Improved SAT-based Reachability Analysis with Observability Don't Cares

Sean Safarpour   sean@eecg.toronto.edu
Andreas Veneris   veneris@eecg.toronto.edu
Department of Electrical & Computer Engineering,
University of Toronto, Canada

Rolf Drechsler   drechsle@informatik.uni-bremen.de
Department of Computer Science,
Bremen University, Germany

Abstract

The dramatic performance improvements of SAT solvers over the past decade have increased their deployment in hardware verification applications. Many problems that were previously too large and complex for SAT techniques can now be handled in an efficient manner. One such problem is reachability analysis, whose instances are found throughout verification applications such as unbounded model checking and trace reduction. In circuit-based reachability analysis important circuit information is often lost during the circuit-to-SAT translation process. Observability Don't Cares (ODCs) are an example of such information that can potentially help achieve better and faster results for the SAT solver. This work proposes to use the ODCs to improve the quality and performance of SAT-based reachability analysis frameworks. Since ODCs represent variables whose values do not affect the outcome of a problem, it is possible to satisfy a problem with fewer assigned variables. This in turn leads to more compact solutions and thus fewer solutions to cover the entire solution space. Specifically, this work presents an efficient way to identify ODCs, proves the correctness of leaving ODC variables unassigned, and develops a reachability analysis platform that benefits greatly from the ODCs. The advantages of using ODCs in reachability analysis is demonstrated through extensive experiments on unbounded model checking and trace reduction applications.

KEYWORDS: *SAT solver, reachability analysis, model checking, observability don't cares*

Submitted January 2007; revised April 2008; published June 2008

1. Introduction

In today's VLSI design cycle, verification is a major bottleneck as it often consumes over 70% of the design effort [26]. Due to the ever-increasing size and complexity of designs, the effort and costs associated with verification are continuously increasing. Despite the countless advancements in the field, new verification methodologies and techniques are required to alleviate some of the pain.

Reachability analysis is a popular procedure employed by verification applications such as model checking and trace reduction [20, 31]. The efficiency of a reachability analysis engine directly impacts the end application. Informally, reachability analysis determines

whether a given state can be reached by another state in a circuit. Traditionally, Binary Decision Diagram (BDDs) have been the underlying engine of choice for reachability analysis. However, for certain problems, building and manipulating the BDDs require excessive memory which sometimes makes the overall approach inviable. Recent improvements of SAT algorithms offer a competitive alternative for such problems [20, 15].

A slight derivative of the conventional SAT solver, namely the *all-solution* SAT solver, is found to be an effective engine for reachability analysis because it finds many reachable states in one iteration [9]. As the name implies, all-solution SAT solvers find all the satisfying solutions to a SAT problem. The main challenge of all-solution SAT solvers is to find satisfying solutions comprised of few variable assignments, also known as small solutions cubes [20]. A small solution cube has unassigned variables whose exact values do not affect the outcome of a problem. As such, a single solution cube can effectively represent multiple larger solution cubes. For all-solution SAT solvers, small solution cubes require less memory for storage as well as demanding a smaller aggregate run time for the overall problem. Thus the efficiency of an all-solution SAT solver and the reachability analysis engine is directly related to the ability of finding small solution cubes in a timely manner.

This work proposes an improved SAT-based reachability analysis engine which makes use of the inherent circuit Observability Don't Cares (ODCs) of a given problem [1.]. ODCs are widely used in CAD algorithms such as circuit synthesis and test to improve the quality and efficiency of the overall tools [1, 6]. They provide extra degrees of freedom by not constraining certain variables to any particular logic value.

For the majority of circuit-based SAT problems, extracting the ODCs from the circuit can be performed with relative ease during the circuit-to-SAT translation process. Once available, this information can be used dynamically during the SAT solving process to prevent logic value assignments to ODC variables, which do not affect the outcome of the SAT problem. Subsequently, solutions to the SAT problem will contain unassigned variables thus leading to small solutions cubes.

To demonstrate the correctness of the proposed approach, this work proves that leaving ODC variables unassigned does not affect the outcome of the problem. Furthermore, a novel decision-making procedure is proposed that takes advantage of the ODCs to find even smaller solution cubes. Finally, the benefits of the observability don't cares are demonstrated within reachability analysis as well as two end applications, namely Unbounded Model Checking (UMC) and trace reduction.

The effectiveness of ODCs for reachability analysis applications are demonstrated using a set of different applications. First, the performance of the all-solution SAT solver is analyzed in isolation of the end application to demonstrate the effectiveness of the proposed procedures. Next, a UMC framework is used to show the effect of ODCs on the end application. Finally, the proposed techniques are evaluated on trace reduction problems. The experiments overwhelmingly demonstrate that ODCs are beneficial for these applications, with performance improvements of approximately $4\times$ over previous approaches for UMC and orders of magnitude greater for trace reduction.

In the next section, some related work on ODC-based SAT solvers and reachability analysis approaches is presented with the major differences with our work outlined. Sec-

---

1. This work builds on top of the advancements of [27, 28, 29]

tion 3 provides background information on image computation and observability don't cares. Section 4 presents Theorem 1 stating that ODC variables can be left unassigned without affecting the outcome of the SAT problem. This theorem is then used to reduce the size of the SAT solution cubes. Section 5 presents a decision-making procedure that can further reduce solution cubes as required by reachability analysis problems. In Section 6, the application domains for the experiments are presented with the results discussed in Section 7. Finally, Section 8 concludes this paper.

