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

Youpyo Hong
Synopsys, Inc
19500 Gibbs Dr. Hillsboro, OR 97006

Peter A. Beerel
Department of Electrical Engineering - Systems
University of Southern California, LA, CA 90089

## Abstract

*Reachability analysis of finite state machines is essential to many computer-aided design applications. We present new 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 in subsequent iterations for better approximation. Second, we propose new techniques to use the final approximation to enhance the capability and efficiency of exact reachability analysis. Experimental results show that the new techniques can improve reachability analysis significantly.*

## 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 consists of BDD variable ordering heuristics [2, 3] because the size of a BDD depends greatly on the ordering of the BDD variables [1].

Numerous other techniques to analyze large FSMs have been developed. 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 at the expense of more 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 for FSMs with high sequential depth.

The run-time performance of reachability analysis may also be improved by using don't cares (DCs) [4,11]. Coudert et al. [4] proposed to reduce the *frontier set* BDD using reached states, i.e. reachable states computed thus far, as DCs. Brayton et al. [11] presented techniques to reduce the size of the *transition relation BDDs* (*TR*) using transitions originating from non-frontier states and transitions to reached states as DCs. The major disadvantage of these approaches is that the minimization needs to be repeated in each iteration because the reached states, and consequently the DC sets, are updated at each iteration.

Ranjan et al. [10] discussed techniques to reduce the size of TRs using DCs derived from overapproximated reachable states [12]. 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. Hong et al. first reported successful results based on TR minimization in [23], a preliminary version of this paper. Recently, Moon et al. [24] showed that TR minimization using approximate reachable states can improve CTL model checking.

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.
- An *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.

After describing background in Section 2, we present the new techniques in Section 3. Section 4 reports our experimental results and Section 5 presents some conclusions.

## 2. Background

The next state function $\delta$ for a FSM with $n$ state bits consists of $\delta_i$'s for $1 \le i \le n$, referred to as a *transition function vector* [4], where $\delta_i$ 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.

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

where $x$ represents present state and input variables, $y$ represents next state variables, and $TF_i(x, y)$ represents $y_i \equiv \delta_i(x)$.

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]. 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, denoted $\exists$, and conjunction as follows.

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

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 quantify out such variables from the intermediate results before the entire multiplication is finished to reduce the size of intermediate result BDDs [6]. This technique is called *early quantification*.

## 3. Symbolic Traversal Using Don't Cares

Because the size and complexity of the TR greatly affects the run-time performance and capacity of symbolic FSM traversal, techniques to simplify of the TR can be very useful.

An effective way of simplifying a BDD is to utilize DC information, i.e. the flexibility of the function that the BDD represents. For the BDD representing the TR, any transition originating from an unreachable state is a DC for exact reachability analysis because such a transition will never be explored during the analysis. Although the complete set of unreachable states cannot be available until exact analysis is completed, a subset of unreachable states can be computed with significantly less computational resources using approximate traversal techniques [12]. Thus, transitions from this subset of states can be a computationally cheap source of DCs for the TR used in exact reachability analysis. Moreover, we will show how these DCs can be used to improve approximate reachability analysis.

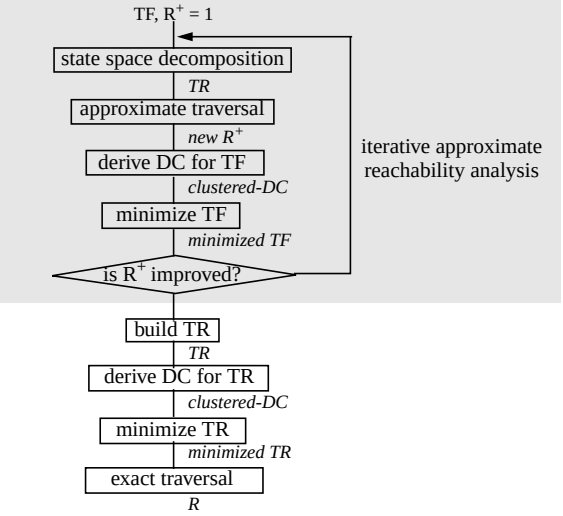


Figure 1: Flow of our approach

The overall flow of our approach is shown in Fig. 1. We first run coarse approximate reachability analysis and derive *clustered DCs* from the approximation results. Note that this clustered DC approach is applicable to both approximate and exact analysis. We then use these clustered DCs to improve approximate reachability analysis using an *iterative approach*, yielding smaller overapproximated reachable state set. Finally, we derive clustered DCs from the final approximation and utilize them to improve exact reachability analysis.

