Efficient Reachability Analysis of Large Finite State Machines Using Don't Care-Based BDD Minimization

Youpyo Hong and Peter A. Beerel

CENG 97-25

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213) 740-4481
December 1997

Efficient Reachability Analysis of Large Finite State Machines

Using Don't Care-Based BDD Minimization

Youpyo Hong and Peter A. Beerel

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California

Abstract

Reachability analysis of finite state machines is essential to many computer-aided design applications. This paper presents techniques to improve the efficiency of symbolic reachability analysis using don't care-based BDD minimization. We use approximate reachability analysis to derive don't care sets and use the don't care sets to reduce the size of the transition relation BDDs which are intensively used during exact reachability analysis. Our techniques can be easily incorporated into most existing reachability analysis frameworks because our approaches are orthogonal to others. Experimental results show that our techniques can lead to more than an order of magnitude runtime improvement for large examples when compared to traditional reachability analysis.

Keywords: Binary Decision Diagram, Don't Care, Formal Verification, Reachability Analysis

1 Introduction

Many algorithms used in computer-aided design, e.g., equivalence checking and model checking, are closely related to the problem of reachability analysis. Symbolic reachability analysis techniques using binary decision diagrams (BDD's) [1] were introduced by Coudert et al. [7], 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 [7, 9, 10]. 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, e.g., BDDs representing reachable states or intermediate results. Because for large FSMs these latter BDDs tend to be much larger than the transition relation BDDs, techniques to dynamically reorder the variables (e.g. [5]) are often used[17, 20, 21]. Although dynamic variable ordering can be time consuming, it can allow larger FSMs to be analyzed and can sometimes improve the overall reachability analysis runtime by dramatically reducing BDD sizes [17].

Numerous other techniques to analyze large FSMs have been developed. For example, Ravi et al. [20] 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 typically requires more iterations to complete reachability analysis. In addition, Cabodi et al. [21] proposed to temporarily simplify FSMs by latch removal and reinsert the removed latches at the end of each breadth-first step. Recently, Cabodi et al. [22] presented an approach that extends the application of disjunctive partitioned transition relations from asynchronous cir-

cuits to synchronous ones and utilizes iterative squaring. Their approach is effective when the FSM requires many iterations.

The performance of the reachability analysis may also be improved by applying BDD minimization using don't cares (DCs) [7,18] to reduce the size of the BDDs. A number of different BDDs and DCs have been explored.

For example, Coudert et al. [7] proposed to reduce the *frontier set* BDD which is used as a state space search frontier at each iteration. They proposed using the reachable states found in previous iterations as DCs because all transitions from these states have already been explored. This technique, called *frontier-set simplification*, can lead to significant runtime improvements with insignificant increase in memory usage [12].

In addition, Brayton et al. [18] presented techniques to reduce the size of the *transition relation (TR) BDD* which represent the possible state transitions of an FSM. 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 because the DC set changes in each iteration.

Moreover, Ranjan et al. [17] discussed techniques to reduce the size of TR BDDs using conservative approximations of the unreachable states [19] to derive DCs. 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 thus did not lead to a performance improvement.

In this paper, we also simplify the TR BDDs using DCs derived from approximate reachability analysis with more promising results than described in [17]. The new features of our approach include the following.

- The incorporation of a larger DC set obtained from approximate reachability analysis than used by Ranjan et al. in [17].

- The use of *clustered DC BDDs* to minimize the size of DC BDDs without significantly sacrificing BDD minimization quality, thereby speeding up minimization.
- The use of *source-less states* to efficiently derive a smaller subset of unreachable states when approximate reachability analysis is too time-consuming.
- The inclusion of options to trade off the quality and the runtime of approximate reachability analysis.
- The inclusion of options to trade off the overhead due to DC-based BDD minimization and speed-up of reachability analysis.

We conducted experiments on a variety of large examples taken from ISCAS 89 and ISCAS-ADDENDUM 93 benchmarks. Our results show that we can achieve as much as an order of magnitude runtime improvement using DC-based TR BDD minimization.

