

Trace Compaction using SAT-based Reachability Analysis

Sean Safarpour, Andreas Veneris, Hratch Mangassarian
Department of Electrical and Computer Engineering
University of Toronto, Toronto, Canada
{sean, veneris, hratch}@eecg.toronto.edu

Abstract— In today’s designs, when functional verification fails, engineers perform debugging using the provided error traces. Reducing the length of error traces can help the debugging task by decreasing the number of variables and clock cycles that must be considered. We propose a novel trace length compaction approach based on SAT-based reachability analysis. We develop procedures and algorithms using pre-image computation to efficiently traverse the state space and reduce the trace lengths. We further introduce a data structure used to store the visited states which is critical to the performance of the proposed approach. Experiments demonstrate the effectiveness of the reachability approach as approximately 75% of the traces are reduced by one or two orders of magnitudes.

I. INTRODUCTION

Functional verification of digital circuits is a major problem for the VLSI design community. It is reported that up 70% of the cost and effort of VLSI design is due to verification and debugging [1]. Debugging, which consists of locating and fixing errors or bugs in an erroneous design, is responsible for approximately 50% of the overall verification cost [1].

Given a sequential circuit and golden model that specifies the correct behavior of the circuit, verification tools can determine whether the circuit is consistent with the golden model. Many different verification approaches exist today such as simulation-based methods, and bounded and unbounded formal techniques [2]. Despite the recent advances in the field of formal verification, most VLSI companies still use simulation techniques as a central verification strategy [1].

Performing verification via simulation cannot prove the correctness of a design unless the complete behavior of the design is exercised [2]. Since proving the correctness may not be an option for today’s large designs, performing a large number of simulations can achieve a high level of confidence in the design’s correctness. A testbench can exercise the design with the help of random or semi-random stimulus generators. The testbench can also determine whether the design and the model are inconsistent in their response to the stimulus. In this case, an *error trace* or a *counter-example*, consisting of a sequence of actions or states from the initial states to the error, is produced.

The verification engineer has the responsibility of determining why a design and a golden model have inconsistent behaviors based on the *error trace(s)*. Since a trace is often derived from random simulation, the sequence of events leading to the error can be unnecessarily long. In other words, a shorter error trace may be able to describe the same erroneous behavior in less clock cycles. With a shorter trace, the debugging task of the verification engineer can be considerably reduced as fewer signals and clock cycles must be considered. As a result, reducing the length of traces can substantially increase the efficiency of design debugging.

Previous work shows that for random and semi-random based simulations, error traces can often be reduced to a fraction of their

initial size [3], [4], [5], [6]. One such technique uses forward image computation using Binary Decision Diagrams (BDDs) to reduce the traces [3]. In [4], techniques are presented to remove variables from counter examples in order to simplify them, but their lengths are not reduced. Another recent work uses several techniques based on performing further simulations and Bounded Model Checking (BMC) to achieve small traces [5]. The technique of [6] is the closest to ours as they utilize a sequential Boolean Satisfiability (SAT) solver to find short-cuts in the original trace. More specifically, [6] seeks to find the shortest path from the initial state to some candidate intermediate state similar to BMC but using a sequential SAT solver.

In this work, we propose a trace length compaction technique where the shortest path from the initial state to a final state is sought. This approach is based on *reachability analysis* where an all-solution SAT solver is used as the *pre-image* computation engine [7], [8], [9]. The benefits over the existing BDD [3] and BMC techniques [5] are that the BDD memory explosion problem can be averted and that compactations exceeding the finite bound of BMC approaches may be applied. Our technique appears to share many of the advantages of the sequential SAT approach proposed in [6]. The main difference is that ours relies on reachability analysis and pre-image computation while making use of a novel data structure to determine state containment relationships.

More specifically, the contributions of this paper are the following.

- A trace compaction technique based purely on pre-image computation and reachability analysis using an all-solution SAT solver.
- A set of containment rules that help draw relationships between existing states and states found through pre-image computation which may result in shorter traces.
- A state selection procedure within the reachability analysis engine and a set of heuristics that improve the performance of the overall approach in practice.
- A novel data structure for storing visited states that allows for quick identification of state containment relationships.

This paper is organized as follows. In the next section, some background information is provided on finite state machines, pre-image computation, and reachability analysis. Section III presents the proposed trace compaction approach and discusses its central procedures. Section IV, introduces a novel data structure critical for the efficient performance of the proposed approach. Sections V and VI demonstrate the experimental results and conclude the paper, respectively.

II. PRELIMINARIES

In this section we provide some background on Finite State Machines, traces, image and pre-image computation, and reachability analysis. We assume that the reader is familiar with SAT solver terminology [7].

A. Finite State Machines

A sequential digital circuit can be modeled by a Finite State Machine (FSM) represented by a 6-tuple $M := (Q, \Sigma, \Delta, \delta, \lambda, q_0)$

where Q is the finite set of states, Σ and Δ are the input and output alphabets respectively, $\delta : Q \times \Sigma \rightarrow Q$ is the state transition function, $\lambda : Q \times \Sigma \rightarrow \Delta$ is the output function, and q_0 is the initial state [2]. Figure 1 illustrates a simple FSM where the states are represented by nodes and the transitions are represented by edges.