### 3.1. DC Transitions and Clustered DC BDDs

Currently known approximate reachability techniques are based on state space decomposition [12, 13, 14]. They produce a superset of reachable states associated with the state space of each cluster in the form of a list of BDDs, $R^+_1, \ldots, R^+_n$. The conjunction of all $R^+_i$'s yields the smallest superset of the reachable states for the entire state space. Thus, its complement yields the largest DC set using the overapproximated reachable states. 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 2 shows the algorithm to build clustered-DCs.

Procedure **Build-clustered-DC** ($\{TR_i\}$, $\{R^+_j\}$)
  **for** $i$ = 1 to $m$ /* $m$ : number of $TR_i$*'s* */
    $C_i$ = bdd_one
    **for** $j$ = 1 to $n$
      **if** support($TR_i$) $\cap$ support($R_j$) $\neq \emptyset$ **then** $C_i = C_i \cdot (\underset{x \notin TR_i}{\exists} R^+_j)$
  **return** ($\{\overline{C}_i\}$)

Figure 2: Algorithm to build clustered-DCs

## 3.2. Iterative Approximate FSM Traversal

The main reason for the computational efficiency of the approximate reachability analysis is that each sub-FSM represented by a cluster is traversed separately. The separated traversal, however, introduces some loss of information concerning the interaction among sub-FSMs. If we decompose a 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 [12, 14].

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 the 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 now 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 currently use the number of clusters as a simple cost function rather than the exact DC fraction because the exact DC fraction (which requires the conjunction of all $R^+_i$'s when the state space of each $R^+_i$' is not mutually disjoint) is computationally expensive if not infeasible. 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, 14] 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* [4] because it is fast and is competitive in terms of both size minimization and support set minimization. Compaction algorithms [9] 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 aggressive variants of *restrict* presented by Shiple et al. [8] but they are prohibitively slow for large BDDs such as the TRs.

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 [12]. This typically means that the total number of sub-FSM traversals needed is reduced, reducing overall run-time.

## 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

[1] Analogies can be found in the logic synthesis area [15, 16]. For example, the satisfiability DC set of the whole network is too large for a two-level minimizer [15] so a subset of the DC set is used instead.

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 $f_x \oplus f_{\bar{x}} \subseteq \bar{c}_x + \bar{c}_{\bar{x}}$. Equivalently, *x* is a nonessential variable of *ff* if and only if there exists a cover of *ff* that does not depend on *x*. For example, consider an incompletely specified function denoted by $f = x\bar{y}$ and $c = \bar{x}\bar{y} + x\bar{y}$, i.e. $\overline{xy}$ and $xy$ are DC minterms. Since $f_x \oplus f_{\bar{x}} = \bar{y} \subseteq \bar{c}_x + \bar{c}_{\bar{x}} = 1$, *x* is a nonessential variable. This makes sense because, if we let $f(\overline{xy}) = 1$ and $f(xy) = 0$, we obtain a cover $f' = \bar{y}$ which does not depend on *x*.

Note that, in general, there can be more than one cover that does not depend on a particular nonessential variable. In this work, we use $f' = f_x c_x + f_{\bar{x}} c_{\bar{x}}$ which was shown to be a cover of *f* in [8] and clearly does not depend on *x*.

An incompletely specified function may have more than one nonessential variable. Unfortunately, it may not be possible to simultaneously remove all nonessential variables from its cover because different nonessential variables may require conflicting DC assignments. For the above example, notice that *y* is also a nonessential variable because $f_y \oplus f_{\bar{y}} = x \subseteq \bar{c}_y + \bar{c}_{\bar{y}} = 1$. The only cover that does not depend on *y* is $f' = x$ which is derived by assigning $f(\overline{xy}) = 0$ and $f(xy) = 1$. Notice that because the required don't care assignments are in conflict, there does not exist any cover that is independent of both *x* and *y*. This means that we must choose which nonessential variable to remove when multiple nonessential variables cannot be simultaneously removed.