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

2 Background

In this section, we overview the four basic components of our approach: standard (exact) symbolic reachability analysis, approximate reachability analysis, BDD minimization using DCs, and dynamic variable ordering.

2.1 Symbolic Reachability Analysis of Finite State Machines

This section reviews the definitions in symbolic reachability analysis needed in this paper. We refer the reader to [7, 11, 17] for more detailed information.

A sequential circuits can be described by a finite state machine (FSM) which is a 6-tuple (I, O, S, S^0, δ, λ), where I is the input space, O is the output space, S is the set of states, S^0 is the set of initial states, δ is the next state function ($\delta : I \times S \rightarrow S$), and λ is the output function ($\lambda : I \times$

$S \rightarrow O$). In this paper, we consider synchronous sequential circuits and we omit the standard input/output specification of the FSM description to both simplify the notation and 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. A variable x_i in x has a corresponding variable y_i in y and vice versa. The next state function δ for a FSM with n state bits consists of δ_i's for $1 \leq i \leq n$, referred to as a *transition function vector* [7], where δ_i represents the next state logic for i^{th} state bit. A *transition relation* [7], denoted by *TR,* defines all possible current/next state pairs. The transition relation for a synchronous FSM is defined as the conjunction of the transition function vectors as follows.

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

To represent the transition relations, BDDs are used in symbolic approach [7]. The size of the BDD representing the *TR* can be prohibitively large when the number of state bits required to represent the FSM is large. Therefore, techniques to represent the transition relation using a list of t_i's of (1), referred to as a *conjunctive partitioned transition relation*, have been proposed [11]. Later, an automatic way of partitioning the t_i's into groups and using the conjunction of t_i's in each group, called *clusters*, to reduce the number of partitions was developed [17]. 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 transition relation by performing existential quantification and conjunction as follows.

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

1. $a \equiv b = a \cdot b + \bar{a} \cdot \bar{b}$

where $\exists$ represents existential quantification of the variables below the symbol. Note that $From(x)$ and $From(y)$ represent the same set of states but in terms of current state variables and next state variables, respectively. When a partitioned transition relation is used,

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

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. As a result, we can quantify out such variables from the partial product before the entire multiplication is finished, leading to smaller intermediate result BDDs [12]. For example, consider a set of present state variable $x = \{x_1, x_2\}$. If x_1 is not a supporting variable for TR_2, we can quantify out the variable x_1 early as follows.

$$To(y) = From(y) \cup \underset{x_2}{\exists} [\underset{x_1}{\exists} [From(x) \cdot TR_1(x, y)] \cdot TR_2(x, y)].$$

Each such calculation represents one traversal step and is called an *iteration*. The *To* set computed in one iteration becomes the *From* set in the following iteration. Starting with a *From* set that represents the initial states and iterating until a fixed point is reached yields a *To* set that represents all reachable states. This entire process is called *reachability analysis*.

2.2 Approximate Reachability Analysis

Cho et al. [19] presented various approximate reachability analysis techniques that are guaranteed to produce a superset of the reachable states. They concluded that the most robust technique for very large FSMs is their *machine-by-machine* (*MBM*) *traversal* technique.

The MBM traversal technique starts by decomposing the state space by partitioning the set of next state variables. It then take the product of the transition function vectors associated with the next state variables belonging to the same partition to form a cluster. Note that many conventional reachability analysis tools, e.g. [18], are also based on using clusters although they have different clustering criteria.

The key idea to the MBM traversal technique is that it views a cluster as a sub-FSM by treating the current state variables that do not have corresponding next state variables in the cluster as external input variables. Consequently, each sub-FSM can be traversed separately to compute its set of reachable states in its corresponding state sub-space by conjuncting the known reachable states from other sub-FSMs to identify the possible combinations of external inputs. More specifically, the set of possible combinations of external inputs are assumed to be identical in all sub-FSM states. For this reason, the approach may significantly overestimate the possible values of external inputs in each state of the machine, consequently overestimating reachable states of the FSM.

