explored during exact reachability analysis.

This section exploits these DCs to improve both approximate and exact reachability analysis. We first describe how to derive *clustered* DC BDDs from the results of approximate reachability analysis. We then describe how these clustered DCs can be used to improve approximate reachability analysis using an *iterative approach*, yielding larger DC sets. Finally, we describe how these DCs can be used to improve exact reachability analysis, introducing a new technique based on *support set minimization*.

3.1 Don't Care Transitions and Clustered Don't Care BDDs

The approximate reachability techniques known to us are based on state space decomposition [13, 14, 15]. The results of these analysis are typically in the form of a list of BDDs, $R^+_1, \ldots, R^+_n$, each of which corresponds to a superset of reachable states associated with the state space of each cluster. The conjunction of all R^+_i's yields the smallest superset of the reachable states associated with the entire state space. Thus, its complement yields the largest DC set. In practice, however, the conjuncted BDD can be prohibitively large. Alternatively, it is possible to individually apply BDD minimization to each TR_i (or each TF_i if the TF is minimized) using their respective R^+_i as the care set. However, this approach may lead to poor minimization because the DC fraction obtained from individual R^+_i may be very small. This suggests that we need to derive DC sets with reasonably high DC fractions and affordable BDD size.[1]

We propose a c*lustered DC* approach*,* in which we construct a BDD C_i representing the care set associated with TR_i by conjuncting R^+_j's which are likely to improve the minimization quality for TR_i. We identify R^+_j*'s* for each TR_i such that the support set of R^+_j includes at least one variable that is also contained in the support set of TR_i. Then we existentially quantify out the nonsupporting variables of TR_i from such R^+_j's to further reduce their size. This means that the support set of C_i is a subset of the support set of TR_i and consequently that C_i is typically much smaller than TR_i. Figure 1 shows the algorithm to build clustered-DCs.

Procedure **Build-clustered-DC** ($\{TR_i\}$, $\{R^+_j\}$)
  for i = 1 to m /* m : number of TR */
    C_i = bdd_one
    for j = 1 to n
      if support(TR_i) $\cap$ support(TR_j) $\neq \emptyset$ **then** $C_i = C_i \cdot (\exists_{x \notin TR_i} R^+_j)$
  return ($\{C_i\}$)

Figure 1: Algorithm to build clustered-DCs

We note that it is also possible to use the transitions *to* unreachable states as DC transitions during a traversal iteration. The problem here is that the incorrect states may be added during the traversal which must be filtered out at the end of each iteration, causing significant computational overhead. We refer the reader to [8] for more details.

3.2 Iterative Approximate FSM Traversal

The main reason for the computational efficiency of the approximate reachability analysis is that each sub-FSM rep-

[1] An analogy can be found in logic synthesis area where the satisfiability DC set of the whole network is too large for two-level minimizer [16] so a subset of the DC set is used instead.

resented by a cluster is traversed separately. The separated traversal, however, introduces some loss of information concerning the interaction among each sub-FSM. If we decompose an FSM into too many sub-FSMs, the information loss is increased leading to a coarse approximation. Consequently, there is a trade-off between the computational efficiency and the quality of the approximation and existing clustering techniques ensure that each cluster size is smaller than a predetermined threshold [13, 15].

We propose to improve the quality of the approximation using BDD minimization. In particular, we propose to derive DCs from one approximate reachability analysis, minimize the original TF_i's using these DCs, re-cluster these minimized TF_i's, and repeat approximate reachability analysis using the new clusters. Because the TF_i's are smaller, the re-clustering algorithm can lead to fewer clusters and consequently the approximation can be improved, leading to more DCs. The larger set of DCs often leads to better minimization of the TF_i's. Therefore, we propose to repeat this process until no significant improvement is made.

To quantify improvement we use the number of clusters as our cost function rather than the exact DC fraction, because the exact DC fraction (that requires the conjunction of all $R^+{}_i$'s when the state space of each $R^+{}_i$' is not mutually disjoint) is computationally expensive or infeasible to compute. Consequently, the process is completed when the number of clusters obtained is not reduced.

Another important factor determining the quality of the approximation is the effectiveness of the state decomposition. The basic idea of existing state decomposition techniques [22, 15] is to combine *closely related* TF_i's into one cluster where the closeness of the relationship between two TF_i's is estimated by examining the number of common variables in their support sets. BDD minimization can indirectly lead to better state decomposition because it sometimes removes some variables from the support sets of BDDs leading to a more accurate analysis of the relationships among TF_i's.

