Preprint from Proceedings of Design Automation Conference, Los Angeles, USA, June 2000

Distance Driven Finite State Machine Traversal

Andreas Hett Christoph Scholl Bernd Becker

Institute of Computer Science, Albert-Ludwigs-University, 79110 Freiburg i. Br., Germany
email: <name>@informatik.uni-freiburg.de

ABSTRACT

Symbolic techniques have revolutionized reachability analysis in the last years. Extending their applicability to handle large, industrial designs is a key issue, involving the need to focus on memory consumption for BDD representation as well as time consumption to perform symbolic traversals of Finite State Machines (FSMs). We address the problem of reachability analysis for large FSMs, introducing a novel technique that performs reachability analysis using a sequence of "distance driven" partial traversals based on dynamically chosen prunings of the transition relation. Experiments are given to demonstrate the efficiency and robustness of our approach: We succeed in completing reachability problems with significantly smaller memory requirements and improved time performance.

1. INTRODUCTION

To decide whether a set of target states of a given *Finite State Machine* (FSM) can be reached from a set of initial states is a key problem in functional design verification. Forward state space traversal techniques solve this problem by an iterative fixed-point computation of all reachable states starting from a set of initial states. A significant number of techniques and refinements have been developed to make *Reachability Analysis* applicable for large designs. Especially symbolic techniques which avoid an explicit representation of the set of reachable states and the FSM Transition Relation (TR) by using BDD representations increased the problem sizes which could be solved by FSM traversal to a large extent [8, 9, 13, 3].

In order to reduce time and memory consumption for circuits with realistic sizes, several improvements of the basic symbolic FSM traversal techniques have been proposed. To avoid huge BDD representations of monolithic TRs for large FSMs, decomposition has been used: conjunctive partitioning for approximate FSM traversal (e.g. [7]) and disjunctive partitioning for exact FSM traversal (e.g. [4, 11]).

Taking into account that the largest BDDs often occur during intermediate steps, other researchers replaced the pure breadth-first traversal of the original approach by a sequence of partial traversals [12, 5]. Thus a sequence of simpler partial traversals is used to avoid large intermediate peak memory requirements.

In [12] single symbolic traversal steps are initiated only from subsets of all newly reached states which are chosen in a way that their BDD representation has a "high density", i.e. many states are represented by a compact BDD. In [5] a partial traversal is carried out, based on a pruned TR, where "high cost" nodes are replaced by terminal zero. These nodes are determined by establishing an "activity profile" according to data collected during a limited number of iterations (the learning phase). Thus, an underapproximation of the reachable states is enabled and at the end, all formerly left-out states are accumulated by a traversal using the original TR.

In this paper we introduce a novel technique for symbolic FSM traversal using sequences of partial traversals to avoid large peak memory requirements. In contrast to [12] and [5] our method has the following properties:

1. We perform a pruning of the TR based on an analysis of the newly reached states set BDD. Thereby, two concepts are combined: partial traversals based on pruned TRs and partial traversals based on subsets of the set of newly reached states.
2. At first, we only traverse "short edges" in the state transition diagram. In the successive phases of the algorithm "longer and longer" edges are used (see below).
3. Pruning is done dynamically during the traversal.
4. In spite of the dynamic application of pruning, efficiency of the Computed Table (CT)[1] is guaranteed. The importance of this property is proven by recent research (e.g. [15]) which has shown, that the efficiency of the CT plays a much more vital part in sequential than in combinational applications.

Our experiments underline the quality and robustness of the approach for monolithic and partitioned TRs.

The paper is structured as follows: In Section 2 basic definitions are given. Section 3 presents our approach to reachability analysis using distance driven partial traversals. Experimental results are presented in Section 4. Finally the results are summarized in Section 5.

2. PRELIMINARIES

In this section we briefly provide essential definitions of *Binary Decision Diagrams*, *FSMs* and *Exact State Space Traversal.*

Binary Decision Diagrams (BDDs) are directed acyclic graphs representing Boolean functions. In the restricted form of reduced, ordered BDDs (ROBDDs) as used for our work they provide canonical representations [2]. In the following we briefly call them BDDs. BDDs have proven to be an efficient data structure and nowadays are widely used in applications of VLSI CAD, including traversals of FSMs.