The algorithm starts by assuming that all states of all sub-FSMs are reachable and all sub-FSMs are scheduled for traversal. Anytime the reached state of a sub-FSM changes, all the other sub-FSMs that depend on it are re-scheduled for traversal because their external inputs may now be further constrained. Each scheduled sub-FSM is processed until a fixed point in the computation of the reached set is obtained.

The approximate reachability analysis can handle much larger FSMs than feasible using the exact approach because the reachable states associated with each sub-FSM are maintained separately in the approximated reachability analysis while the exact approach uses a single BDD to represent the reachable states.

2.3 BDD Minimization Using Don't Cares

A Boolean function can be described as the set of all input points, i.e. minterms, for which the function evaluates to 1, referred to as the *on-set* of f. Similarly, the *off-set* of f is the set of all input points for which the function evaluates to 0. If we do not care if the function evaluates to a value 0 or 1 for a set of input points, the function is said *incompletely specified*, and such input points are called *don't care-set*. Therefore, the domain of any single-output Boolean function can be partitioned into three subsets, the on-set, the off-set, and the don't care-set. An incompletely specified function can be represented by a pair of functions $[f, c]$ where $f \cdot c$ denotes the on-set, and $\bar{f} \cdot c$

denotes the off-set, and c denotes the care-set of the function. A function g is a *cover* of $[f, c]$ if f and g have the same function values for all care minterms, i.e., $f \cdot c = g \cdot c$ and $\bar{f} \cdot c = \bar{g} \cdot c$. Given f and c, finding the cover of $[f, c]$ with smallest BDD representation is called BDD minimization using DCs. Because the problem is known to be NP-complete [6], heuristics for DC-based BDD minimization are typically used [7, 13, 14, 15, 16]. For example, the *constrain* operator is a BDD minimization algorithm that produces a minimum sized BDD when c is a cube [14]. *Restrict* is a similar BDD minimization algorithm that typically outperforms *constrain* by avoiding the problem of introducing nodes associated with non-supporting variables into the minimized BDD. *Basic-compaction* and *LI-compaction* [16] are BDD minimization algorithms which are guaranteed to never increase the BDD size and are very useful when the DC fraction is relatively small. Drechsler et al. [15] proposed evolutionary algorithm-based BDD minimization algorithms.

2.4 Dynamic Variable Ordering

Numerous dynamic variable ordering techniques have been developed [3, 4, 5]. One of the most effective and widely used technique is *sifting* [5]. The basic idea of sifting is to move each variable throughout the order to find its optimal position assuming the position of the other variables are fixed.

Dynamic variable ordering can be invoked either when the user explicitly calls it or when the number of BDD nodes in use reaches a dynamically set limit, which, for example, may be doubled each time dynamic variable ordering is invoked.

The simplest form of dynamic variable ordering minimizes the total number of BDD nodes needed. Dynamic variable ordering, however, can be targeted towards a subset of the BDDs used. This can lead to a significant overall improvement when only a few BDDs are critical to the quality of the result.

3 Improving Symbolic Traversal

This section develops techniques to improve the efficiency of exact reachability analysis using

DCs derived from the approximate reachability analysis. First, we explain a simple *thresholding* technique to keep the run-time of the approximate analysis reasonably low. Secondly, we describe which DC transitions we use to simplify the transition relation BDDs. Then, we address practical issues on managing DC BDDs and performing BDD minimization, including using *clustered DC BDDs*. Lastly, we explain how to efficiently use BDD minimization using DCs with and without dynamic variable ordering, respectively.

3.1 Thresholded Approximate Reachability Analysis

The main reason for the success of the approximate reachability is that the size of each sub-FSM can be made small. However, the approximate reachability analysis can be time-consuming if some sub-FSMs require many iterations to compute their reachable states. For this reason, we propose to impose a threshold on the number of iterations allowed for each sub-FSM traversal. That is, when the number of iteration reaches the threshold value, we abandon the MBM approximate reachability analysis of this sub-FSM.