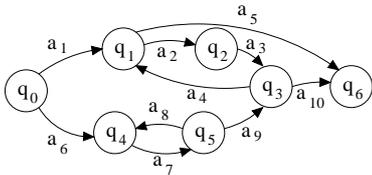


Fig. 1. Finite State Machine with 7 states

A trace of length k for an FSM is an input sequence $\langle a_1, a_1, \dots, a_k \rangle$ that leads the FSM through a sequence of states $\langle q_0, q_1, \dots, q_{k-1}, q_k \rangle$. Note that some states may be repeated in the state sequence. Figure 2 represents one possible trace for the FSM of Figure 1.

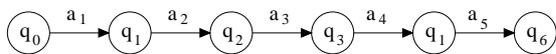


Fig. 2. A sample trace for the above FSM

B. Image and Pre-image Computation

Given a sequential circuit with current state variables V and next state variables V' , a set of current states and a set of next states are labeled by $Q(V)$ and $Q(V')$ respectively. The transition relation from a set of states $Q(V)$ to $Q(V')$, denoted by $T(Q(V), Q(V'))$, is true for each pair of $Q(V)$ and $Q(V')$ if $\delta(Q(V)) = Q(V')$ for a set of input assignments [2]. Given the above, the image and pre-image of a circuit can be defined as follows.

$$\text{IMAGE: } Q(V') = \exists V. (T(Q(V), Q(V')) \wedge Q(V))$$

$$\text{PRE-IMAGE: } Q(V) = \exists V'. (T(Q(V), Q(V')) \wedge Q(V'))$$

Intuitively, the image of a state q_i is all the states that can be reached from q_i under all possible input combinations in a single clock cycle. Similarly, the pre-image of q_i comprises of all the states that can lead to q_i under all possible input combinations in one clock cycle. In the FSM of Figure 1, the image of state q_1 is $\{q_2, q_6\}$ while its pre-image is $\{q_0, q_3\}$.

Although the image and pre-image of circuits are traditionally computed using BDDs [2], some techniques based on all-solution Boolean Satisfiability (SAT) solvers can also be used [8], [9], [10], [11]. All-solution SAT solvers can compute the pre-image set $Q(V)$ by constraining the circuit CNF to $Q(V')$ and iteratively finding all the solutions that satisfy the CNF in terms of the current state variables V [10]. Recent work on SAT-based Unbounded Model Checking (UMC) and pre-image computation techniques have demonstrated considerable advancements [8], [9], [10], [11].

In this work, we are mainly concerned with SAT-based pre-image computation. Since this technique finds states one at a time, we use the term *pre-image* loosely to also refer to a *single* state q_j that belongs to the pre-image of q_i . Furthermore, we use the term *state* to refer to a *state cube*, which is a state encoding that may contain *unassigned* or *don't care* variables. As such, a state may be a superset (cover) of other states. For instance, the state cube $\{v_1, v_2, v_3\} = 1X1$ covers the states $\{v_1, v_2, v_3\} = 101$ and $\{v_1, v_2, v_3\} = 111$. For brevity, in the remaining of this paper we drop the variable names (i.e. v_1, v_2, v_3) when describing state values.

C. Reachability Analysis

Reachability analysis is the process of determining whether a state q_k is reachable from another state q_0 . In the realm of UMC,

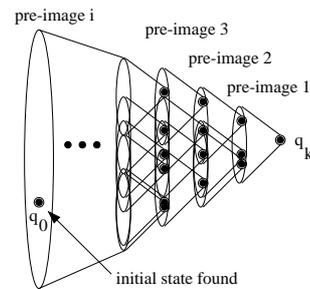


Fig. 3. Illustration of reachability analysis

reachability analysis can be used to check CTL properties of type EFq_k where q_k is a bad state and q_0 is a legal or initial state [2].

Intuitively, reachability analysis traverses the state space backwards from state q_k until a state q_0 is found or a fix-point, where no new states are found, is reached [2]. Pre-image computation is a central procedure of reachability analysis as it performs the single backward steps. The manner in which the state space is traversed depends on which of the visited states is selected for each pre-image computation step. If the visited states are stored in a stack-like data structure, a depth-first traversal is performed, while a queue-like data structure results in a breadth-first traversal. Figure 3 illustrates a breadth-first reachability analysis process that eventually finds the initial state q_0 . In this figure, the black nodes represent states while each cone represents a set of states found by one pre-image computation step.

III. PROPOSED TRACE COMPACTION APPROACH

In this section we present our proposed trace length compaction approach. First we introduce the central concept followed by details of the state selection procedure and the all-solution SAT solver.