A **Finite State Machine** (FSM) is defined as $(I, O, S, \delta, \lambda, s_0)$, a 6-tuple where I (O) is the input (output) alphabet, S is a non-empty finite set of states, $\delta : S \times I \to S$ is the next state function, $\lambda : S \times I \to O$ is the output function, and $s_0 \in S$ is the initial state. Since we only consider FSMs corresponding to sequential circuits, in the following $I = \{0,1\}^k$, $O = \{0,1\}^m$ and $S = \{0,1\}^n$ contain bit vectors of fixed length. Then, the *characteristic functions* χ_R of subsets $R \subseteq S$ are Boolean functions $\chi_R : \{0,1\}^n \to \{0,1\}$ with $\chi_R(x) = 1 \iff x \in R$. The transition function $\delta : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$ can also be represented by the characteristic function of its Boolean relation $TR : \{0,1\}^n \times \{0,1\}^k \times \{0,1\}^n \to \{0,1\}$ with $TR(x, i, x') = 1 \iff \delta(x, i) = x'$. TR (the transition relation) is the characteristic function describing all existing

[1]The CT is used in BDD applications to prevent that identical computations are performed more than once [1].

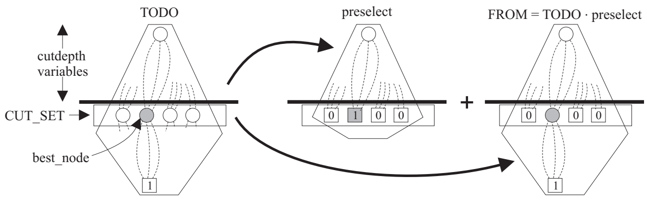

Figure 1: Determination of condition $preselect$**.**

transitions between states of the given FSM. The variables $x_1, \dots, x_n$ ($x'_1, \dots, x'_n$) corresponding to the first (last) n arguments of TR are called *current (next) state variables*, the variables $i_1, \dots, i_k$ are called *primary input variables.*
If $FROM$ is a set of states in S, the image of $FROM$ under δ is defined as follows:

$$\mathbf{Image}(\delta, FROM) := \{x' \in S \mid \exists i \in I, x \in FROM \text{ with } \delta(x,i) = x'\} \tag{1}$$

I.e. $\mathbf{Image}(\delta, FROM)$ is the set of states that can be reached from the set of states $FROM$ by means of a single time-step (transition). Thus, if the set of states $FROM$ is given by its characteristic function $FROM(x)$ and the TR is given by its characteristic function $TR(x, i, x')$, the image computation to determine the characteristic function $REACHED(x')$ of all states that can be reached from the set of states $FROM$ by a single transition can be performed by the following Boolean operations:

$$\begin{aligned} REACHED(x') &:= \mathbf{Image}(TR(x,i,x'), FROM(x)) \\ &:= \exists_{x,i}(TR(x,i,x') \cdot FROM(x)) \\ &:= \exists_x(\tilde{TR}(x,x') \cdot FROM(x)) \end{aligned} \tag{2}$$

with $\tilde{TR}(x,x') = \exists_i TR(x,i,x')$. Since the existential quantification for the input variables i can be done *before* the image computation for $FROM$, we assume in the following, that this existential quantification was done at the beginning of the FSM traversal and for simplicity we write $TR(x,x')$ instead of $\tilde{TR}(x,x')$.

Symbolic forward FSM traversal performs a fixed-point iteration process applying a sequence of image computations that accumulate all reachable states ($TOTAL\_REACHED$) for a starting set ($FROM$).

3. DISTANCE DRIVEN TRAVERSALS

3.1 Main Idea and Goals

In this section we describe our approach to perform FSM traversals by a sequence of *distance driven partial traversals.* The purpose of this approach is to prevent peak sizes in memory consumption, when the final reachable state set allows a compact BDD representation, but intermediate results of the straightforward BFS based traversal cannot be represented in reasonable size. We have the challenge to choose a suitable order for the collection of new reachable states to the set of reached states such that the representation is as compact as possible. More precisely, we pursue the following goals with our distance driven partial traversal strategy:

Goal 1: We try to use $FROM$ sets with compact BDD representations as starting points for image computations.

Goal 2: For each image computation we use a subset of the TR, that should contain only transitions leading to new states that provide a compact BDD representation when added to the set of already accumulated reachable states $Total\_REACHED$.

