Distance Driven Finite State Machine Traversal

Andreas Hett         Christoph Scholl      Bernd Becker
Institute of Computer Science, Albert-Ludwigs-University, 79110 Freiburg i. Br., Germany
e-mail: <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 state space 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 \((I, O, S, \delta, s_0)\), a 6-tuple where \(I\) (\(O\)) is the input (output) alphabet, \(S\) is a non-empty finite set of states, \(\delta : S \times I \rightarrow S\) is the next state function, \(\lambda : S \times I \rightarrow O\) 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 \(x_R\) of subsets \(R \subseteq S\) are Boolean functions \(x_R : \{0, 1\}^n \rightarrow \{0, 1\}\) with \(x_R(x) = 1 \iff x \in R\). The transition function \(\delta : \{0, 1\}^k \times \{0, 1\}^k \rightarrow \{0, 1\}^n\) can also be represented by the characteristic function of its Boolean relation \(\delta : \{0, 1\}^k \times \{0, 1\}^k \rightarrow \{0, 1\}^n\) with \(\delta(x, y) = 1 \iff \delta(x) = y\). 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].
transitions between states of the given FSM. The variables \( x_1, \ldots, x_n \), corresponding to the first (last) \( n \) arguments of \( TR \) are called current (next) state variables, the variables \( i_1, \ldots, i_k \) are called primary input variables. If \( FROM \) is a set of states in \( S \), the image of \( FROM \) under \( \delta \) is defined as follows:

\[
\text{Image}(\delta, FROM) := \{ x' | \exists i \in I, x \in FROM \text{ with } \delta(x, i) = x' \}
\]

(1)

I.e. \( \text{Image}(\delta, FROM) \) 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:

\[
\begin{align*}
REACHED(x') &:= \text{Image}(TR(x, i, x'), FROM(x)) \\
&:= \exists i \cdot (TR(x, i, x') \cdot FROM(x)) \\
&:= \exists i \cdot (TR(x, i, x') \cdot FROM(x))
\end{align*}
\]

(2)

with \( TR(x, i, x') = \exists TR(x, i, x') \cdot FROM(x) \). 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 \( TR(x, i, x') \) instead of \( TR(x, i, x') \).

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 straight-forward 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 representation 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.

---

### 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 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 \( \text{variables} \) (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. Since we want to start with states having low HWs we choose the node \( \text{best} \_node \in CUT \_SET \) as the one with

---

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.

This computation can be done in time linear to the number of nodes of \( TODO \).
Reachability Analysis

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 - preselect - 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 HammingWeight.

Partial Traversal in Phases: Using our pruning methods for the BDD TOD0 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 \( \log(\text{cutdepth}) + 1 \) 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 \textit{Iterate until converge} (see Figure 2) performs a single round of the algorithm. It performs a fixed point iteration starting from a set TOD0 of states using the pruned TR = TR - preselect - select. All reached states are collected in set Total\_REACHED. Since \textit{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 TOD0 [lines 3, 9].

Figure 3 gives an overview of the whole FSM traversal algorithm: We start the first phase with HammingWeight = 1 to compute the selectors select and preselect (line 18). This process is repeated until the set TOD0, which is provided by \textit{Iterate until converge}, will become empty (loop of lines 16-19).

When TOD0 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 HammingWeight (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 HammingWeight and repeat the process until HammingWeight 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 the 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 traversals), preselect and select depend only on the first 2 - cutdepth variables in the variable order, such that factors of TR := TR - preselect - select with respect to 2 - cutdepth variables (or more variables) will also occur as factors of TR. Since the recursive BDD synthesis procedures are always working with a same set of factors of TR, we achieve an efficient CT usage leading also to small runtimes (see Section 4).

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 CUDU package [6] fully exploiting the rich set of newly added features for version 2.3.0 (e.g. the death-row for delayed freee 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 improvements even increase up to a factor of about 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. over15 and mmp@50 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 as for non-monolithic TRs.