A. Reachability Based Trace Compaction

A trace can be represented by a directed graph $G = (N, E)$ where the nodes N represent states and the edges E represent transitions between states. An edge from state q_i to q_j denotes that q_i belongs to the pre-image of q_j and q_j belongs to the image of q_i . Our objective is to reduce the length of the path from the initial state q_0 to the final state q_k by applying pre-image computation and reachability analysis techniques.

Our proposed approach performs reachability analysis on all the states belonging to the original trace. The manner in which states are selected for reachability analysis is described in Section III-C. All the states (or state cubes) found by the pre-image computation steps of the reachability engine are added to the graph G . Graph G is updated with edges denoting that each newly found states q_i is a pre-image of some state q_j , selected for pre-image computation.

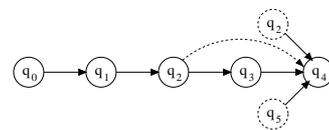


Fig. 4. Updating the graph G with new nodes and edges

When states found by pre-image computation already exist in the graph G , extra edges may be drawn in G to illustrate new legal transitions. These transitions may provide a shorter path (or short-cut) from the initial state to the final state thus reducing the overall trace length. For example consider the situation described in Figure 4 where the original trace is shown as the sequence $\langle q_0, q_1, q_2, q_3, q_4 \rangle$ and the dashed nodes are states found through reachability analysis. Since q_2 is found as a pre-image of q_4 , and q_1 is the pre-image of q_2 in the original trace, a new edge shown

as dashed line can be drawn directly from the original (non-dashed) q_2 to q_4 and the dashed q_2 can be removed. The overall result is a shorter path from q_0 to q_4 which skips node q_3 .

As motivated by the above example, finding state equivalences in the graph G can lead to more “short-cuts” which can reduce the overall trace size. Along with the state equivalence relation discussed, there are other state containment relationships that can lead to further short-cuts in the graph. The following rules determine how the graph G is updated after each pre-images computation step.

Consider state q_i found as a pre-image of state q_{i+1} , and the sequence $\langle q_{j-1}, q_j, q_{j+1} \rangle$ existing in the graph G .

- **Rule 1.** If $q_i = q_j$: State q_i is not added to G , but an edge is drawn from q_j to q_{i+1} .
- **Rule 2.** If $q_i \supset q_j$: State q_i is added to G , an edge is drawn from q_i to q_{i+1} , and another edge is drawn from q_j to q_{i+1} .
- **Rule 3.** If $q_i \subset q_j$: State q_i is added to G , an edge is drawn from q_i to q_{i+1} , another edge is drawn from q_{j-1} to q_i , and another edge is drawn from q_i to q_{j+1} .

The correctness of rule 1 is evident as the images of equivalent states are also equivalent. Rule 2 can be explained by expanding the state cube q_i into two components $q_i = \{q_j\} \cup \{q_i - q_j\}$. From here we use the fact that any image of q_i is also an image of q_j . Similarly, rule 3 can be explained by expanding q_j into two components $q_j = \{q_i\} \cup \{q_j - q_i\}$.

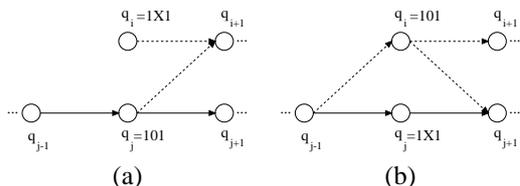


Fig. 5. Illustrating rules 2 and 3

The following example helps clarify rules 2 and 3. Consider state q_i found as a pre-image of state q_{i+1} , and the sequence $\langle q_{j-1}, q_j, q_{j+1} \rangle$, where state $q_i = 1X1$ and the state $q_j = 101$. By rule 2, an edge is first drawn from q_i to q_{i+1} to indicate that q_i is a pre-image of state q_{i+1} . Since $1X1 \supset 101$ and q_{i+1} is an image of $q_i = 1X1 = \{101\} \cup \{111\}$, then q_{i+1} must also be an image of $q_j = 101$. This scenario is illustrated in Figure 5 (a) with the new edges drawn as dashed lines. Similarly, by rule 3 an edge is first drawn to indicate that q_i is a pre-image of state q_{i+1} . Since state $q_i = 101$ is a subset of state $q_j = 1X1 = \{101\} \cup \{111\}$, then the states q_{j-1} and q_{j+1} must be a pre-image and an image of q_i also, respectively. The three edges added in this scenario are drawn as dashed lines in Figure 5 (b).

Our overall trace compaction technique using reachability analysis is shown in Figure 6. Lines 1-7 set up the problem, build the initial graph G and determine the initial trace length. The remaining lines perform reachability analysis by selecting a state for pre-image computation (line 10), computing the pre-images (line 12), and applying the state containment rules (line 14). The reachability analysis is terminated after all states have been selected for pre-image computation or after a maximum, max , number of steps have been performed determined by the counter.