Our third goal is motivated by the great importance of the BDD Computed Table (CT) [1] to avoid identical computations especially for sequential applications [15]:

Goal 3: The subsetting of the TR should not decrease the performance of the CT.

```
procedure iterate_until_converge (TODO, preselect, select )
    FROM := TODO · preselect
    TODO := TODO · ¬preselect
    do
        TR' := TR · preselect · select
        New_REACHED := Image( TR', FROM)
        Total_REACHED :=
                Total_REACHED ∪ New_REACHED
        FROM := New_REACHED · preselect
        TODO := TODO ∪ (New_REACHED · ¬preselect)
    until (empty( FROM ))
```

Figure 2: Partial Traversal w.r.t. $preselect$**,** $select$

The intuition behind our method to achieve Goal 1 is that states with similar Hamming weights (number of 1's in the bit vector) [10] are supposed to combine to a compact BDD representation. Therefore the $FROM$ set of image computation should contain states with similar Hamming weights (HWs). We assume that we start the FSM traversal from the initial state $(0 \dots 0)$ with HW 0. Then we continue with states having small HWs and – step by step – we increase the HWs of the states, which are starting points of one time-step of reachability analysis. The subsetting of the TR is done in a way that we reach only new states "with a small distance" to the states in the $FROM$ set, i.e. states whose HW does not differ very much from the HWs of the states in the $FROM$ set. This fulfills (heuristically) our Goal 2. If we change the pruning of the TR several times during the reachability analysis algorithm, we have to be careful how to prune the TR to achieve *Goal 3* all the same.
In contrast to [5], where nodes with "high cost" are replaced by terminal zero, we pay more attention to the fact that two TRs, which result from different prunings, will have many cofactors in common. For the recursive procedure to compute the AND-EXIST operator of equation (2) this increases the probability to encounter common subproblems and thus to profit from CT hits. Since we want to adapt pruning dynamically during the traversal, this pruning strategy is essential to fulfill *Goal 3.*[2]
More details of the complete algorithm and the pruning method in particular are given in the next section.

3.2 Detailed Description of the Algorithm

In the first part of this section we describe, how the BDD $TODO$ representing reached states as starting points for image computations, is pruned before an image computation to achieve *Goal 1.* Afterwards we describe our dynamic pruning of the TR and finally show, how all parts work together leading to an algorithm, which performs a full FSM traversal using a sequence of partial distance driven traversals.

Pruning of $\boldsymbol{TODO}$**:** In contrast to straightforward BFS traversal algorithms we do not start an image computation from the set of all newly reached states, but only from a subset of them to achieve *Goal 1.* To restrict the states we perform an AND operation between the representation of the states, which were not yet processed ($TODO$), and a characteristic function $preselect$. $preselect$ is determined based on a Hamming weight (HW) metric. We consider a set CUT_SET of nodes of $TODO$ immediately below a cut line after the first *cutdepth* variables (see also Figure 1). In a first step for each node v_i in CUT_SET we consider all assignments to current state variables, which define a path passing through v_i and leading to terminal one (these assignments represent certain states of $TODO$) and for each node v_i we compute the sum of the HWs of these assignments.[3] Since we want to start with states having low HWs we choose the node $best\_node \in CUT\_SET$ as the one with

[2]Note that a decrease of CT efficiency due to dynamic application of pruning for several times is not a problem in [5], since pruning is performed only once based on an initial learning phase.

[3]This computation can be done in time linear to the number of nodes of $TODO$.

```
11 Reachability_Analysis
12 ...
13 TODO := s_0
14 Hamming_Weight := 1
15 do
16       do
17           (preselect, select) := determine_selectors( TODO,
                                                    Hamming_Weight )
18           iterate_until_converge( TODO, preselect, select )
19       until (empty( TODO ))
20       increase Hamming_Weight
21       TODO := Total_Reached
22 until ( Hamming_Weight = #next_state_variables )
```

Figure 3: Reachability Analysis using Distances

the smallest sum. Now *preselect* is the characteristic function of all assignments to the first *cutdepth* variables which lead to node *best_node*. The image computation is then started with $preselect \cdot TODO$ instead of $TODO$.