2. Related Work

This paper builds on top of the ODC advancements of [27, 28, 29]. In [28], Observability and Controllability Don't Cares are used to improve the run time of SAT-based combinational equivalence checkers. In [27], ODCs are applied to all-solution SAT solvers, while [29] evaluates ODCs for trace reduction applications. The presented work encompasses the above, proves its correctness, and demonstrates the benefits of ODCs in general for reachability analysis.

2.1 Incorporating ODCs with SAT Solvers

Improving the efficiency of SAT solvers with circuit observability don't cares has been explored previously [8, 10, 35, 34, 36]. In [10], ODCs are used to mark clauses as *inactive* which are removed dynamically from the CNF. The inactive clauses are identified by performing a backward traversal of the CNF starting at the output variables in order to find the ODCs. The work of [35] exploits ODCs specifically for if-then-else trees by simplifying the resulting CNF of the problem. This approach is very effective for microprocessor circuits with datapaths which contain many multiplexers. The techniques introduced by [34, 8] recognize the benefits of ODCs as it incorporates this information directly in the problem CNF by introducing don't care literals. The work of [36] finds and exploits ODCs during SAT sweeping to simplify the problem representation.

The above work is a testament to the importance of considering ODCs in circuit-based SAT problems. The presented work is considerably different from the related work in the way ODCs are found as well as how they are exploited by an all-solution SAT solver within reachability analysis. Here, potential ODCs in the circuit are identified through a quick preprocessing step before the SAT solving begins. During the SAT solving process, the active ODCs (which are pre-computed) are simply ignored thereafter by SAT procedures such as decision-making and Boolean constraint propagation.

2.2 SAT-based Reachability Analysis

SAT solvers and all-solutions SAT solvers are found to be beneficial for certain model checking problems [9, 11, 13, 15, 18, 20]. Both [15] and [20] develop all-solution SAT frameworks without significantly modifying the internal SAT engines. In [20] an implication graph rooted at the primary inputs is used to generate small blocking clauses, while [15] uses traditional justification procedures [1]. Justification procedures use circuit information to determine the necessary gate assignment in order to justify other circuit assignments [1]. In [11], a reduction algorithm is developed to determine the necessary input assignments by

performing a forward traversal of the circuit from primary inputs to the outputs. In [9], the main contribution is the use of *cofactoring* to reduce the solution cubes for UMC problems while using the SAT solver from [10] which is aware of ODCs. A new all-solution SAT solver for UMC application is proposed in [18] with many specific contributions in the form of algorithms and data structures based on ATPG to help *quantify* primary inputs and prune the solution space. A hybrid SAT solver is proposed in [13] where the conflict analysis and implication routines are performed on the circuit and the CNF to achieve smaller blocking clauses. There is also dedicated work on reducing the size of solutions cubes or blocking clauses for solution enumeration. These work use techniques to generate blocking clauses that cover as much of the solution space as possible through the use of heuristics such as variable lifting (i.e., removal of variables that are irrelevant to the SAT problem for given variable assignment) [14, 22].

As discussed above, most of the related work on all-solution SAT solvers is concerned with reducing the size of the blocking clauses or the solution cubes. It should be noted that all solution cube reduction techniques implicitly make use of the problem's don't care space to achieve smaller cubes. In contrast, the proposed work is concerned with using a SAT solver that is explicitly aware of ODCs and makes decisions on the fly to take advantage of the ODCs. The result is a SAT solver that inherently finds small solution cubes even before applying the reduction techniques. Moreover, the proposed technique can incorporate many of the related work to provide further performance gains.

3. Preliminaries

This section provides some background on image and pre-image computation as well as circuit observability don't cares. It is assumed that the reader is familiar with SAT solving algorithms and their terminology [19].

3.1 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 state variables V to V', denoted by $T(V, V')$, is true for each pair of $Q(V)$ and $Q(V')$ if and only if $Q(V')$ is reachable from $Q(V)$ in one clock cycle. Given the above, the image and pre-image of a circuit are defined as follows:

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

Although the image and pre-image of circuits are traditionally computed using BDDs [5], some techniques based on all-solution Boolean Satisfiability (SAT) solvers can also be used [15, 18, 20, 27]. This work focuses on finding the pre-images efficiently using circuit-based SAT techniques. All-solution SAT solvers can compute all the pre-image sets by constraining the transition relation $T(V, V')$ to $Q(V')$ and iteratively finding every solution that satisfies the problem. These solutions are found with respect to the current state variables V [27]. Recent work on SAT-based Unbounded Model Checking (UMC) and pre-

image computation techniques have demonstrated considerable advancements in terms of capability and run time [15, 18, 20, 27].

In this paper, the term state refers 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 $\{v_1, v_2, v_3\} = \{111\}$. In the context of cubes, the term *small* refers to the number of assigned variables, thus a small cube may cover a larger cube.

3.2 Observability Don't Cares

Informally, a signal is an Observability Don't Care (ODC) if its value does not influence the output of a circuit. For the sake of simplicity, current state and next state variables are treated as primary inputs and outputs, respectively. In the following, the terms circuit line, signal, and variable are used interchangeably. The definitions provided below are common in the circuit testing community [12].

Definition 1. *In a combinational circuit, a signal is* unobservable *if assigning it a 0 or 1 logic value does not change the value of any primary output.*

Definition 2. *Given a circuit where some signals are assigned a 0 or 1 logic value, an Observability Don't Care is a signal that is* unobservable.