Our heuristic resolves this choice by removing nonessential nodes one by one in a greedy fashion based on a priority function that reflects the number of intermediate results from which the nonessential variable is removed.

First, we compute the *benefit* of removing *every* supporting variable in a cluster under the assumption that the variable is nonessential. Let's consider a *TR* consisting of *n* clusters, $TR_1$ to $TR_n$, and a variable *x* of cluster $TR_i$. If *x* is in the support set of any cluster whose index is larger than *i*, removing *x* from $TR_i$ will not improve the early quantification schedule (under the current cluster ordering). In other words, the number of intermediate results containing *x* does not change and the benefit of removing *x* is thus defined as 0. On the other hand, if *x* is not in the support set of any cluster with index larger than *i*, removing *x* from $TR_i$ will allow *x* to be quantified out earlier than specified by the original early quantification schedule. More precisely, *x* can now be quantified out immediately after the intermediate result is multiplied with the last cluster that has *x* in its support set (not including the *i*-th cluster). Let this cluster be $TR_k$ and notice that *k* must be less than *i*. The benefit of removing *x*, i.e., the number of intermediate results from which the variable x is removed, is then *i* - *k*.

The second step is to check and remove non-essential variables following their priority. Notice that the benefit of removing a nonessential variable depends on the support set of later clusters. To ensure that this benefit is not changed by subsequent nonessential variable removal, the clusters are processed from last to first. More specifically, all nonessential variables that can be greedily removed from one cluster are removed before the next earlier cluster is processed. After all clusters are processed, we re-order the clusters to further improve the early quantification schedule.

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 continue the process of checking and removing the next best nonessential variable.

The fact that the DC set changes after removing one nonessential variable is the reason why we chose to compute the benefits of every variable before checking to see whether any variable is essential. In particular, this order of operations avoids an initial relatively expensive calculation of essentiality of a lower priority variable using a DC set that may be invalidated by the removal of a higher priority essential variable.

Note also that the size of the cover with reduced support set can be larger than the original cover. We only remove a nonessential variable if the size of the cover is smaller than the maximum cluster size. Lastly, note that we also remove a non-essential variable with benefit 0 as long as it leads to a smaller cluster.

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 SSM based exact FSM traversal

### 3.4. Applicability of Our Techniques to Existing Exact Traversal Algorithms

A salient feature of our *approximate reachable states based TR minimization technique* is that it is applicable to most existing TR-based exact reachability analysis algorithms. We explain how to apply our techniques to minimize various types of transition relations, i.e. monolithic TR, conjunctive partitioned TR, and disjunctive partitioned TR, and discuss their impacts.

The application of DC-based minimization to monolithic TR is straightforward. The support set minimization is not helpful in this case because no early quantification is used in monolithic TR based traversal. Yet we can still expect performance improvement due to the smaller TR size.

The conjunctive partitioned TR based traversal algorithms form a major category in reachability analysis and many state-of-the-art traversal algorithms [17, 18, 19] have been developed. A common idea of these algorithms is to extract subsets of the state sets that are representable with small BDDs and use the subsets as frontier sets. We may store the clusters on hard disk to better utilize main memory but the functionality of TR is not modified. Therefore our DC-based BDD minimization techniques can be applied to conjunctive partitioned TR without any modification.

We also note that the method proposed by Touati et al. [5] using an implicit conjunction of $TF_i$'s to represent the TR can also be improved using our techniques. They first multiply the frontier set with each $TF_i$ and conjunct the results to form the final results in a balanced binary tree fashion. During this process, they existentially quantify out all variables from a partial product when all the BDDs that depend on the variables are combined into the partial product. We can derive the clustered-DC for each partial product and use it to minimize the support set size of the partial product in order to maximize the number of variables that we can existentially quantify out.

Lastly, we consider Cabodi et al.'s work in which the disjunctive TR, originally introduced to model asynchronous circuits [7], is extended to model synchronous circuits [20]. The basic idea is to represent the TR by the list $TR{\cdot}p_1, \ldots, TR{\cdot}p_k$ implicitly disjuncted, i.e. $TR = TR{\cdot}p_1 + \ldots + TR{\cdot}p_k$, with $\Sigma\ p_i = 1$. During their reachable state computation, existential quantification is performed independently for each partitioned TR. Consequently, support set minimization of each partitioned TR may not be useful. However, since the size of each partition is still a major concern, our DC-based minimization technique can still improve its performance.