Pruning of TR: To achieve our *Goal 2*, we prune the TR to collect only states with similar HWs. The pruning can be viewed as a selection of edges in the state transition diagram of the FSM. It is done by a conjunction $TR' := TR \cdot preselect \cdot select$ of TR with the characteristic function *preselect* and a new characteristic function *select*. First, the characteristic function *preselect* selects only edges, which start from states fulfilling condition *preselect*. However, not all such edges are considered, but only "short edges". Here "short edges" denote edges connecting states with similar HWs. We select only edges between states whose Hamming distance is less or equal to a constant $Hamming\_Weight$[4]. The selection of short edges is done by a conjunction with the characteristic function *select* depending on next state variables where $ON(select) = \{y \mid \exists y' \in ON(preselect)$ with $Hamming\_distance(y, y') \leq Hamming\_Weight\}$.

Partial Traversal in Phases: Using our pruning methods for the BDD $TODO$ and for the TR we obtain an algorithm for FSM traversal which proceeds in rounds and phases. The algorithm is illustrated in Figures 2 and 3.
In summary, the complete algorithm proceeds in $\lceil \log(cutdepth) \rceil + 1$ phases. In each phase we work with a constant HW to restrict the "length of edges" in the TR. Each phase is divided into rounds. In each round, depending on the choice of the condition *preselect*, we process a different subspace of the total state space until no new states can be reached in this subspace. In each round a pruned transition relation TR' is chosen dynamically.
Procedure **iterate_until_converge** (see Figure 2) performs a single round of the algorithm. It performs a fixed point iteration starting from a set $TODO$ of states using the pruned $TR' = TR \cdot preselect \cdot select$. All reached states are collected in set $Total\_REACHED$. Since **iterate_until_converge** starts image computations only from states fulfilling condition *preselect*, we have to collect states which are reached, but not yet processed by image computations, in a new set $TODO$ (lines 3, 9).
Figure 3 gives an overview of the whole FSM traversal algorithm: We start the first phase with $Hamming\_Weight$ 1 to compute the selectors *select* and *preselect* (lines 14, 17). Now we iterate in procedure **iterate_until_converge** the image computation until no new states are reached assuming selectors *select* and *preselect* (line 18). This process is repeated until the set $TODO$, which is provided by **iterate_until_converge**, will become empty (loop of lines 16–19).
When $TODO$ is empty, we are not finished however, since we used a pruned TR with only "short edges". Now we have to enter a new phase: We increase $Hamming\_Weight$ (line 20), which restricts the selection of edges to be included in the pruned transition relation, now allowing also longer edges. For each phase we double the constant $Hamming\_Weight$ and we repeat the process until $Hamming\_Weight$ is maximal, i.e. until it equals the number of state variables (loop of lines 15–22). Finally we have accumulated all reachable states in $Total\_REACHED$.
Experimental results in Section 4 prove that the order in which we visit new reached states in our distance driven traversal is really efficient to reduce peak sizes in memory consumption, which occur for the straightforward BFS based traversal.
Furthermore, also the runtime behaviour is improved. Using our special method to prune the TR we also succeed in achieving Goal 3: If we can assume that corresponding current state and next state variables are neighboured in the BDD variable order (which is usually true in FSM traversal applications), *preselect* and *select* depend only on the first $2 \cdot cutdepth$ variables in the variable order, such that cofactors of $TR' := TR \cdot preselect \cdot select$ with respect to $2 \cdot cutdepth$ variables (or more variables) will also occur as cofactors of TR. Since the recursive BDD synthesis procedures are always working with a same set of cofactors of TR, we achieve an efficient CT usage leading also to small runtimes (see Section 4).

[4]For reasons of efficiency the Hamming weight of the states is only considered for the first *cutdepth* state variables here.