Definition 3. *Given a gate G and an input signal w, a* neighbor *of w is any input signal to gate G other than w.*

Definition 4. *A* controlling value *assignment is a logic value assignment to a gate input (fanin) such that all other inputs to the gate are unobservable.*
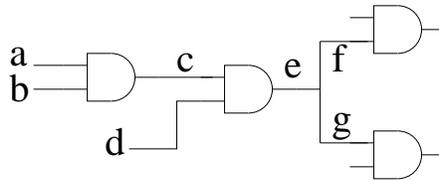
Figure 1. A two gate circuit

As an example, consider the circuit in Figure 1. When signal d is assigned a controlling value of logic 0, signal c becomes unobservable because its value does not influence the value of the output signal e. Since signal c is the fanout of the AND gate with fanins a and b, both fanins become unobservable too. In summary, under the assignment $d = 0$, signals a, b, and c are all observability don't cares.

In a circuit, the output of a multiple fanout gate can be represented by a common wire segment that forks into multiple wire segments that directly input the subsequent gates. The common segment is referred to as a *stem*, while the fanin segments are referred to as *branches*. For example, in Figure 1, wire segment e is a stem, while wire segments f

and g are branches. To simplify definitions, in the following, different variables are used to identify circuit stems and branches.

From Definition 4, logic value assignment to circuit lines can result in the creation of ODC signals under the following three distinct conditions:

Condition 1. When the fanin of a gate is assigned a controlling value (a 0 for a (N)AND gate and a 1 for (N)OR gate)[2], all other fanins excluding stems become ODCs.

Condition 2. When the fanout of any gate is an ODC, all of the fanins of the gate excluding stems (including branches) become ODCs.

Condition 3. When all branches of a stem are ODCs (due to condition 1 or 2), the stem itself becomes an ODC.

The first condition directly depends on the logic value assignment to circuit lines. The other two conditions depend on other ODC signals and may be implied by the first condition. In other words, when a line in the circuit is assigned a logic value, the first condition can determine what signals become ODCs, thus implying other signals to become ODCs as well. In the next section, a technique is developed that can efficiently test for these conditions and determine when variables become ODCs.

## 4. Managing Observability Don't Cares in SAT

Observability Don't Cares (ODCs) are signals or variables whose values do not affect the output of the circuit. Since ODCs can be ignored in circuits, they can also be ignored in SAT instances derived from them. Ignoring these variables can lead to run time savings by skipping procedures such as decision-making and Boolean Constraint Propagation (BCP). Moreover, leaving ODC variables unassigned by ignoring them can lead to smaller solution cubes. These benefits may not be applied to every circuit-based SAT problem blindly, as they may alter the problem under some conditions. This section presents conditions common to many circuit-based SAT problems under which ODC variables can be left unassigned and proves that the solution's correctness is maintained.

4.1 Structure of Circuit-based SAT Problems

In SAT-based verification problems derived from circuits such as combinational equivalence checking [28] and bounded model checking [3] a circuit is often augmented with extra constraints. These constraints typically restrict the value assignments to the primary inputs, primary outputs, and state variables. In this work, these variables are referred to as *boundary* signals or variables while all other variables are called *internal* signals or variables. Typically, to construct a SAT problem, the given circuit is converted to CNF [17, 25, 33, 35] while constraints on the boundary signals are added using extra clauses. Figure 2 illustrates this construction for circuit-based problems.

---

2. The presented technique applies to more complex gate types. However, since other logic gate types such as multiplexers can be constructed from (N)AND and (N)OR gates, other gate types are not discussed in this work.
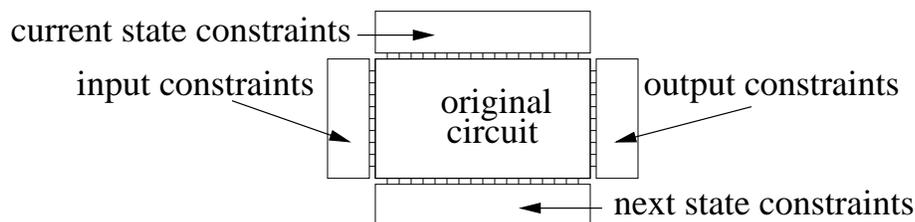
Figure 2. Example of a circuit-based SAT problem

Since most circuit-based verification problems are derived in the manner described above, the analysis in this paper is restricted to CNF instances generated from such problems.

4.2 Value Assignment to ODC Variables

By definition, ODC variables do not affect the output values of circuits. In circuit-based SAT problems, constructed as described in Section 4.1, these variables should not affect the outcome of the SAT problem either. More specifically, leaving ODC variables unassigned in a SAT problem should not change the outcome of the problem. Theorem 1 states this for DPLL-based SAT solvers [19].

The following assumptions, referred to as the *ignore assumptions*, and the subsequent lemmas are needed to prove Theorem 1.

Assumption 1. The problem CNF must be derived from a gate-level circuit representation as described in 4.1.

Assumption 2. Additional clauses in the problem CNF are allowed only for:

(a) *Redundant clauses*: clauses providing information already contained in the problem CNF such as conflict clauses found by conflict analysis procedures [19].

(b) *Boundary unit clauses*: clauses containing a single literal representing a 0 or 1 constraint on a boundary variable[3.].

(c) *Boundary constraint clauses*: clauses containing only boundary variables to prevent combination of logic value assignments from occurring on all or a subset of these variables.

Assumption 3. In this context, a variable is identified as an ODC if and only if it is unassigned, is not constrained by a unit clause, and at least one of the ODC conditions of Section 3.2 holds for the corresponding signal.

Lemma 1. A SAT problem derived from a gate-level circuit as outlined in [17] without any extra clauses in its CNF is SATISFIABLE.