## 4. Experimental Results

We incorporated the new techniques into VIS-1.2 [10] using Long's BDD package [21]. We conducted experiments on the reachability analysis of ISCAS 89 and ISCAS-Addendum 93 benchmark circuits using a SUN UltraSPARC with 168MHz clock. 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 [10] for both approximate and exact reachability analysis. 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 [12] 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 [14]. 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 fractions of the 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 | 4.31e-8 | 261 | 4.33e-9* | 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 exam-

ples 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 run times 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 run-time 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: Exact traversal results comparison. (* 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.

Because our approach is orthogonal to most existing reachability analysis algorithms, our new techniques can further improve the capability of recently proposed reachability analysis algorithms.


### Acknowledgments

The authors would like to thank A Narayan for his help on our experimental setup and H. Cho, F. Somenzi, and G. Hachtel for constructive 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," *Proc. DAC*, pp. 283-288, 1994.

[3] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams," *Proc. ICCAD*, 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," *Proc. ICCAD*, 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," *Proc. DAC*, pp. 46-51, 1990.

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

[8] T. Shiple, R. Hojati, A. Sangiovanni-Vincentelli, and R. K. Brayton, "Heuristic Minimization of BDDs Using Don't Cares," *Proc. DAC*, pp. 225-231, 1994.

[9] Y. Hong, P. A. Beerel, J. R. Burch, and K. L. McMillan, "Safe BDD Minimization Using Don't Cares," *Proc. DAC*, pp.208-213, 1997.

[10] 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.

[11] R. K. Brayton et. al., “VIS: A System for Verification and Synthesis,” *Proc. Int’l Conference on CAV*, pp. 428-432, July, 1996.

[12] H. Cho, G. D. Hachtel, E. Macii, B. Plessier, and F. Somenzi, “Algorithms for Approximate FSM Traversal,” *Proc. DAC*, pp. 25-30, 1993.

[13] W. Lee, A. Pardo, J. Jang, G. Hachtel, and F. Somenzi, “Tearing Based Automatic Abstraction for CTL Model Checking,” *Proc. ICCAD*, pp.76-81, 1996.

[14] S. G. Govindaraju, D. L. Dill, A. J. Hu, and M. A. Horowitz, “Approximate Reachability with BDDs Using Overlapping Projections”, *Proc. DAC*, pp. 451-456,1998.

[15] H. Savoj, R. K. Brayton, and H. J. Touati, “Extracting Local Don’t Cares for Network Optimization,” in *Proc. ICCAD*, pp. 514-517, 1991.

[16] D. Brand, R. A. Bergamaschi, and L. Stok, “Be Careful with Don’t Cares,” *Proc. ICCAD*, pp. 83-86, 1995.

[17] K. Ravi and F. Somenzi, “High-Density Reachability Analysis,” *Proc. ICCAD*, pp. 154-158, 1995.

[18] G. Cabodi, P. Camurati, and S. Quer, “Improved Reachability Analysis of Large Finite State Machines,” *Proc. ICCAD*, pp. 354 - 360, 1996.

[19] A. Narayan, A. J. Isles, J. Jain, R. K. Brayton, and A. Sangiovanni-Vincentelli, “Reachability Analysis Using Partitioned-ROBDDs”, *Proc. ICCAD*, pp. 388-393, 1997.

[20] G. Cabodi, P. Camurati, L. Lavagno, and S. Quer, “Disjunctive Partitioning and Partial Iterative Squaring,” *Proc. DAC*, 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,” *Proc. ICCAD*, pp. 236-239, 1994.

[23] Y. Hong and Peter A. Beerel, “Symbolic Reachability Analsying of Large Finite State Machines Using Don’t Cares,” IWLS98, 1998.

[24] I. Moon, J. Jang, G. Hachtel, F. Somenzi, J. Yuan, C. Pixley, “Approximate Reachability Analysis Don’t Cares for CTL Model Checking,” *Proc. ICCAD*, pp. 351-358, 1998.