It is obvious that the choice of a BDD minimization heuristic is important. We use *restrict*[1] [4] because it is fast and is competitive in terms of both size minimization and support set minimization. For example, *constrain* [4] has the problem of introducing variables to the BDD to minimize from its care BDD as a side effect. Compaction algorithms [10] are typically better than *restrict* in terms of minimization power but worse in terms of support set reduction. This is because compaction algorithms apply a node minimization operation, called *sibling-substitution*, less aggressively than *restrict* does to prevent overall BDD size growth. There exists even more aggressive minimization heuristics presented by Shiple et al. [9] but they are prohibitively slow for large BDDs such as the TRs and clustered DC BDDs.

Finally, we note that reduced support sets of the sub-FSMs can also lead to faster run-times in the approximate FSM traversal. This is because if fewer sub-FSMs have support sets that intersect with the support set of a sub-FSM, fewer sub-FSMs need to be re-traversed after the sub-FSM is traversed [13]. This typically means that the total number of sub-FSM traversals needed is reduced, reducing overall run-time.

[1] To be precise, we use *thresholded restrict* [10] which returns the smaller BDD from the result of restrict and the original BDD.

Procedure **Iterative-approximate-traversal** ($\{TF_i\}$, I: initial states)
  $\{R^+{}_0\}$= {bdd_one}, $\{C_0\}$= {bdd_one}, *n_clusters* = infinity
  do {
    old_n_clusters = *n_clusters*
    $\{C_i\}$ = build-clustered-DC ($\{TF_i\}$, $\{R^+{}_j\}$)
    $\{TF_i\}$ = minimize-and-reorder-using-DC ($\{TF_i\}$, $\{C_i\}$)
    $\{TR_k\}$ = build-clusters ($\{TF_i\}$)
    $\{R^+{}_l\}$ = $\{R^+{}_l\}$ $\cup$ standard-approximate-traversal ($\{TR_k\}$, I)
    n_clusters = *number_of_clusters*($\{TR_k\}$)
  } **while** (*n_clusters* < *old_n_clusters*)
  return $\{R^+{}_l\}$

Figure 2: Algorithm for iterative approximate traversal

3.3 Exact FSM Traversal Using Support Set Minimization (SSM)

Using the DCs obtained from the above approximate reachability analysis, we propose to minimize the TRs used in the exact traversal. This has two effects. First, the smaller TR size can directly lead to reduced run-times because the TRs are heavily used. Second, BDD minimization can minimize the support set of each cluster which can lead to a better early quantification schedule. This reduces the maximum number of variables that an intermediate result can have, leading to reduced memory requirements. We first consider using *restrict* as the TR minimization algorithm and then explore a new heuristic specifically targeting support set minimization.

The choice of minimization algorithms is important. Ideally we want an algorithm that minimizes the support set while still reducing BDD size. Unfortunately, there is typically a trade-off between BDD size and support set size, both cannot be minimum simultaneously. Our experiences suggest that the more important factor is the support set size because the quality of the early quantification schedule often dictates the size of the systems that can be analyzed.

We first apply *restrict* because it heuristically minimizes the support set without increasing the BDD size and it is fast. Then, we apply a more expensive new heuristic that specifically tries to reduce the support set size.

Our heuristic is based on the notion of a *nonessential* variable. Given an incompletely specified function *ff* represented by f and c, where f is a cover of *ff* and c is the care-function of *ff*, we call a variable x nonessential if $f_x \oplus f_{\bar{x}} \subseteq \bar{c}_x + \bar{c}_{\bar{x}}$. A nonessential variable can be removed from the support set of the function by reassigning some function values on don't care points, e.g. $f' = f_x c_x + f_{\bar{x}} c_{\bar{x}}$ is a cover f whose support set does not include the nonessential variable x. Note that f' has the same function values of f on all care points [9]. Notice however that it may not be possible to simultaneously remove all nonessential variables from a cover because different nonessential variables may require conflicting DC assignments.