B. Creating More Short-cuts

As discussed in the previous section, the containment rules are critical for creating short-cuts in the graph G . To increase the likelihood of applying these rules, the reachability engine is slightly modified from its typical UMC application. Traditionally in UMC, reachability engines focus on finding only new states and “block” previously visited states [8]. This allows them to quickly identify

```

1:  $G = \emptyset$ 
2:  $Visited = \emptyset$ 
3:  $counter = 0$ 
4: for all (states  $q_i$  between  $q_0$  to  $q_k$  (inclusive))
   do
5:    $Visited.add(q_i)$ 
6:    $G = add\_to\_graph(q_i)$ 
7: end for
8:  $length = BFS(G, q_k, q_0)$ 
9: while ( $counter \leq max$  &&  $!Visited.empty()$ )
   do
10:   $q_j = select\_state(Visited)$ 
11:   $Visited = Visited - q_j$ 
12:   $PreImages = pre\_image(q_j)$ 
13:  for all (states  $q_i \in PreImages$ ) do
14:     $apply\_rules1.2.3(G, q_i, q_j)$ 
15:  end for
16:   $Visited = Visited \cup PreImages$ 
17:   $counter = counter + 1$ 
18:   $length = BFS(G, q_k, q_0)$ 
19:   $Print(Trace\ is\ of\ size\ length)$ 
20: end while
21: return  $length$ 

```

Fig. 6. Trace compaction procedure using reachability analysis

when a fixed-point is reached, or when all legal states are visited [9]. In contrast, this work encourages finding previous states or states that cover or are covered by others. These containment relationships allow us to draw additional edges between nodes and increasing the likelihood of reducing the trace. It should be noted that precautions are taken to avoid repeatedly visiting the same set of states.

A second technique used to increase the likelihood of applying the containment rules is to populate the graph with more states than those provided in the original trace. Since the original trace only has as many states as its trace length, there may not be enough unique states to create many short-cuts. We propose populating the graph initially by computing a single pre-image for the states in the original trace. This approach allows us to quickly add state cubes to the graph which leads to more applications of the containment rules. The practical advantage of this technique is highlighted in the experiments of Section V.

C. State Selection Procedure

During reachability analysis, which state is selected for pre-image computation determines the manner in which the state space is traversed. For instance, if the most recently visited (found) state is always selected, then the state space is traversed in a depth-first manner. Here, we develop state selection criteria that help guide the reachability engine towards finding short-cuts from the initial state to the final state. It should be noted that these criterias are heuristics which may not always be advantageous.

The first criteria is to select a candidate state from the set of visited states with the smallest *hamming distance* to the initial state q_0 . The hamming distance between two states is the number of state variables with different values (0 or 1). For states with don’t cares (X), every X matches both the 0 and 1 value. For instance, if states $\{1100, 1011, 110X, XX01\}$ are visited and $q_0 = 0000$, then state $XX01$ is selected since it has a hamming distance of 1 with respect to q_0 . The intuition behind the above criteria is that states with a smaller hamming distance to q_0 require less state variables to change to reach q_0 as a pre-image. Therefore, the likelihood of finding q_0 at the next step may be higher.

A second factor that influences the state selection procedure is the path length from a candidate state to the last state q_k . If this length is greater than 50% of the current shortest path from q_0 to q_k then the state is not considered for selection. This criteria encourages finding many pre-images near the end of the trace (closer to q_k) and less closer to the initial state. Together, both criterias increase the probability of creating large short-cuts between states at the two ends of the original trace.

D. All-Solution SAT Solver

The reachability engine is highly dependent on the performance of the pre-image computation engine, which is based on an all-solution SAT solver. This SAT solver uses circuit don't cares to determine whether variables may remain unassigned while satisfying the problem [10], [12]. Since the don't cares are propagated backwards through a gate (from output to input) they are ideal for pre-image computation where current state variables V can be viewed as pseudo inputs to the circuit. The all-solution SAT solver contains many solution reduction techniques to ensure that small solutions are returned in an efficient manner [8], [9], [10]. For our application, achieving small state cubes is critical to traversing the state space efficiently.

Each pre-image computation step corresponds to a call to the all-solution SAT solver. Since it may not be practical to find all of the pre-image states due to the exponential nature of the problem, the all-solution SAT solver is also equipped with a limit t . If all the pre-image state cubes are not found in a time and memory efficient manner, the all-solution SAT solver will return the first t state cubes it finds. This allows us to perform reachability analysis by finding partial pre-images.

IV. STORING VISITED STATES

The success of the reachability analysis approach described in Section III depends on the ability to quickly apply the rules of Section III-A. More specifically, the situations where a newly found state q_i 1) is equal to existing states, 2) is a superset of existing states, or 3) is a subset of existing states must be rapidly identified. In this section we introduce a data structure that stores all the states belonging to G while identifying the state containment relationships quickly. Note that this data structure is not only viable for trace compaction, but can also be used for reachability analysis within a UMC framework [9], [8], [11].

A. Determining State Containment Relationships

The data structure described here is composed of two components 1) a binary tree T and 2) a hash table. The binary tree is used to detect the state containment relationships, while the hash table is used to locate the exact state.