---

3. Note: these clauses may be removed through BCP; However, we do not assume that BCP is run

Proof. For any circuit, simulating an arbitrary input vector (assignments of 0 or 1 values to all primary input and current state variables) on the circuit problem will result in all circuit lines to take well defined 0 or 1 values [12]. Since circuit simulation assigns values to signals according to the circuit topology (from input to output), every signal is assigned a Boolean value according the the gate driving it and the fanins of that gate. Thus simulation ensures that all signals from primary inputs to primary outputs are assigned a Boolean value that respects the implications of each and every gate. As a result, there cannot be any conflicts in the assignments made by the simulation process. Thus, the corresponding value assignment to the circuit CNF must be SATISFIABLE.

□

Lemma 2. In a SAT problem derived from a gate-level circuit, internal ODC variables do not propagate logic values between the input and output (boundary) variables. That is, internal ODC variables do not accommodate implications from the inputs to the outputs or from the outputs to the inputs.

Proof. Consider any internal variable v that is an ODC under one of the conditions of Section 3.2. Variable v can be assigned any logic value that is implied by its transitive fanin. However, since variable v is an ODC and its output is unobservable through any primary outputs by definition, the effect of its value does not imply any values on the fanout(s) of v including the primary outputs.

□

Theorem 1. *Given a CNF problem which is an instance of a circuit SAT problem that satisfies the* ignore *assumptions, a DPLL-based SAT solver can leave the ODC variables unassigned without affecting the outcome of the SAT problem.*

Proof. This proof uses contradiction. Assume that ignoring the ODC variables (i.e., leaving them unassigned) leads to an incorrect outcome of the SAT solver. There are two scenarios under which this can occur:

> **Case 1.** The problem is UNSATISFIABLE but by ignoring the ODC variables the SAT solver returns SATISFIABLE.

> **Case 2.** The problem is SATISFIABLE but by ignoring the ODC variables the SAT solver returns UNSATISFIABLE.

Each case is proved to be unrealizable below.

> **Case 1. The problem is UNSATISFIABLE but by ignoring the ODC variables the SAT solver returns SATISFIABLE.**
> Since a CNF derived from a circuit is SATISFIABLE by Lemma 1, extra clauses added to the CNF problem relating to the boundary variables (under the *ignore assumptions*) must generate constraints such that the problem is UNSATISFIABLE. In other words, the combination of the constraints from the extra clauses containing boundary variables and the problem structure constraints make the SAT problem UNSATISFIABLE. However, in this case ignoring ODC variables changes the outcome of the SAT problem to SATISFIABLE.

Given that all ODC variables are identified according to the *ignore assumptions* and that the logic value of internal ODC variables does not propagate to the boundary signals by Lemma 2, ignoring ODC variables does not affect the logic value of the boundary variables. As a result, the constraints making the problem UNSATISFIABLE are unaffected and the SAT solver *cannot* return SATISFIABLE.
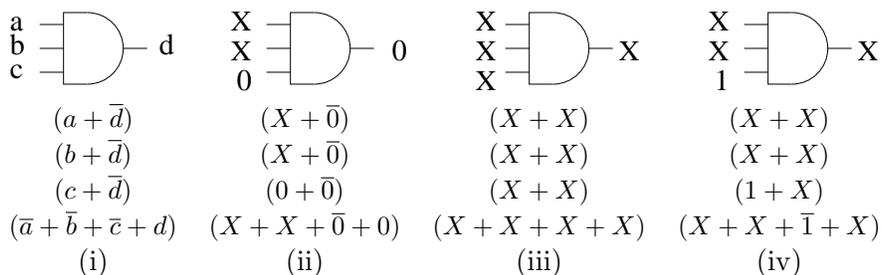
Case 2. The problem is SATISFIABLE but by ignoring the ODC variables the SAT solver returns UNSATISFIABLE.
To show that ODC variables do not affect the outcome of a SATISFIABLE instance of a circuit-based SAT problem, all CNF clauses containing ODC variables must be shown to be satisfiable. Recall that in the CNF format all clauses are in conjunction and each one must be satisfied for the overall CNF formula to be satisfied. The following situations cover all possible clause types allowed in the CNF problem under the ignore assumptions. Figure 3 shows an example of a three input AND gate, as part of a larger circuit, with inputs a, b, c and output d. The gates and the CNF in figures 3 (ii), (iii), and (iv) help illustrate situations 1, 2, and 3 below. In these figures, "X" denotes a line with an ODC variable.

1. Clauses representing a gate where one or more, but not all, inputs are ODCs and the output is not an ODC. This situation occurs only under *Condition 1* of Section 3.2 where at least one fanin of the gate is assigned a well-defined logic controlling value (*i.e.,* 0 for (N)AND gate and 1 for an (N)OR gate). As a result of the controlling value, all clauses derived from that gate are satisfied, also shown in Figure 3 (ii).

2. Clauses representing a gate where all inputs are ODCs. This situation occurs only under *Condition 2* where the fanout of the gate is identified as an ODC. As a result, there exists a logic assignment to each variable consistent with the characteristic function of the gate such that all the clauses are satisfied. This assignment must also be consistent with the values of variables in the transitive fanin and fanout of the gate, which can be determined through BCP. In the example of Figure 3 (iii), any assignment in agreement with the characteristic function of an AND gate and the transitive fanin and fanout values will satisfy all clauses.