Our heuristic removes nonessential nodes greedily in each cluster in order from last cluster to first cluster. The first step is to compute the *benefit* of removing each supporting variable in a cluster. Let's consider TR consisting of n clusters, TR_1 to TR_n, and a non-essential variable x of a cluster TR_i. If x is in the support set of any cluster whose index is larger than i, that means removing x from TR_i will not improve early quantification schedule under current cluster ordering. In that case, the benefit of removing x is described as 0. However, if v is not in the support sets of all clusters with index larger than i, removing x from TR_i will allow x to be quantified out right after the last cluster TR_k with x in its support set is multiplied to intermediate results. The benefit of removing x is $i - k + 1$ in this case.

The second step is to actually remove non-essential variables following the order determined by the benefit computed in the previous step. We first check if the variable with the highest benefit is essential for the cluster. If it is nonessential, we remove the variable and move on to the next variable with highest benefit.

Note that removing a nonessential variable from a cluster implies that some DCs must be fixed to boolean values. Consequently, the care set of the cluster is expanded once a cover replaces the original function. In particular, when we remove a nonessential variable x from a cluster f, we produce a new cover $f' = f_x c_x + f_{\bar{x}} c_{\bar{x}}$ with the new care set $c' = c_x + c_{\bar{x}}$. With the updated cover and care set, we repeat the process of checking and removing the next best nonessential variable.

Note also that the size of the cover with reduced support set can be larger than the original cover. For a nonessential variable with benefit larger than 0, if the size of the cover is smaller than the maximum cluster size, we remove the variable because it leads to direct early quantification schedule improvement. If the size of the cover is larger than the maximum cluster size, we choose not to remove this variable and instead consider removing the next best nonessential variable. For a nonessential variable with benefit 0, we remove the variable only if the cover size is smaller than the original cluster because the benefit 0 means we cannot achieve direct early quantification schedule improvement. If the cover size is smaller, we remove the variable because it leads to smaller cluster as well as potential improvement on early quantification schedule under reordered clustering.

After all clusters are processed, we re-order the clusters to further improve the early quantification schedule. Thus, in this case the best choice of which nonessential variable to remove is not obvious and we are currently exploring alternative heuristics.

Figure 3 shows our algorithm which we call support set minimization (*SSM*) based exact reachability analysis.

Procedure **SSM-traversal** ($\{TF_i\}$, I)
  $\{R^+_j\}$ = iterative-approximate-traversal ($\{TF_i\}$, I)
  $\{TR_k\}$ = build-clusters ($\{TF_i\}$)
  $\{C_k\}$ = build-clustered-DC ($\{TR_k\}$, $\{R^+_j\}$)
  $\{TR_k\}$ = restrict ($\{TR_k\}$, $\{C_k\}$)
  reorder ($\{TR_k\}$)
  $\{TR_k\}$ = support-set-minimize ($\{TR_k\}$, $\{C_k\}$)
  reorder ($\{TR_k\}$)
  R = standard-exact-traversal ($\{TR_k\}$, I)
  return R

Figure 3: Algorithm for support set minimization (SSM) based exact reachability analysis

4 Experimental Results

We incorporated the new techniques into VIS-1.2 [11] using Long's BDD package [21]. We conducted experiments on the reachability analysis of large examples taken from ISCAS 89 and ISCAS-Addendum 93 benchmark circuits. All the experiments were conducted on a SUN UltraSPARC 3000 / 168MHz. The data-size limit was set to 128Mb. We used all default parameter setting provided by VIS including image_cluster_size of 5000 and the clustering heuristic described in [11] for both approximate and exact reachability analysis. The sifting algorithm [3] was used for dynamic variable ordering. Dynamic variable ordering was enabled at all times except during approximate reachability analysis. Because different variable ordering typically leads to different clusterings, dynamic variable ordering during iterative approximate traversal can yield better approximation results. For these experiments, however, we disabled it so that we can accurately estimate the contribution of BDD minimization.

The results from approximate reachability analysis are given in Table 1. We used Cho et al.'s MBM traversal [13] as the standard approximate traversal algorithm. The column $\boldsymbol{R^+}$ represents the percentage of approximate reachable states over the entire state space, ***Time*** denotes the run-time in CPU seconds, and ***# Iter*** reports the number of standard approximate traversals our approach performed. In one case, we could not compute the R^+ fraction exactly because R^+ was too large to build. In this case, denoted with a *, we used an upper bound of the approximate reachable states computed using the method described in [15]. The last column describes the improvement factor by showing the ratio of the R^+ obtained by the standard approximate traversal over the R^+ obtained by our iterative approximate traversal.