The state containment relationship depends on the number of don't cares in each state. A state with more don't cares may cover one with fewer, while the converse is not true irrespective of the actual position of the don't cares. To take advantage of the above, we allocate an *ordered cube* for each state. The ordered cube is defined as the state value with all the zeros in the most significant positions, followed by all ones, followed by the don't cares (X) in the least significant positions. For example, five states and their corresponding ordered cubes are shown below.

states	1101X	001X1	XX001	X00X1	X11XX
ordered cube	0111X	0011X	001XX	001XX	11XXX

When states are added to the graph G , they are also stored according to their ordered cube in the binary tree T . Each node of a given depth in the binary tree corresponds to a position in the ordered cube. The top-most node at depth zero of the tree represents the most significant position, the nodes at depth 1 represent the second most significant position, the nodes at depth 2 represent the third most significant position, etc. The left (right) edge of a node denotes a zero (one) in the ordered cube at the position corresponding to the parent node. There are no edges corresponding to a don't care in the ordered cube. By scanning over the values of an ordered cube from the most significant to the least significant, the binary tree is traversed for that cube. Traversal ends when the ordered cube is fully scanned or when a don't care is encountered. By the end of the traversal, the final visited node points to a hash table where the state value is stored.

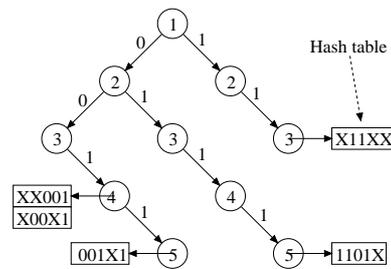


Fig. 7. Illustrating state storage data structure

The hash table contains all states that map to the same ordered cube. For instance, at the node corresponding to the ordered cube 001XX in Figure 7, there can be two unique state cubes XX001 and X00X1. Figure 7 illustrates how the states 1101X, 001X1, XX001, X00X1, X11XX are stored in the described data structure.

Given a state q_i , this data structure can efficiently determine whether q_i already exists in G , whether q_i is a subset of other states in G , and whether q_i is a superset of other states in G . For all three tasks, first the node n_i corresponding to ordered cube of q_i must be located in the binary tree. If q_i exists in the hash table pointed by node n_i , then q_i already exists in G .

To find whether q_i is a proper subset of other states, all the nodes with at least as many don't cares (X) as n_i have to be visited. At each node, the states within the hash tables must be tested to determine if q_i is a subset. Within the tree T , the nodes with at least as many don't cares as n_i are found inside an $(r+1) \times (s+1)$ rectangle, where r is the number of zeros and s is the number of ones in q_i . Therefore, there are $(r+1) \times (s+1)$ nodes that can potentially contain supersets of q_i (including node n_i). These nodes are illustrated in the dashed rectangle above node n_i in Figure 8.

Similarly, to find whether q_i is a proper superset of other states, all the nodes with at least as many zeros and ones must be visited and the states within the hash tables must be tested to determine if q_i is a superset. Within the tree T , these nodes are found inside an isosceles triangle with equivalent sides $n - r - s$. Therefore, there are $\frac{(n-r-s)(n-r-s+1)}{2}$ nodes that can potentially be subsets of q_i (including node n_i). These nodes are illustrated in the dashed triangle under the node n_i in Figure 8.

As demonstrated through Figure 8, only the white nodes must be considered when searching for subsets and supersets. Therefore, the number of comparisons required may be only a fraction of the total number of existing states. In practice, this data structure is found to be very efficient since the tree T is often not fully populated and the number of items in each hash table is relatively small.

The procedure for finding the supersets (covers) of a given state q_i is presented in Figure 9. Lines 2-3 generate the ordered cube and find its location in the tree T . Line 4 gets all the potential superset

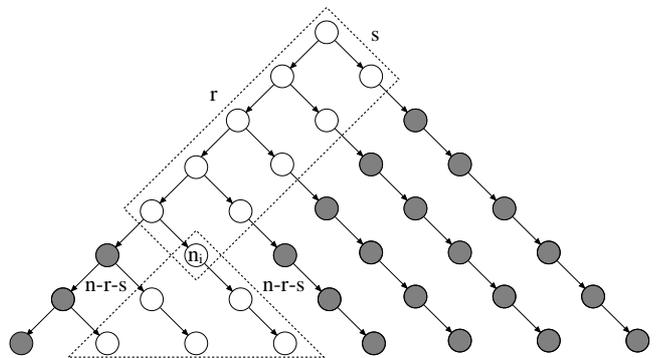


Fig. 8. Finding supersets and subsets in the tree T

```

1: Covers =  $\emptyset$ 
2: ordered_cube = Order( $q_i$ )
3:  $n_i$  = Get_tree_node(ordered_cube)
4: Supset = get_rectangle( $n_i$ )
5: for all (nodes  $n_j$  in Supset) do
6:   for all (states  $q_j$  in hash table of  $n_j$ ) do
7:     if ( $q_j \supseteq q_i$ ) then
8:       Covers = Covers  $\cup$   $q_j$ 
9:     end if
10:  end for
11: end for
12: return Covers

```

Fig. 9. Determine the states that are supersets of this state

nodes by finding the nodes contained in the rectangle. The remaining lines iterate through these nodes and test the states inside the hash tables to determine whether they are supersets of q_i . Note that testing whether a particular node is a superset or a subset of another is a simple comparison procedure where the states must be identical over all positions except where the superset is a don't care. A procedure similar to that of Figure 9 is used to find the subsets of q_i where the *get_rectangle* procedure is replaced with *get_triangle* as described previously.