When an MBM traversal is abandoned for a particular sub-FSM, we propose to use a coarser estimate of the reachable states which is much simpler to obtain. Our idea is based on the notion of *source-less states* which are states for which no transition enters. These states can be efficiently calculated using existential quantification and complementation as follows:

$$\overline{[\underset{x_i \in x}{\exists} [TR_i(x, y)]]}$$

We observe that source-less states that are not initial states are unreachable. The complement of these unreachable states yields a superset of the reachable states of the sub-FSM.

For the following discussions, we assume that the approximate reachable states for the i^{th} sub-FSM is denoted by $R^+{}_i(x)$ and let the set of approximate reachable states be $R^+(x) = R^+{}_1(x) \cdot \ldots \cdot R^+{}_n(\bar{x})$.

3.2 Don't Care Transitions

We propose to two different TR minimization techniques, one using current-state-based DCs and one using next-state-based DCs.

A. Current-state-based DC transitions (CDC)

We observe that the transitions originating from states that are guaranteed unreachable (by the approximate reachability analysis) can be used as don't cares because such transitions cannot occur. That is,

$$\textit{minimized-}TR_i(x, y) = \text{bdd-minimize}(TR_i(x, y), R^+(x)),$$

where bdd-minimize(f, c) means that we minimize the BDD f using the complement of the BDD c as DCs. In other words, the BDD c denotes the care-set.

B. Next-state-based DC transitions (NDC)

We can also let transitions to unreachable states be DCs, i.e.,

$$\textit{minimized-}TR_i(x, y) = \text{bdd-minimize}(TR_i(x, y), R^+(y)).$$

However, to obtain correct results, these states must be removed at the end of each iteration. The efficient implementation of this removal will be described below.

3.3 Clustered Don't Care BDD

From the approximate reachability analysis, we obtain a superset of reachable states associated with each cluster separately, i.e., $R^+{}_1(x), \ldots, R^+{}_n(x)$, each of which identifies a care-set for the corresponding sub-FSM. Because the quality of BDD minimization is typically proportional to the size of DC set, an ideal approach may be to simplify the transition relation BDDs using R^+, which is the conjunction of all $R^+{}_i$'s, as a care-BDD because it provides minimum care-set size. In practice, however, this approach often fails because the size of R^+ can be prohibitively large.

Moreover, even when the size of R^+ is manageable, the BDD minimization algorithm may be quite time-consuming. Alternatively, it is possible to individually apply BDD minimization to each TR_i using R^+_i. However, this approach may lead to too poor minimization quality because the individual R^+_i may not provide a large enough DC set.

For these reasons, we propose a *clustered DC* approach, in which a DC cluster is formed for each TR_i by conjuncting those R^+_i's which are likely to improve the minimization quality for TR_i. To obtain the current-state-based DCs for TR_i, we identify $R^+_j(x)$*'s* for each TR_i such that the supporting variable of $R^+_j(x)$ includes at least one variable v which is also contained in the support set of TR_i. The reason is that a non-supporting variables in the care-BDD typically degrade the BDD minimization quality (which was the motivation in developing *restrict).* Our experiments indicate that the size of the partial conjunction of such R^+_j*'s* is generally quite manageable.

For the next-state-based DCs, we also use clustered DC BDDs. In this case, however, they take on a simplified form. Recall that all supporting next state variable sets associated with each cluster are pairwise disjoint and the supporting variable set of $R^+_i(y)$ is a subset of the supporting next state variable set of $TR_i(x, y)$. Consequently, the supporting variable set of $R^+_i(y)$ intersects only with the supporting variable set of TR_i (x, y). For this reason, we apply BDD minimization to TR_i (x, y) using only $R^+_i(y)$.