Table 2: We found it beneficial to show the following series of experiments the traversal tool PDrav 1.2 provided by [6] was used. It needs to be mentioned that the benchmarks s1512, s3583 and s5578 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 PDrav 1.2 traversal tool TD.
Table 1: FMCAD’98 benchmarks – monolithic TRs

<table>
<thead>
<tr>
<th>Circuit</th>
<th>TR</th>
<th>Depth</th>
<th>#Resolved</th>
<th>#Resolved</th>
<th>Original Method</th>
<th>Activity Profiling</th>
<th>Driven Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jnmcx7</td>
<td>7,264</td>
<td>174</td>
<td>845</td>
<td>8.0 x 10^7</td>
<td>4.0 M</td>
<td>139</td>
<td>0.2 M</td>
</tr>
<tr>
<td>kg610</td>
<td>7,426</td>
<td>151</td>
<td>17,179</td>
<td>1.1 x 10^10</td>
<td>3.8 M</td>
<td>165</td>
<td>2.1 M</td>
</tr>
<tr>
<td>over12</td>
<td>6,792</td>
<td>200</td>
<td>3,671</td>
<td>5.0 x 10^6</td>
<td>9.2 M</td>
<td>507</td>
<td>1.2 M</td>
</tr>
<tr>
<td>mtmg20</td>
<td>6,167</td>
<td>144</td>
<td>9,756</td>
<td>8.1 x 10^11</td>
<td>4.2 M</td>
<td>248</td>
<td>0.7 M</td>
</tr>
<tr>
<td>dmc816</td>
<td>141,840</td>
<td>433</td>
<td>8,353</td>
<td>1.4 x 10^8</td>
<td>5.7 M</td>
<td>500</td>
<td>3.1 M</td>
</tr>
<tr>
<td>dpd75</td>
<td>7,409</td>
<td>371</td>
<td>4,396</td>
<td>4.1 x 10^7</td>
<td>5.2 M</td>
<td>702</td>
<td>2.4 M</td>
</tr>
<tr>
<td>rpm3</td>
<td>6,309</td>
<td>657</td>
<td>52,937</td>
<td>5.0 x 10^8</td>
<td>3.6 M</td>
<td>239</td>
<td>2.5 M</td>
</tr>
</tbody>
</table>

Table 2: ISCAS’89 benchmarks – partitioned TRs

<table>
<thead>
<tr>
<th>Circuit</th>
<th>TD</th>
<th>#BS</th>
<th>PT</th>
<th>#C</th>
<th>TR</th>
<th>Original Method</th>
<th>Activity Profiling</th>
<th>Driven Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>s3269</td>
<td>9</td>
<td>1.3 x 10^8</td>
<td>5,000</td>
<td>6</td>
<td>12,122</td>
<td>10.7 M</td>
<td>4.096</td>
<td>0.4 M</td>
</tr>
<tr>
<td>s3721</td>
<td>16</td>
<td>1.3 x 10^3</td>
<td>500</td>
<td>17</td>
<td>6,158</td>
<td>1.0 M</td>
<td>4,191</td>
<td>1.3 M</td>
</tr>
<tr>
<td>s3330</td>
<td>7</td>
<td>7.3 x 10^7</td>
<td>500</td>
<td>17</td>
<td>7,891</td>
<td>timeout</td>
<td>timeout</td>
<td>1.4 M</td>
</tr>
<tr>
<td>s4663</td>
<td>4</td>
<td>2.2 x 10^5</td>
<td>5,000</td>
<td>39</td>
<td>85,384</td>
<td>0.4 M</td>
<td>53</td>
<td>0.2 M</td>
</tr>
<tr>
<td>s3269</td>
<td>9</td>
<td>1.3 x 10^8</td>
<td>5,000</td>
<td>7</td>
<td>21,403</td>
<td>10.2 M</td>
<td>4,577</td>
<td>11.6 M</td>
</tr>
<tr>
<td>s3721</td>
<td>16</td>
<td>1.3 x 10^3</td>
<td>5,000</td>
<td>7</td>
<td>21,403</td>
<td>timeout</td>
<td>timeout</td>
<td>1.5 M</td>
</tr>
<tr>
<td>s3330</td>
<td>7</td>
<td>7.3 x 10^7</td>
<td>5,000</td>
<td>6</td>
<td>20,050</td>
<td>timeout</td>
<td>timeout</td>
<td>0.6 M</td>
</tr>
<tr>
<td>s4663</td>
<td>4</td>
<td>2.2 x 10^5</td>
<td>500</td>
<td>50</td>
<td>61,447</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
</tr>
</tbody>
</table>

6. REFERENCES