V. EXPERIMENTS

In this section we demonstrate the effectiveness and efficiency of the proposed trace compaction approach. All experiments are conducted on a Sun Blade 1000 with a 750MHz Sparc processor and 2.5GB of memory. Traces of length 50, 100, and 1000 are obtained via random simulation for the circuits in the ISCAS'89 and ITC'99 benchmark suites. The reachability analysis engine is developed using the all-solutions SAT solver of [10] which is a circuit variant of zChaff [7] and Grasp [13]. To evaluate the overall proposed approach we limit the number of stored states to at most 10,000 state *cubes* and do not use an explicit timeout. Since the compaction techniques of previous works [3], [4], [6] are not publicly available and due to the fact that the assertions and errors used are unknown, we cannot directly or indirectly compare with them.

We first evaluate the effectiveness of the state selection procedure described in Section III-C. We compare this heuristic against three other selection approaches, Depth-First Search (DFS), Breadth-First Search (BFS), and random selection. The above techniques are used to perform reachability analysis from a random state to the initial state given a timeout of 200 seconds. The runtimes over all the benchmarks are collected and presented in Figure 10. Both the DFS and BFS methods result in runtimes of over 4000 seconds, while the random method fares better at over 3500 seconds. The proposed state selection strategy based on the smallest hamming distance relative to the initial state and the position of the state in the graph G results in runtimes of just over 3000 seconds. This performance demonstrates that the proposed state selection heuristics is an efficient overall reachability analysis procedure.

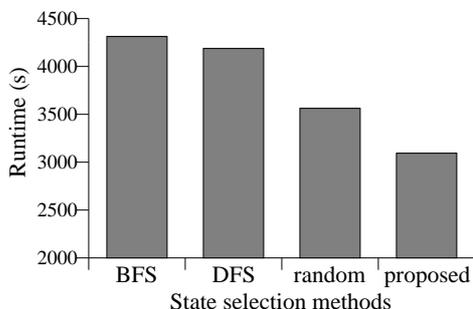


Fig. 10. Comparison of state selection methods

Next, we demonstrate the effectiveness of the overall proposed trace compaction approach. Table I illustrates the results of the

experiments on all ISCAS'89 and ITC'99 circuits for traces of length 50, 100 and 1000. The first column shows the circuit names while the remaining columns are organized into three sections based on their original trace length. The first column of each section labeled *org* describes the original length of each trace (50, 100, or 1000). The second column of each section labeled *pre* describes the length of the traces after performing the single step pre-image process described in Section III-B. We chose to find single step pre-images for no more than 50 states to achieve a balance between the number of pre-images found and the time required to find them. The third column of each section labeled *reach*, presents the length of the traces after applying the proposed reachability analysis method. As described in section III-B, it is most beneficial to first find the single step pre-images followed by the reachability analysis (*reach*) method. The fourth and fifth columns of each section, labeled *cpu pre* and *cpu reach* respectively, present the runtimes in seconds associated to the *pre* and *reach* techniques.

Table I shows that the pre-image computation techniques help reduce the traces considerably. For many circuits, the original trace length is first reduced greatly by the single step pre-image (*pre*) technique and further reduced by the reachability analysis (*reach*). For example, the trace for circuit s344 is first reduced from 50 to 33 using *pre*, and then again from 33 to 1 using *reach*.

Analyzing the results of Table I, we notice that many traces are reduced to having a single clock cycle (length of 1) or a very small trace size after applying reachability analysis. This result can be partially attributed to the state selection heuristics of Section III-C and the performance improvement techniques of Section III-B. These techniques can increase the number of "short-cuts" created through the graph G and likelihood that they will lead to the initial state.

Table II summarizes the results in Table I by providing the average length compactions (reductions) achieved by the different components of the proposed approach for traces of size 50, 100, and 1000. Similar to Table I, the summaries are provided for each original trace length separately. Column one presents the name of the compaction method: single step pre-image computation (*pre*), reachability analysis (*reach*), or combined. For each trace length, the overall average reduction is presented under the label *avg. reduced*. This field is calculated by adding the reduction in size over all circuits divided over the number of circuits. Since not all circuit traces are reduced by the proposed method, this number may not provide a good representation of the average factor of reduction achieved. Instead, the columns labeled *affected* and *reduced* show the percentage of traces that are affected by each approach and the amount by which they are reduced, respectively. For example, for traces of length 50, the proposed approaches separately achieve 10.08 times and 3.81 times reductions while the combined approach reaches 19.67 times reductions. Furthermore, approximately 70% of the circuits are affected by the *pre* techniques which results in an average reduction of 13.77 times. Similarly, the *reach* technique and the combined approach affect 37% and 74% of traces for a reduction of 8.45 times and 25.72 times, respectively.