4. EXPERIMENTAL RESULTS

In this section, experimental results for our method are compared with standard, partitioned and activity profiling [5] traversals. All measurements were executed on an Ultra-II model 2170 with 1 GByte main memory. A memory limit of 800 MByte and a time limit of 5,000 CPU seconds were given. Improvements of more than 100 % are presented in bold face, runtimes are given in seconds, peak sizes represent numbers of BDD nodes.
Table 1 contains results for model checking traces first introduced in [15, 14]. For these traces the relevant FSM information for performing reachability analysis has been extracted, without any modifications of the synthesis process originally given.
$|TR|$ denotes the number of BDDs nodes of the TR. *Depth* is the traversal depth of the FSM. The columns $|Reached|$ and *#Reached* denote the number of BDDs nodes for the reachable states set and the number of reachable states, respectively.
The column *Original Method* denotes our competitor, a standard FSM traversal process provided by the CUDD package [6] fully exploiting the rich set of newly added features for version 2.3.0 (e.g. the death-row for delayed freeage of BDDs improving CT efficiency). For our method (denoted by *Distance Driven*) we applied a "cutdepth" value of 8 variables. When comparing the values presented in Table 1, an average performance improvement of a factor of about 2.9 for the time performance can be noticed. For some traces, the runtimes even yield an improvement factor up to 17 (*furnace17*). Large peak sizes can be avoided by our traversal thanks to the focus on compact state sets representation, yielding an average improvement factor of almost 3 concerning peak sizes. Again some of the benchmarks yield results outstandingly better than the average value, e.g. *over12* and *mmgt20* with improvement factors of about 7.
A major problem when performing reachability analysis relies on the fact that in many cases it is not feasible to even construct the initial TR monolithically. Therefore the TR needs to be build using a conjunctive or disjunctive partitioning. In the following we will underline the fact, that our approach yields adequate results as shown for monolithic TRs as well for non-monolithic TRs.
As underlying software platform for the following series of experiments the traversal tool *PdTrav 1.2* provided by [6] was used. It needs to be mentioned that the benchmarks *s1512*, *s3384* and *s5378* were excluded from the tables since, independent of the approach considered here, they did not finish calculations either due to given memory or runtime limit when using a fixed variable ordering. All initial variable orderings used were provided by [6].
In Table 2 we present comparisons of three traversal methods all implemented in the *PdTrav 1.2* traversal tool. *TD*

Circuit	$\|TR\|$	Depth	$\|Reached\|$	#Reached	Original Method		Distance Driven	
					Peak Size	Runtime	Peak Size	Runtime
furnace17	7,264	174	845	8.9×10^{19}	4.0 M	139	**0.2 M**	**8**
key10	9,426	151	17,179	1.1×10^{12}	3.8 M	165	2.1 M	**67**
over12	6,782	90	3,671	5.9×10^{16}	9.2 M	507	**1.2 M**	**32**
mmgt20	6,167	144	9,756	8.1×10^{31}	4.2 M	248	**0.7 M**	**30**
dme2-16	141,840	433	8,353	1.4×10^{18}	5.7 M	500	3.8 M	269
dpd75	7,409	371	4,396	4.1×10^{60}	5.2 M	766	**2.4 M**	**238**
ftp3	6,399	58	55,937	5.9×10^{8}	3.6 M	339	2.5 M	283

Table 1: FMCAD'98 benchmarks - monolithic TRs

Circuit	TD	#RS	PT	#Cl	$\|TR\|$	Original Method		Activity Profiling		Distance Driven	
						Peak Size	Runtime	Peak Size	Runtime	Peak Size	Runtime
s1269	9	1.1×10^{9}	5,000	6	12,122	10.7 M	4,596	0.4 M	18	0.8 M	52
s3271	16	1.3×10^{31}	500	17	6,158	1.9 M	4,191	1.3 M	664	3.5 M	**329**
s3330	7	7.3×10^{17}	500	17	7,891	timeout	timeout	1.4 M	358	1.9 M	320
s4863	4	2.2×10^{19}	5,000	39	85,384	0.4 M	53	0.2 M	76	0.9 M	49
s1269	9	1.1×10^{9}	500	12	5,946	10.2 M	4,577	11.6 M	2,411	**1.3 M**	**109**
s3271	16	1.3×10^{31}	5,000	7	21,403	timeout	timeout	1.5 M	1,761	4.8 M	**545**
s3330	7	7.3×10^{17}	5,000	6	20,950	timeout	timeout	0.6 M	2,610	2.2 M	**798**
s4863	4	2.2×10^{19}	500	50	61,447	timeout	timeout	timeout	timeout	**5.6 M**	**350**