Recall that, for the next-state-based DCs, we must remove unreachable states that are inadvertently added to the reachable state set after each iteration. A naive implementation of this removal is to conjunct the set of reachable states found at each iteration with $R^+(y)$. Since the size of $R^+(y)$ is often too large, this is approach is infeasible. An alternative approach is to conjunct all the individual $R^+_i(y)$'s to the set of reachable states found at each iteration. This approach, however, is computationally expensive because it requires many BDD operations. We propose a third approach motivated by the observation that, for many clusters, minimizing TR_i (x, y) by $R^+_i(y)$ leads to no change in TR_i (x, y). This observation is important because it is sufficient to remove

only those states that are not contained in the conjunction of $R^+_i(y)$'s that alter their respective TR_i (x, y) via BDD minimization. Finding and using this set, denoted by R^*, is efficient because the conjunction need be done only once (before the iterations begin) and because this reduced set of conjunctions typically leads to a much smaller BDD than $R^+(y)$.

3.4 Transition Relation Minimization Using Don't Cares without Dynamic Variable Ordering

The algorithm for the improved reachability analysis is presented in Figure 1. Among the various BDD minimization algorithms, we use *restrict*. This is because *restrict* is very fast and effective when the DC fraction is very high [14, 16], which is the case when using our approximate reachability analysis as the source of DCs.

```
improved-reachability-analysis (TR(x, y): {TR_i | 1 ≤ i ≤ n}, S^0(x): initial states)
    R (x) = R_new(x) = S^0(x),  R*(x) = bdd_one
    R+(x) = approximate- reachability-analysis (TR, I)   /* R+ = {R+_i | 1 ≤ i ≤ n} */
    for i = 1 to n
        C(x) = bdd_one
        for j = 1 to n
            if j ∈ FANIN(TR_i) then C(x) = C(x) · R+_j(x)
        TR_i = bdd-minimize(TR_i(x, y), C(x))

    for i = 1 to n
        TR_i' = bdd-minimize(TR_i(x, y), R+_i(y))
        if TR_i' ≠ TR_i then
            R*(x) = R*(x) · R+_i(x)
            TR_i = TR_i'
    do {
        R (x) = R_new(x)
        R_new(x) = image (TR(x, y), R(x)) · R*(x)
    } while (R_new(x) ≠ R(x))
    return (R(x))
```

Figure 1. Improved reachability analysis using DC-based BDD minimization without dynamic variable ordering (R: reachable states, R^*: superset of R^+ used to remove inadvertently added unreachable states, R^+: approximate reachable states, $j \in$ FANIN(TR_i) if TR_j feeds TR_i)

3.5 Transition Relation Minimization Using Don't Cares with Dynamic Variable Ordering

Our transition relation minimization technique is general, so it can easily be incorporated into the reachability analysis using dynamic variable ordering. To maximize the efficiency of the reachability analysis, however, we explore some options motivated by the interaction between BDD minimization using DCs and dynamic variable ordering.

The first option is to perform BDD minimization using DCs only once, at the beginning of the reachability analysis. Since the BDDs representing the approximated reachable states are no longer needed after the minimization, they are removed. This reduces the number of BDD nodes in use, thereby reducing the run-time required for dynamic variable ordering.

The second option is motivated by the observation that if the minimized BDDs are restructured by dynamic variable ordering, then the new BDDs may be further reduced by repeating DC-based minimization after each re-ordering. This approach requires that the care-BDDs are maintained throughout the exact reachability analysis which may adversely effect the dynamic variable ordering in terms of degraded size reduction and increased runtime. We note, however, that if the dynamic variable ordering is set to ignore the minimization quality of the care-BDDs, the size degradation may be avoided at the expense of some additional run-time overhead.

Both options apply to both set of DCs discussed above. In the presence of dynamic variable ordering, using the next-state-based DC sets may not be desirable because the next-state-based DCs requires extra BDDs which may force dynamic variable ordering to be called more frequently, thereby possibly increasing CPU time. In fact, our experiments suggest that this disadvantage of the next-state-based DCs often outweighs the advantage gained by BDD minimization.

There are many factors determining which of the two options should be used, including the expected minimization ratio and the care-BDDs size. If, for example, the expected minimization ratio is significantly high and the size of care-BDDs is small, the second option probably will perform better than the first.