The experimental results demonstrate that not only is the proposed approach effective for reducing traces, but it is also very efficient. For the majority of circuits in Table I, compacted traces are found within a few minutes. This performance reaffirms the practicality of the data structure introduced in Section IV. The memory requirements of the overall approach are also manageable since memory usage never exceeds 300MB when storing up to 10,000 state cubes. The ability to quickly reduce traces in a memory efficient manner is crucial for making this approach viable in real-life debugging environments.

VI. CONCLUSION

This work proposed a novel trace reduction technique using SAT-based reachability analysis and a set of state containment relationships. The components of the reachability analysis engine are fine-tuned to increase the likelihood of generating short-cuts in the original

circuits	org	pre	reach	cpu pre	cpu reach	org	pre	reach	cpu pre	cpu reach	org	pre	reach	cpu pre	cpu reach
s208.1	50	25	25	0.00	0.56	100	51	51	0.07	0.60	1000	244	244	0.08	9.26
s298	50	1	1	0.00	0.00	100	3	1	0.59	0.86	1000	1	1	0.34	0.01
s344	50	33	1	0.00	0.00	100	55	1	0.31	0.00	1000	10	5	0.42	0.08
s349	50	33	1	0.00	0.00	100	55	1	0.32	0.00	1000	10	5	0.39	0.08
s382	50	3	1	0.00	0.17	100	4	2	0.75	0.00	1000	1	1	0.89	0.00
s386	50	1	1	0.00	0.00	100	2	2	0.09	0.00	1000	2	2	0.06	0.00
s400	50	3	1	0.00	0.01	100	2	1	0.69	0.01	1000	2	1	0.74	0.05
s420.1	50	21	21	0.01	1.20	100	44	44	0.13	0.97	1000	505	505	0.14	25.85
s444	50	2	1	0.01	0.01	100	3	1	0.98	0.93	1000	1	1	0.67	0.01
s510	50	24	24	0.00	0.87	100	10	10	0.13	0.66	1000	25	25	0.12	0.56
s526	50	2	1	0.00	0.03	100	3	1	1.27	0.86	1000	1	1	1.09	0.03
s526n	50	2	1	0.00	0.03	100	3	1	1.26	0.86	1000	1	1	1.17	0.02
s641	50	3	3	0.00	1.65	100	4	4	1.81	2.10	1000	2	2	1.72	5.86
s713	50	3	3	0.00	1.65	100	4	4	1.80	2.01	1000	2	2	1.76	2.88
s820	50	1	1	0.00	0.00	100	1	1	0.00	0.00	1000	1	1	0.38	0.00
s832	50	1	1	0.00	0.00	100	1	1	0.00	0.00	1000	1	1	0.4	0.00
s838.1	50	26	26	0.00	1.87	100	45	45	0.26	2.07	1000	510	510	0.27	48.48
s953	50	6	5	0.00	1.38	100	1	1	2.52	0.00	1000	1	1	3.25	0.01
s1196	50	8	1	0.00	0.05	100	14	1	0.89	0.12	1000	5	1	1.11	0.03
s1238	50	8	1	0.01	0.05	100	14	1	0.84	0.11	1000	5	1	0.96	0.02
s1423	50	50	2	0.01	3.41	100	57	2	6.19	3.55	1000	15	3	6.24	67.61
s5378	50	50	50	0.04	0.89	100	100	100	23.76	1.03	1000	1000	1000	26.18	5.86
s9234.1	50	50	50	0.04	22.67	100	100	100	50.26	1.76	1000	1000	1000	49.89	11.55
s9234	50	34	34	0.02	1.67	100	36	36	46.99	1.66	1000	35	35	47.41	10.76
s13207.1	50	50	50	0.28	3.52	100	100	100	96.76	4.20	1000	1000	1000	105.92	7.61
s13207	50	50	50	0.23	3.29	100	100	100	91.57	4.17	1000	1000	1000	98.79	7.74
s15850.1	50	50	50	0.12	5.82	100	100	100	145.67	87.18	1000	1000	1000	140.31	9.01
s15850	50	50	50	0.07	3.45	100	100	100	96.18	4.19	1000	1000	1000	222.94	8.09
s38417	50	50	50	1.07	40.58	100	100	100	311.05	154.30	1000	1000	1000	340.83	25.74
s38584.1	50	50	50	1.27	11.83	100	100	100	336.97	12.37	1000	1000	1000	375.70	25.68
s38584	50	50	50	1.26	59.15	100	100	100	315.11	185.30	1000	1000	1000	344.44	23.85
b01	50	6	2	0.09	0.00	100	4	4	0.10	0.04	1000	4	4	0.9	0.04
b02	50	2	2	0.04	0.00	100	4	4	0.04	0.01	1000	4	4	0.4	0.01
b03	50	14	2	1.17	0.07	100	26	2	1.20	0.05	1000	8	8	1.32	13.54
b04	50	50	50	6.57	3.79	100	100	100	6.26	4.54	1000	1000	1000	6.89	27.88
b06	50	3	1	0.65	0.00	100	3	3	0.63	0.04	1000	2	2	0.62	0.03
b07	50	43	43	0.35	1.81	100	51	51	0.34	1.70	1000	56	56	0.28	14.56
b08	50	43	7	0.11	1.17	100	92	2	0.16	0.00	1000	329	5	0.16	0.02
b09	50	50	17	0.16	0.96	100	97	97	0.17	1.56	1000	82	82	0.18	18.70
b10	50	22	22	0.36	1.19	100	45	21	0.31	1.56	1000	32	32	0.59	9.56
b11	50	35	25	1.44	3.06	100	98	88	2.50	4.07	1000	550	550	1.92	27.68
b12	50	14	14	8.03	5.97	100	20	20	7.51	2.50	1000	36	36	7.89	34.85
b13	50	45	45	2.10	2.22	100	99	98	2.05	2.80	1000	1000	1000	2.57	19.54
b14	50	50	50	42.48	1.84	100	100	100	47.68	24.18	1000	1000	1000	52.92	3.93
b15	50	49	49	76.63	6.19	100	100	100	56.65	43.78	1000	87	87	52.11	230.41