Table 2: ISCAS'89 benchmarks – partitioned TRs

gives the traversal depth of the circuits, *#RS* the number of reachable states. *Original Method* denotes a straightforward BFS based traversal, *Activity Profiling* shows the results for the approach presented in [5]. The usage of this method demands the setting of several parameters (e.g. pruning threshold and heuristics, number of iterations for the learning phase, choice of image computation during learning phase). For the sake of simplicity we used the parameter settings separately provided for each benchmark [6] and applied the available scripts. The scripts also provide a suggested clustering for the transition relation. The partitioning threshold (*PT*) is given in column 2 of the table. The corresponding number of clusters (*#Cl*) and the size $|TR|$ of the shared BDDs representing the TR are presented in columns 5 and 6. To underline the quality of our results, i.e. giving an impression of the robustness of our method, we additionally present measurements for a second partitioning threshold (*PT*) (taken from the set {500, 5.000}), giving a different starting point for the same circuit. The results for the second set of PTs are shown in the lower half of Table 2. For our method (denoted by *Distance Driven*) we applied a "cutdepth" value of 14 variables. It should be mentioned that this is the only parameter that has to be set for distance driven FSM traversal. Moreover, we made the experience that the heuristic is robust against small changes in the "cutdepth" value.
Obviously, the choice of clustering, i.e. the representation of the initial problem has large impact on the overall complexity of the synthesis process. Benchmark *s4863* is a good example, being handled in less than one minute when having a "good" clustering, but on the other hand it is not solvable for the *Original Method* and *Activity Profiling* within the given limits, if not. As shown, among the approaches considered here, only the method *Distance Driven* is capable to partly overcome the handicap of a "bad" clustering and offers a reasonably robust behaviour. Overall, our approach outperforms a straightforward BFS based traversal both in BDD node peak sizes and runtimes for non-monolithic TRs. For some of the peak sizes our results are slightly worse than the *Activity Profiling* results, but our approach has the advantage of an improved robustness behaviour. Concerning runtimes, up to one exception, our runtimes are the best of all presented competitors.

5. CONCLUSIONS

In this paper we have presented a novel technique for symbolic FSM traversal which is based on a sequence of distance driven partial traversals using a dynamic pruning of the TR and the state sets as well.
Our experimental results underline the quality of the approach, showing that distance driven FSM traversal has much smaller memory requirements than straightforward BFS based traversal and significantly improved time performance. Furthermore, it also compares favorably to more sophisticated methods, like partitioned traversal combined with activity profiling.
As part of ongoing work, we are currently investigating the chances and influences of an automatical adjustment and variation of the "best" cutdepth during the reachability process. Since for non-monolithic TRs the variable support for all partitions can vary a lot (e.g. dependent on the methods for clustering), another point of great interest is the heuristical choice of the "cutdepth" for each TR partition independently.

6. REFERENCES

[1] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[2] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[3] J.R. Burch, E.M. Clark, K.L. McMillan, and D.L. Dill. Sequential circuit verification using symbolic model checking. In *Design Automation Conf.*, pages 46–51, 1990.

[4] G. Cabodi, P. Camurati, L. Lavagno, and S. Quer. Disjunctive partitioning and partial iterative squaring: An effective approach for symbolic traversal of large circuits. *Design Automation Conf.*, 34:728–733, 1997.

[5] G. Cabodi, P. Camurati, and S. Quer. Improving symbolic traversals by means of activity profiles. *Design Automation Conf.*, 36:306–311, 1999.

[6] G. Cabodi and S. Quer. http://www.polito.it/~quer/software.htm.

[7] H. Cho, G.D. Hachtel, E. Macii, B. Plessier, and F. Somenzi. Algorithms for approximate FSM traversal. In *Design Automation Conf.*, pages 25–30, 1993.

[8] O. Coudert, C. Berthet, and J.C. Madre. Verification of sequential machines based on symbolic execution. In *Automatic Verification Methods for Finite State Systems, LNCS 407*, pages 365–373, 1989.

[9] O. Coudert and J.C. Madre. A unified framework for the formal verification of sequential circuits. In *Int'l Conf. on CAD*, pages 126–129, 1990.

[10] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Jour.*, 9:147–160, April 1950.

[11] A. Narayan, A. Isles, J. Jain, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Reachability analysis using Partitioned-ROBDDs. In *Int'l Conf. on CAD*, pages 388–393, 1997.

[12] K. Ravi and F. Somenzi. High-density reachability analysis. In *Int'l Conf. on CAD*, pages 154–158, 1995.

[13] H. Touati, H. Savoj, B. Lin, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Implicit enumeration of finite state machines using BDDs. In *Int'l Conf. on CAD*, pages 130–133, 1990.

[14] B. Yang. http://www-cgi.cs.cmu.edu/afs/cs.cmu.edu/user/bwolen/Web/software/.

[15] B. Yang, R.E. Bryant, D.R. O'Hallaron, A. Biere, O. Coudert, G. Janssen, R.K. Ranjan, and F. Somenzi. A performance study of BDD-based model checking. In *Proceedings of Formal Methods in Computer-Aided Design, LNCS 1522*, pages 255–289, 1998.