Figure 2 shows the algorithms for the improved methods with dynamic variable ordering

```
improved-reachability-analysis-with-dvo (TR(x, y), S^0)
    R = R_new = S^0, dvoFlag = 0, R+ = approximate-reachability-analysis (TR, S^0)
    for i = 1 to n
        C_i(x) = bdd_one
        for j = 1 to n
            if j ∈ FANIN(TR_i) then C_i(x) = C_i(x) ·R+_j(x)
            TR_i = bdd-minimize(TR_i(x, y), C_i(x))
    if ndc == 1 then
        R*(x) = bdd_zero
        for i = 1 to n
            TR_i' = bdd-minimize(TR_i(x, y), R+_i(y))
        if TR_i' ≠ TR_i then
            R*(x) = R*(x) · R+_i(x)
            TR_i = TR_i'
    do {
        R(x) = R_new(x)
        for i = 1 to n
            if mb == 1 then
                if dvoFlag == 1 then
                    for j = 1 to n
                        update_j = 1
                    dvoFlag = 0
            if update_i == 1 then
                TR_i = bdd-minimize(TR_i(x, y), C_i)
                update_i = 1
        R_new(x) = image (TR(x, y), R(x))
        if ndc == 1 then R_new(x) = R_new(x) · R*(x)
    } while (R_new(x) ≠ R(x))
    return (R(x̄))
```

Figure 2. Improved reachability analysis using DC-based BDD minimization when dynamic variable ordering is not used. (*dvoFlag*: dynamic variable ordering sets this field when it is called. *ndc* = 1 if next-state based DC is used. *mb* = 1 if multiple BDD minimization is applied. Other notations are the same as in Figure 1.)

enabled. The bold-faced parts are needed only for the second approach in which we keep the care-BDDs and apply BDD minimization using DCs to a cluster if dynamic variable ordering has been performed at least once since the cluster has been last visited. The invocation of dynamic variable ordering is not shown in the algorithm because it's controlled by the BDD package. We use a global variable, denoted by *dvoFlag*, to monitor when dynamic variable ordering has been called since the variable was last cleared.

4 Experimental Results

We incorporated the new techniques into VIS [17] using Long' BDD package [23]. 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 an UltraSPARC-Enterprise / 1G. The default clustering heuristic in VIS described in [17] is used in our experiments.

In our first experiment, dynamic variable ordering was disabled for the entire process. Table 1 reports the results from approximate reachability analysis. The fractions of approximate reachable states are very small for all examples. Only the example s1423 took more than 20 iterations for a sub-FSM traversal, and the results from the thresholding technique (with iteration threshold 20) are shown inside parenthesis. For this example, the maximum number of iterations required to finish a sub-FSM traversal is 242.

Circuit	$\# R^+$	$\% R^+$	*T (min)*
s1269	3.15e09	2.30	0.2
s1423	1.09e20 (3.89e20)	0.58 (2.06)	3.6 (0.3)
s3330	5.80e22	1.07e-15	0.3
s5378	2.48e35	1.06e-12	0.2

Table 1. Approximate reachability analysis without dynamic variable ordering. ($\#R^+$: approximate reachable states,$\%R^+$: {approximate reachable states / all states}$\cdot$100, T: runtime)

The exact reachability analysis was not completed for any circuit because the algorithm ran out of memory. We report the BDD size and run-time characteristics of each successfully completed iteration that took longer than one minute and less than 3 hours.

Table 2 illustrates the results from standard approximate reachability analysis and our new approach. Recall that here we apply minimization using both *CDC* (current-state-based DCs) and *NDC* (next-state-based DCs). For all circuits, we can see that BDD minimization leads to an overall runtime improvement for all completed iterations. Results from the circuit s1423 show that the

improvement factor increases as the traversal progresses and the involved BDD sizes become larger. For most of the circuits, we can see more than a 60% speed-up.

Notice that the runtime for the approximate reachability analysis is negligible compared to total runtime except for the example s1423. The results from the thresholding technique are shown inside parenthesis for s1423. We can see that the overall runtime is improved using thresholded approximation, but not significantly.