TABLE I

EXPERIMENTAL RESULTS FOR THE PROPOSED TRACE LENGTH COMPACTION APPROACH FOR TRACES OF LENGTH 50, 100 AND 1000.

approach	original size 50			original size 100			original size 1000		
	avg. reduced	affected	reduced	avg. reduced	affected	reduced	avg. reduced	affected	reduced
pre	10.08 X	70 %	13.77 X	16.88 X	72 %	22.66 X	266.35 X	71 %	362.84 X
reach	3.81 X	37 %	8.54 X	6.10 X	35 %	15.36 X	2.77 X	15 %	12.40 X
combined	19.67 X	74 %	25.72 X	36.21 X	72 %	49.01 X	327.76 X	72 %	446.59 X

TABLE II

SUMMARY OF THE RESULTS FOR THE PROPOSED TRACE LENGTH COMPACTION APPROACH

trace. Furthermore, a novel data structure is presented which stores visited states such that the state containment relationships can be quickly applied. Experiments demonstrate the effectiveness of the proposed techniques as approximately 75% of the traces are reduced by one or two orders of magnitude.

REFERENCES

- [1] P. Rashinkar, P. Paterson, and L. Singh, *System-on-a-chip Verification: Methodology and Techniques*. Kluwer Academic Publisher, 1996.
- [2] T. Kropf, *Introduction to Formal Hardware Verification*. Springer, 1999.
- [3] Y. Chen and F. Chen, "Algorithms for compacting error traces," in *ASP Design Automation Conf.*, 2003, pp. 99–103.
- [4] S. Shen, Y. Qin, and S. Li, "A faster counterexample minimization algorithm based on refutation analysis," in *Design, Automation and Test in Europe*, 2005, pp. 672–677.
- [5] K. Chang, V. Bertacco, and I. Markov, "Simulation-based bug trace minimization with BMC-based refinement," in *Int'l Conf. on CAD*, 2005, pp. 1045–1051.
- [6] S.-J. Pan, K.-T. Cheng, J. Moondanos, and Z. Hanna, "Generation of shorter sequences for high resolution error diagnosis using sequential sat," in *ASP Design Automation Conf.*, 2006, pp. 25–29.
- [7] J. Marques-Silva and K. Sakallah, "GRASP – a new search algorithm for satisfiability," in *Int'l Conf. on CAD*, 1996, pp. 220–227.
- [8] K. McMillan, "Applying SAT methods in unbounded symbolic model checking," in *Computer Aided Verification*, 2002, pp. 250–264.
- [9] H.-J. Kang and I.-C. Park, "SAT-based unbounded symbolic model checking," *IEEE Trans. on CAD*, vol. 24, no. 2, pp. 129–140, 2005.
- [10] S. Safarpour, A. Veneris, and R. Drechsler, "Integrating observability don't cares in all-solution SAT solvers," in *IEEE International Symposium on Circuits and Systems*, 2006, pp. 1587–1590.
- [11] B. Li, M. Hsiao, and S. Sheng, "A novel SAT all-solutions solver for efficient preimage computation," in *Design, Automation and Test in Europe*, 2004, pp. 272–277.
- [12] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "Sat sweeping with local observability don't-cares," in *Design Automation Conf.*, 2006, pp. 229–234.
- [13] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Design Automation Conf.*, 2001, pp. 530–535.