3. Clauses representing a gate where the output is an ODC but one or more input is not. This situation occurs when some of the fanin stems are not ODCs or when some of the fanins are not unassigned. In this situation, there exists a logic assignment to each variable which is consistent with the characteristic function of the gate and all previously assigned variable such that all the clauses are satisfied. This assignment must also be consistent with the values of variables in the transitive fanin and fanout of the gate, which can be determined through BCP. In the example of Figure 3 (iv), any assignment consistent with the characteristic function of an AND gate with respect to the assigned variables will satisfy all clauses.

$$(a + \overline{d}) \quad (X + \overline{0}) \quad (X + X) \quad (X + X)$$
$$(b + \overline{d}) \quad (X + \overline{0}) \quad (X + X) \quad (X + X)$$
$$(c + \overline{d}) \quad (0 + \overline{0}) \quad (X + X) \quad (1 + X)$$
$$(\overline{a} + \overline{b} + \overline{c} + d) \quad (X + X + \overline{0} + 0) \quad (X + X + X + X) \quad (X + X + \overline{1} + X)$$
$$\text{(i)} \qquad\qquad \text{(ii)} \qquad\qquad \text{(iii)} \qquad\qquad \text{(iv)}$$

Figure 3. CNF satisfiability with lazy variables

4. Redundant clauses. Since these clauses provide information that is available through other clauses, any variable assignments that satisfy the other clauses will also satisfy these redundant clauses.

5. Boundary unit clauses. All unit clauses are assigned a satisfying 0 or 1 value and are not identified as ODCs by the ignore assumptions.

6. Boundary constraint clauses. There exists assignments to boundary ODC variables that satisfy the above 5 types of clauses. If none of these assignments can satisfy the boundary constraint clauses then the problem is UNSATISFIABLE with or without ODC variables. Otherwise, there exists and assignment to all variables such that the problem is SATISFIABLE in either case.

All cases are shown to be unrealizable thus contradicting the statement that unassigned ODC variables may result in an incorrect SAT outcome.

□

It has been shown that ODC variables can remain unassigned in a DPLL-based SAT solver if the ignore assumptions and the ODC conditions are respected. The next step is to determine how SAT solvers can benefit from unassigned ODC variables.

4.3 Benefiting from Unassigned ODCs

In the previous section it is shown that ODC variables do not affect the outcome of a SAT problem. This fact can be used to increase the efficiency of a SAT solver and to reduce the size of its solution cubes through some minor modifications. For instance, the decision-making and BCP procedures simply ignore all ODCs variables altogether thus leaving them unassigned.

Before SAT solvers can benefit from ODC variables, these variables must be efficiently identified. Since variables become ODCs dynamically as the SAT solver makes assignments, identifying the ODCs must require little overhead. Indeed, much of the required analysis can be performed as a quick pre-processing step before the SAT solving starts.

The pre-processing step comprises of associating a *controlling value* and a *list of variables* to every variable in the SAT problem. When an assignment is made to a variable v_c, if the assigned value is the same as its controlling value, then all the variables in the list are identified as ODCs. The SAT solver can then ignore these variables for all procedures until backtracking occurs and the initial variable assignment is reversed.

The controlling value is set to 0 (1) for a variable that is an input of an `(N)AND` `((N)OR)` gate. Recall that for simplicity the fanout branch and stem are treated as different variables. For a given variable v_c, the the list is constructed by

1. adding all the neighbors of v_c

2. for each variable in the list v_l, adding each variable in the transitive fanin of v_l only if that variable has a single fanout.

For example, for variable c in Figure 4 the controlling value is 0 and its list contains variables $\{d, e\}$. Similarly, for variable g the controlling value is 1 and its list contains variables $\{f, a, b\}$.

Along with the list of variables, each stem variable also contains a field stating the number of the fanouts (branches) that are ODCs. When all the fanouts are ODCs, the variable itself and the variables in its list become ODCs as well.

Within the SAT solver, when a variable is identified as an ODC, a flag is set to allow the decision-making and BCP procedures to simply skip them and leave them unassigned. The backtrack procedure is responsible for resetting the ODC flags when the decision associated to the initial assignment is reversed.

5. Finding Small Solution Cubes

The previous section showed that ODC variables can be identified and ignored in some SAT procedures. In this section the importance of leaving ODC variables unassigned is emphasized. For SAT-based reachability analysis it is critical to find small solutions cubes in an efficient manner. A solution cube containing fewer literals covers more solutions and reduces the number of iterations to find all solutions. In this regard, SAT solvers that exploit ODCs have the inherent advantage of finding small solution cubes without applying assignment reduction procedures [10, 28].
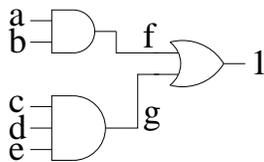
Figure 4. Example showing benefits of ODCs

As an example consider the circuit in Figure 4 where the objective is to find all primary input assignments such that the output is 1. An all-solution SAT solver using observability don't cares can make the assignment $f = 1$ leading to signals c, d, e, g becoming ODCs and resulting in the solution cube $\{a, b\} = \{1, 1\}$. Similarly, in the next iteration it can make the assignment $g = 1$ leading to signals a, b, f becoming ODCs and resulting in the solution cube $\{c, d, e\} = \{1, 1, 1\}$. In contrast, an all-solution SAT solver without any assignment reduction procedures needs up to 11 iterations to find all the solution minterms. These 11 minterms are made up of 8 minterms covered by the cube $\{a, b\} = \{1, 1\}$ plus four minterms covered by the cube $\{c, d, e\} = \{1, 1, 1\}$ minus the common minterm

$\{a, b, c, d, e\} = \{1, 1, 1, 1, 1\}$. Next, a novel decision-making procedure is developed to increase the likelihood of finding small solution cubes as required by reachability analysis engines.