The results show that our iterative approach produces significantly better results in terms of approximation quality with minor run-time overhead (two times slower than standard traversal in the worst case). Note that the run-times for the approximate traversals are still very small fraction of typical run-time needed for an exact traversal of a large FSM.

Circuit	*MBM*		*Iterative*			*MBM / Iterative*
	R^+ (%)	*Time*	*R^+ (%)*	*Time*	*# Iter*	
s1423	0.608	202	0.424	225	3	1.434
s3271	5.32	115	4.520	142	3	1.178
s3330	7.147	119	1.98e-3	140	4	3.61e3
s4863	1.16e-3	490	2.25e-4	548	4	5.167
s5378	4.33e-9	261	4.31e-8*	519	4	> 10.00

Table 1: Comparison between standard and iterative approximate traversals. (* indicates an upper bound)

Before we show the results from exact reachability analysis, we report the size of TR and the number of current state variables from the TR used in each traversal in Table 2. The column *VIS* represents the standard exact traversal and SSM represents our new traversal using both *restrict* and our support set minimization heuristic. The parenthesized number shows the data from the traversal using *restrict only*.

The results show that the TRs used in SSM are significantly smaller in size and have fewer current state variables than the TRs used in the standard traversal. The reduction of current state variables is largest for the three biggest examples whose DC fractions are much higher than those for the smaller examples.

Circuit	*TR Size*			*# Current State Vars*		
	VIS	*SSM (restrict only)*		*VIS*	*SSM (restrict only)*	
s1423	20,039	12,277	(12,277)	237	235	(235)
s3271	14,305	11,654	(11,654)	252	247	(247)
s3330	18,099	19,353	(17,797)	167	162	(167)
s4863	125,206	35,001	(35,344)	508	395	(417)
s5378	43,987	19,141	(18,131)	397	347	(377)

Table 2: Comparison of TRs used in each traversal method.

The results from exact reachability analysis are given in Table 3. The column denoted ***# Level*** reports the number of iterations and ***** symbol indicates the entire traversal successfully completed. The last three columns report the runtimes taken only for exact traversals, i.e., excluding clustering and approximate traversal.

The results show that SSM outperforms standard traversal in all aspects. SSM traversal successfully completes the entire traversal for s3271 while the standard traversal stops after 7 of 17 iterations. The SSM traversal runs out of memory after 7 iterations (6 iterations using *restrict* only) of s5378 while the standard traversal runs out of memory after 4. For all examples, SSM takes significantly less time than the standard traversal to complete the same number of iterations.

Notice that for the examples s3330 and s5378 SSM with support set minimization yields significantly better results than SSM using the "restrict only" method even though the support set minimization increased the BDD sizes significantly. This suggests that the impact of support set minimization on the early quantification schedule may be more critical than the BDD sizes in determining the overall runtime and memory requirements.

Circuit	*#Level*	*# States*	*Traversal Time*		
			VIS	*SSM*	*(restrict only)*
s1423	11	7.991e11	27,020	22,424	(22,420)
s3271	7	4.129e22	5,873	5,269	(5,273)
	17*	1.318e31	space out	13,070	(13,065)
s3330	9*	7.278e17	18,002	10,134	(16,811)
s4863	5*	2.191e19	50,201	7,120	(8,387)
s5378	4	2.393e13	17,802	7,900	(8,371)
	6	2.470e20	space out	10,040	(11,020)
	7	1.165e21	space out	19,040	(space out)

Table 3: Comparison on traversal results. (* indicates complete traversal.)

5 Conclusion

In this paper, we have presented techniques to improve approximate and exact reachability analysis using DCs. The basis of the techniques is that approximated reachable states can be used as DCs to simplify the state transition representation of the FSM to be analyzed.

We have demonstrated the effectiveness of the techniques on a standard reachability analysis framework. The experimental results show that, compared to the traditional approach, our techniques can significantly reduce the reachability analysis run-time for large FSMs and explore more states.

A salient feature of our techniques is that they are orthogonal to most existing reachability analysis algorithms. We believe that our techniques can further improve the capability of recently proposed reachability analysis algorithms that use novel state decomposition techniques [17, 18, 19].

Acknowledgments

The authors would like to thank Amit Narayan for his prompt explanation on VIS experimental setup and Hyunwoo Cho for his comments on our work.

References

[1] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers*, vol. C-35, pp. 677-691, 1986.