		Standard traversal			*Improved traversal*		
Circuit	*#I*	ΣTR	*Peak R*	*T (min)*	$\Sigma TR'$	*Peak R*	*T (min)*
s1269	2	52,504	>2,103K	**> 180.0**	21,373	15,886K	**113.8**
s1423	10	21,572	4,738K	**30.8**	14,771 (14,835)	2,812K (2,836K)	**28.8 (26.5)**
	11		>13,055K	**> 180.0**		7,781K (7,831K)	**108 (107.7)**
s3330	1	37,833	3,588K	**17.4**	31,997	1,971K	**5.7**
s5378	1	51,563	534K	**5.7**	22,667	491K	**2.5**
	2		5,497K	**113.3**		6,013K	**33.9**

Table 2. Comparison between standard and improved reachability analysis without dynamic variable ordering. (*#I*: reachability analysis iterations completed, ΣTR: transition relation size, *Peak R*: maximum size of intermediate results, *T*: runtime, $\Sigma TR'$: minimized transition relation size.)

We also analyzed benchmark circuits s1512 and s4863 but their results are not shown in the table. Circuit s1512 is not shown because the approximate reachability analysis reported all states as reachable states. Circuit s4863 is not shown because the first iteration was trivially fast and the second iteration took longer than 3 hours.

In our second experiment, dynamic variable ordering was enabled during the construction of transition relation and traversal. Due to availability, we only used the standard dynamic variable ordering package which considers all BDDs (i.e., does not target specific BDDs) when optimizing the variable ordering. In this case, we disabled dynamic variable ordering during the approximate reachability analysis to minimize its side-effect and to focus on the impact of BDD minimization. Note, however, that the number of transition function vectors contained in a cluster depends on the target cluster size in VIS, which means that clustering is affected by dynamic vari-

able ordering. For this reason, we obtained different results from approximate reachability analysis when dynamic variable ordering was enabled, which are reported in Table 3.

Circuit	*# R^+*	*% R^+*	*T (min)*
s1269	1.53e09	1.11	0.08
s1423	1.09e20 (3.89e20)	0.58 (2.06)	7.4 (0.8)
s3330	5.80e22	1.07e-15	0.07
s4863	3.10e26	0.002	0.07
s5378	2.48e35	1.06e-12	0.38

Table 3. Approximate reachability analysis with dynamic variable ordering. (Notations are the same as in Table 1)

We re-measured the performance of reachability analysis with the two types of DCs, i.e., *CDC* and *NDC*, together and alone. Also, we compared the effect of single BDD minimization vs. multiple BDD minimization. Specifically, three combinations of these features were analyzed, CDC with single minimization, CDC with multiple minimization, and CDC + NDC with multiple minimization. The last combination of CDC + NDC with multiple minimization is not shown because the impact of the overhead due to NDC and multiple minimization is clearly shown by the other combinations.

The results are given in Table 4. The entire traversal was completed for the circuits s1269 and s3330. The size of transition relation and minimized transition relation dynamically changes so we report the size of them just before the traversal begins. The results indicate that the option in which the transition relation BDDs are minimized once typically outperforms the option of repeating BDD minimization after each invocation of dynamic variable ordering. This indicates that the BDD size reduction obtained by the repeated BDD minimization is not typically large enough to compensate for the associated overhead. Also, the results show that the BDD size reduction achieved by NDC typically does not compensate for the associated overhead.

The single minimization technique with CDC yields smaller runtimes than the traditional technique for all examples. In three of the examples the runtime improvement is over 30% and as much as 70%. In one example (s1423) the TR size was reduced by 97%, indicating up to an order

of magnitude improvement! Consequently, our technique analyzed five iterations in less than 200 minutes while the traditional technique did not finish after 2880 minutes. This suggests that the runtime improvement of our algorithm can be more than 93%.

Notice that for all examples the runtime for full approximate reachability analysis is negligible compared to the overall runtime. This suggests that it may not always be beneficial to impose a threshold to the number of iteration during approximate analysis. For example, the results of s1423 show that the overall runtime is significantly smaller using the larger DC set obtained from the full approximate reachability analysis compared with the DCs obtained from the thresholding technique. This suggests that future work targeting the quality of the approximation may lead to significant improvements in overall runtime.