5.1 Generating more ODCs with a Novel Decision-Making Procedure

The decision-making procedure of a SAT solver can help generate more ODCs and result in smaller solution cubes. This can be accomplished by making logic assignment decisions on variables that lead to more ODC variables. For example, consider the circuit partitioned in fanout free cones as illustrated in Figure 5. A fanout free cone is a subset of a circuit that does not contain multiple fanouts (i.e., stems forking into two or more branches) [12]. Assuming that primary input variables are the variables of interest, to generate small solutions cubes, decisions should be made such that A and B are ODCs first. This will result in the majority of primary inputs being unassigned.
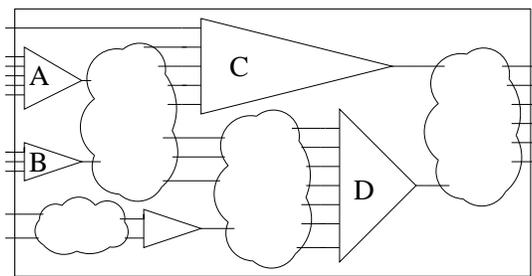

Figure 5. Partition of circuit in fanout free cones

For applications such as model checking and trace reduction, the variables of interest are often the current state and primary input variables. This means that solutions to a SAT problem should be in terms of assignments to these variables. Note that the proposed technique is not restricted to only these variables of interest.

The proposed decision-making procedure branches on variables with both a high score as well as a high VSIDS rank [23]. The score for each variable is calculated in a quick linear time pre-processing step with respect to the problem size. A variable's score represents its ability to make other variables of interest ODCs. The objective of each decision is to branch on the variables with the highest probability of generating the most ODCs on the variables of interest. As such, when a satisfying assignment is found, many variables of interest may be left unassigned leading to a smaller solution cube.

The variable scoring scheme is called *Influence on Variables of Interest* or `Ivi` for short[4]. The `Ivi` value for every variable is found in a pre-processing step using two breadth-first traversals of the circuit. In the first pass, an intermediate value called *Variables of Interest Predecessor* or `Vip` is calculated for each variable. For each circuit line, the `Vip` value increases with the number of variables of interest that exist in its transitive fanin and scales down with the number of its fanouts or branches.

---

4. `Ivi` and `Vip` are generalizations of the `Lipi` and `Pip` scores in [27] as they are concerned with any variable of interest type instead of only primary input types.

```
 1: //Calculate Vip value
 2: for all (gates g in a breadth-first manner) do
 3:    if g is variable of interest then
 4:       if number of fanouts for g > 20 then
 5:          g.Vip=1
 6:       else
 7:          g.Vip=100 − 5∗(number of fanouts-1)
 8:       end if
 9:    else
10:       for all (fanins pred) do
11:          g.Vip=g.Vip + pred.Vip
12:       end for
13:       if number of fanouts for g > 3 then
14:          g.Vip = g.Vip/4
15:       else
16:          g.Pip = g.Pip/(number of fanouts)
17:       end if
18:    end if
19:    //Initialize Ivi
20:    g.Ivi_0 = 0
21:    g.Ivi_1 = 0
22: end for
23: //Calculate Ivi value
24: for all (gates g in a breadth-first manner) do
25:    for all (fanins line of g) do
26:       if (g controlling value = 0) then
27:          for all (fanins neighbor of g other than line) do
28:             line.Ivi_0 = line.Ivi_0+neighbor.Vip
29:          end for
30:       else
31:          for all (fanins neighbor of g other than line ) do
32:             line.Ivi_1 = line.Ivi_1+neighbor.Vip
33:          end for
34:       end if
35:       line.Ivi = max(line.Ivi_0, line.Ivi_1)
36:    end for
37: end for
```

Figure 6. Calculating Vip and Ivi

The algorithm in Figure 6 illustrates how the `Vip` and `Ivi` variables are calculated with exact values attributed to the variables of interest and the scaling factors. Lines 1-22 calculate the `Vip` values. These values are calculated differently for variables of interest such as primary inputs and for other variables. For variables of interest a score is given that is inversely proportional to the number of fanouts. The reason for this relation is that the more fanouts there are the less likely it is for the variable to become and ODC. When the number of fanouts is greater than 20, `Vip` is assigned the lowest value of 1. For all other variables the `Vip` is found to be the sum of the `Vip` of its fanins. These values also decrease with the number of fanouts as shown by lines 13-16.

Lines 23-37 of Figure 6 demonstrate how `Ivi` score for a variable is calculated based on the `Vip` value of its neighbors (the other inputs to the gate). There are two `Ivi` scores for each circuit line, one for each assignment phase (0 or 1). Each `Ivi` score for a line is calculated based on the number of other neighboring lines that become ODCs when this line takes on a controlling value. The largest of the `Ivi` scores from both phases is selected and used in the branching procedure.

Figure 7 shows the `Vip` and `Ivi` values on a sample circuit. For example, to calculate the `Vip` value for signal e, we add the `Vip` value of its fanins, that is 100 for input c + 100 for input d. The `Ivi` score for e is calculated by adding the `Vip` value of the neighboring lines for each phase (i.e., 100 for neighbor a + 100 for neighbor b).