[2] A. Aziz, S. Tasiran, and R. Brayton, "BDD Variable Ordering for Interacting Finite State Machines," in *Proc. ACM/IEEE Design Automation Conference*, pp. 283-288, 1994.

[3] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams," in *Proc. IEEE International Conference on Computer-Aided Design*, pp.42-47, 1993.

[4] O. Coudert, C. Berthet, and J. C. Madre, "Verification of Synchronous Sequential Machines Based on Symbolic Execution," *Automatic Verification Methods for Finite State systems*, Springer-Verlag, pp. 365-373, 1989.

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

[6] J. R. Burch, E. M. Clarke, D. Long, K. L. McMillan, and D. L. Dill, "Sequential Circuit Verification Using Symbolic Model Checking," in *Proc. ACM/IEEE Design Automation Conference*, pp. 46-51, 1990.

[7] J. R. Burch, E. M. Clarke, and D. E. Long, "Representing Circuits More Efficiently in Symbolic Model Checking," in *Proc. ACM/IEEE Design Automation Conference*, pp. 403-407, 1991.

[8] Y. Hong and P. A. Beerel, "Efficient Reachability Analysis of Large Finite State Machines Using Don't Care-Based BDD Minimization," Technical Report CENG 97-24, University of Southern California, Dec., 1997.

[9] T. Shiple, R. Hojati, A. Sangiovanni-Vincentelli, and R. K. Brayton, "Heuristic Minimization of BDDs Using

Don't Cares," in *Proc. ACM/IEEE Design Automation Conference*, pp. 225-231, 1994.

[10] Y. Hong, P. A. Beerel, J. R. Burch, and K. L. McMillan, "Safe BDD Minimization Using Don't Cares," in *Proc. ACM/IEEE Design Automation Conference*, pp. 208-213, 1997.

[11] R. K. Ranjan, A. Aziz, R. K. Brayton, B. Plessier and C. Pixely, "Efficient Formal Design Verification: Data Structure + Algorithms", Technical Report UCB/ERL M94, University of California, Berkeley, Oct., 1994.

[12] R. K. Brayton et. al., "VIS: A System for Verification and Synthesis," in *Proc. International Conference on Computer-Aided Verification*, pp.428-432, July, 1996.

[13] H. Cho, G. D. Hachtel, E. Macii, B. Plessier, and F. Somenzi, "Algorithms for Approximate FSM Traversal," in *Proc. ACM/IEEE Design Automation Conference*, pp. 25-30, 1993.

[14] W. Lee, A. Pardo, J. Jang, G. Hachtel, and F. Somenzi, "Tearing Based Automatic Abstraction for CTL Model Checking," in *Proc. IEEE International Conference on Computer-Aided Design*, pp.76-81, 1996.

[15] S. G. Govindaraju, D. L. Dill, A. J. Hu, and M. A. Horowitz, "Approximate Reachability with BDDs Using Overlapping Projections", to appear in *Proc. ACM/IEEE Design Automation Conference*, 1998.

[16] H. Savoj, R. K. Brayton, and H. J. Touati, "Extracting Local Don't Cares for Network Optimization," in *Proc. IEEE International Conference on Computer-Aided Design*, pp. 514-517, 1991.

[17] K. Ravi and F. Somenzi, "High-Density Reachability Analysis," in *Proc. IEEE International Conference on Computer-Aided Design*, pp. 154-158, 1995.

[18] G. Cabodi, P. Camurati, and S. Quer, "Improved Reachability Analysis of Large Finite State Machines," in *Proc. IEEE International Conference on Computer-Aided Design*, pp. 354 - 360, 1996.

[19] A. Narayan, A. J. Isles, J. Jain, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Reachability Analysis Using Partitioned-ROBDDs", in *Proc. IEEE International Conference on Computer-Aided Design*, pp. 388 - 393, 1997.

[20] G. Cabodi, P. Camurati, L. Lavagno, and S. Quer, "Disjunctive Partitioning and Partial Iterative Squaring," in *Proc. ACM/IEEE Design Automation Conference*, pp. 728-733, 1997.

[21] D. E. Long, A Binary Decision Diagram (BDD) Package, June 1993, Manual Page.

[22] H. Cho, G. D. Hachtel, E. Macii, M. Poncino, and F. Somenzi, "A Structural Approach to State Decomposition for Approximate Reachability Analysis," in *Proc. IEEE International Conference on Computer Design*, pp. 236-239, 1994.