		Standard traversal		*Improved traversal*						
				CDC / Single BDD min		*CDC / Multiple BDD min*		*CDC+NDC / Single BDD min*		
Circuit	*#I*	ΣTR_0	*T*	$\Sigma TR'_0$	*T(min)*	ΣC_0	*T(min)*	$\Sigma TR'_0$	R^*_0	*T(min)*
s1269	10	15,236	**236**	10,034	**235**	125	**225**	9,792	76	**233**
s1423	11	20,039	**327**	15,524 (14,706)	**166** **(238)**	200 (126)	**148** **(389)**	15,243 (15,243)	102 (102)	**179** **(179)**
s3330	9	18,099	**394**	16,701	**197**	1,214	**368**	12,210	1,098	**698**
s4863	5	125,206	**>2880**	3,255	**197**	58	**233**	2,845	52	**204**
s5378	4	43,987	**248**	17,913	**231**	1,257	**222**	13,482	623,765	**273**

Table 4. Comparison between standard and improved reachability analysis with dynamic variable ordering. (*CDC*: current-state-based DC, *NDC*: next-state-based DC, ΣTR_0: transition relation size before traversal, $\Sigma TR'_0$: minimized transition relation size before traversal, ΣC_0: care-BDD size before traversal, R^*_0: size of R^* before traversal. Other notations are the same as in Table 2.)

5 Conclusion

In this paper, we have presented techniques to improve the efficiency of symbolic FSM traversal using DC-based BDD minimization. The key to the algorithm is that we use an approximate FSM traversal technique to derive the DC sets. We demonstrate the effectiveness of the approach on the standard FSM traversal framework. We also explored the impact of DC-based BDD minimization combined with dynamic variable ordering. The experimental results show that our techniques can significantly reduce the reachability analysis run-time for large FSMs. We conjecture that it may

be possible to obtain even better results by using a dynamic variable ordering routine that targets only the transition relation BDDs.

Our techniques are applicable to reachability analysis tools using conjunctive partitioned transition relations which form the largest class of symbolic reachability analysis tools.

We expect that our technique can further improve the efficiency of reachability analysis by incorporating state-of-the-art state decomposition techniques [24] to improve the quality of the approximate reachability analysis. It is also our belief that our approach can be easily incorporated into other applications and we are currently investigating its impact on CTL model checking.

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] M. Fujita, Y. Matsunaga, and T. Kakuda, "On Variable Ordering of Binary Decision Diagrams for the Application of Multi-Level Logic Synthesis," in *Proc. European Design Automation Conference*, pp. 50-54, 1991.

[4] N. Ishiura, H. Sawada, and S. Yajima, "Minimization of Binary Decision Diagrams Based on Exchanges of Variables," in *Proc. IEEE International Conference on Computer-Aided Design*, pp. 472-475, 1991.

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

[6] M. Sauerhoff and I. Wegener, "On the Complexity of Minimizing the OBDD Size for Incompletely Specified Functions," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 1435-1437, Nov. 1996.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan and D. L. Dill, "Symbolic Model Checking for Sequential Circuit Verification," *IEEE Trans. on Computers*, vol. 13, pp. 401-424, 1994.

[13] S. Chang, D. I. Cheng and M. Marek-Sadowska, "Minimizing ROBDD Size of Incompletely Specified Multiple Output Functions," in *Proc. European Design and Test Conference*, pp. 620-624, 1994.

[14] 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.

[15] R. Drechsler and N. Göckel, "Minimization of BDDs by Evolutionary Algorithms," in *Proc. International Workshop on Logic Synthesis,* 1997.

[16] 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.

[17] 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.

[18] R. K. Brayton, G. D. Hactel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Wwamy, and T. Villa, "VIS: A System for Verification and Synthesis," in *Proc. International Conference on Computer-Aided Verification*, pp.428-432, July, 1996.

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

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

[21] 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.

[22] 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.

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

[24] 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.