Consider the circuit problem of Figure 7 in the context of a SAT problem. Since the `Ivi` score for variable a (along with b) is the largest among all variable scores, the first decision made by the SAT solver is $a = 0$ ($b = 0$). As a result of this decision the primary inputs b, c and d become ODCs. If the problem is satisfied without backtracking on the decision $a = 0$, the only assigned primary input variable is a which leads to the solution cube $\{a = 0\}$. This example illustrates that the proposed scoring scheme increases the likelihood of generating small solution cubes.
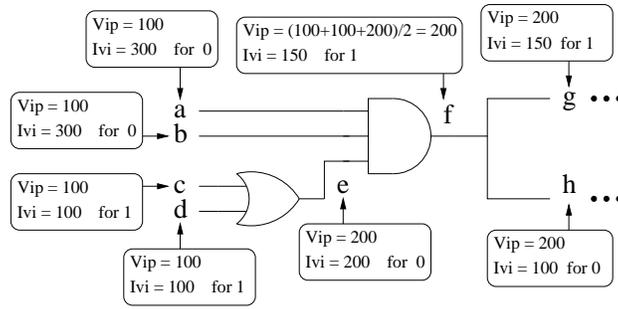

Figure 7. Example of Vip and Ivi assignments

The overall decision-making procedure is dynamic as it picks the variable with the highest `Ivi` score from the top ten variables with the highest VSIDS score as proposed in [28]. This scheme achieves a balance between solving the SAT problem quickly based on the effective decaying scheme of VSIDS and generating many ODCs on the variables of interest. The experiments demonstrate the effectiveness of this branching scheme compared to others.

6. Applications

This section presents applications that the proposed ODC techniques can be applied to. Reachability analysis is discussed first, followed by unbounded model checking and trace reduction.

6.1 Reachability Analysis

Reachability analysis is the process of determining whether a state q_k is reachable from another state q_0. Roughly speaking, reachability analysis can be performed by traversing the state space backwards from state q_k until a state q_0 is found or a fix-point is reached, where no new states are found [16]. Pre-image computation is a central procedure of reachability analysis as it performs the single backward steps.
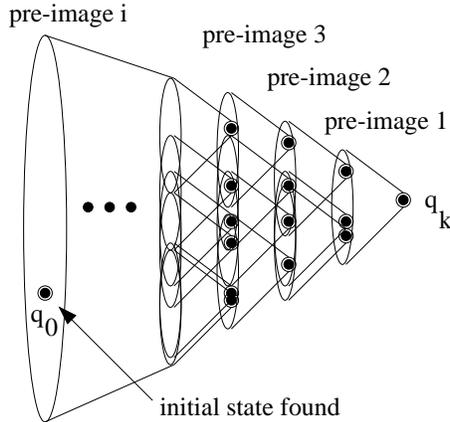
Figure 8. Illustration of reachability analysis

The SAT problem for pre-image computation is created by constraining the next state variables S' of a circuit's transition relation $T(S, S')$ to the state value q_i. Thus the SAT problem is to find the states assignments to the problem $\exists S'.T(S, S') \land (S' \leftrightarrow q_i)$.

A conventional SAT solver is not very efficient for pre-image computation since many solutions, if not all, are often required. An all-solution SAT solver such as the one described in the previous section is ideal for this type of application as the ODCs allow it to find small solution cubes efficiently.

The manner in which the state space is traversed depends on the how states are selected by the reachability analysis engine. 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 8 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.

6.2 Unbounded Model Checking

Unbounded Model Checking (UMC), is a verification technique used to determine whether specific properties are implemented correctly within a design. The properties can be specified in temporal languages such as CTL and LTL [5] to define both correct and incorrect circuit behaviors.

Unlike popular SAT-based Bounded Model Checking techniques that unfold the transition relation for a finite number of clock cycles, UMC performs model checking without this explicit bound [2]. Instead, a single copy of the transition relation is used to find a transition path that violates the given property or to prove that such a path does not exist.

Reachability analysis is a central procedure employed by UMC engines. For simple CTL properties such as EFq_k, where the goal is to determine whether a state q_k is reachable from the initial state q_0, reachability analysis can applied directly to UMC problems [5].

Since UMC does not use an explicit bound, it is critical that the reachability analysis tool be capable of finding small cubes and storing them efficiently. The proposed ODC-based SAT solver appears to be ideal for UMC problems due to its inherent small solution

cubes. The benefit of the proposed techniques on UMC problems are demonstrated in the experiments.

6.3 Trace Reduction

A trace is a single sequence of states that represents specific transitions of a circuit for a given number of clock cycles. A trace, also known as a counter-example, is the outcome of a verification tool such as a simulator or a model checker. Since the majority of verification performed in the industry is based simulation techniques with limited use of formal tools, the traces are often much longer than necessary. In the verification debugging process, these traces must be analyzed to locate error sources when verification tools determine the existence of errors. Since traces can often be thousands of clock cycles long, trace reduction approaches are employed to reduce their size and thus accelerate debugging.

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 is a pre-image of q_j and q_j is an image of q_i. The objective of trace reduction is to shorten the path from the initial state q_0 to the final state q_k.

One approach to trace reduction is to perform reachability analysis on some of the states belonging to the original trace [29]. 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 can then be 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.
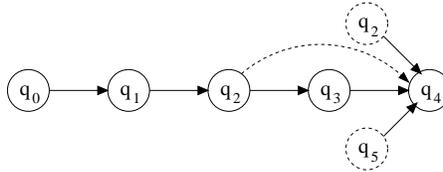
Figure 9. Updating the graph G with new nodes and edges

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 9 where the original trace is shown as the sequence $< q_0, q_1, q_2, q_3, q_4 >$ 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.

For more details about the trace reduction algorithm, the reader is referred to [29]. The benefit of the reachability analysis engine which uses an ODC-based SAT solver on trace reduction is demonstrated through the experiments.

7. Experiments

The proposed ODC-based all-solution SAT solver is developed on top of zChaff 2004.11.15 [23] as the basis of a reachability analysis framework. The all-solution SAT solver is also enhanced with the cube reduction procedure presented in [15]. In this section, the benefits of the ODCs are demonstrated through three set of experiments. First, the ODC-based all-solution SAT solver is compared against other approaches to determine its strengths in isolation. Next, the effectiveness of the reachability analysis engine is shown in the context of two applications, unbounded model checking and trace compaction. All experiments are run on a Sun Blade 1000 machine with a 750 MHz CPU and 2.5 GB of memory. A time out limit of 500 seconds and memory limit of 500 MB are used for all experiments, unless otherwise stated.

7.1 ODC-based SAT Solver

For the first set of experiments, over 1000 problems based on all of the ISCAS'89 benchmarks are used. In these problems, current state and next state variables are replaced with primary inputs and primary outputs, respectively. In order to emulate pre-image computation problems, where a number of outputs are well specified while others are unconstrained, we constrain an arbitrary number of primary outputs to 0 or 1 at random. The objective of these problems is to find all the primary input assignments that satisfy the constraints. In other words, the primary inputs are the variables of interest. Since all the solutions are sought, finding small solution cubes in an efficient manner is critical in these experiments. These problems are similar to scenarios found in many problems such as pre-image computation [20] and circuit optimization [21, 30].

Figure 10 (i) plots the run time of the all-solution SAT solver with and without the ODCs against each other. Points below the diagonal line signify that the ODC technique results in faster run times. Since the majority of points are below the diagonal line, it is evident that the ODC methods developed here are effective for all-solution SAT solvers. On the average, the performance gain is over $1.52\times$, while the scatter plot shows speedups of nearly one order of magnitude in some cases. The reason for this discrepancy is that average performance gain is heavily influenced by the large number of problems with speed-ups between 1-$2\times$.

Figure 10 (ii) and 11 (i) and (ii) illustrate the benefits of the proposed decision-making heuristic over three other methods. As discussed in Section 5.1, the Ivi scoring scheme is quite effective at making decisions that produce many ODC primary inputs, but to achieve a balance between solving each iteration quickly and finding small solution cubes, a strategy similar to [28] is employed. In this strategy the variable with the highest Ivi score is selected from the variables with the highest VSIDS scores. As shown in [28], this decision making approach is found to be well suited for all-solution SAT problems.

In Figure 10 (ii) the proposed scoring scheme is compared against a random scoring scheme to show the impact of the scoring scheme alone. The data points in Figure 10 (ii) clearly demonstrate the out performance of the combined Ivi and VSIDS scoring scheme over the random approach. However, there are certain cases where random fares better which indicates that the proposed heuristic may have room for improvement.
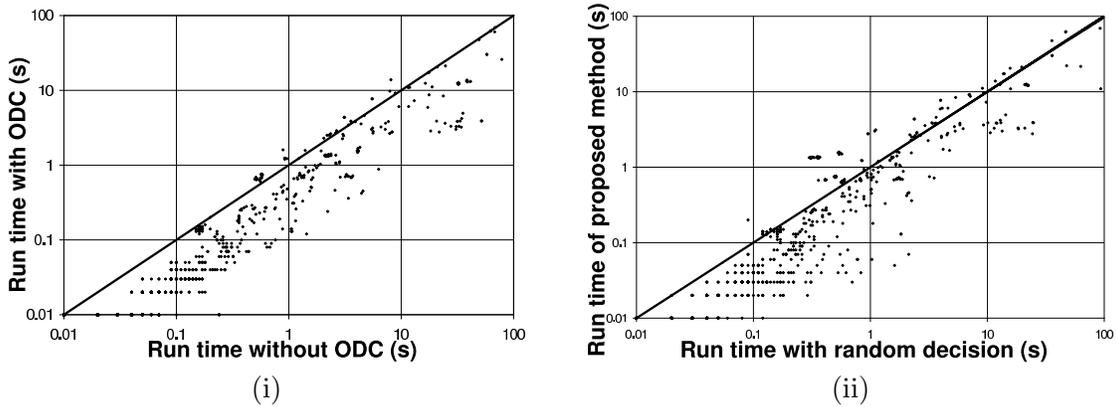
Figure 10. Performance comparison (i) with and without ODCs (ii) with random decisions

Figure 11 (i) compares the proposed approach (combined `Ivi` + VSIDS) against the *lazy* scoring scheme used in [28] which is successful at producing many ODC variables in general. Like the proposed scheme, the lazy method is also a hybrid which picks the variables with the highest lazy score from a window of variables with high VSIDS scores. In [28] it was shown that the lazy scheme outperforms the pure VSIDS scheme for circuit-based problems. Figure 11 (i) clearly demonstrates that the proposed `Ivi` + VSIDS scheme is superior to the lazy + VSIDS scheme as the majority of points are consistently under the diagonal line.

In Figure 11 (ii), the proposed scoring scheme (combined `Ivi` and VSIDS) is compared with the pure `Ivi` scoring scheme to illustrate the trade-off between solving for solutions quickly and finding small solution cubes. The out performance of the `Ivi` and VSIDS scheme shown in Figure 11 (ii) clearly emphasizes the importance of utilizing the dynamic nature of VSIDS. It is interesting to compare the general distribution of the points of Figure 11 (ii) with that of Figure 10 (ii). Although the schemes used are quite different, their general performance versus the proposed scoring scheme appears to be similar. Therefore, the pure `Ivi` and random scheme share some similar properties.

The scatter plots of Figures 10 and 11 clearly demonstrate that the proposed scoring scheme is superior to the other on the average. The `Ivi` score allows for signals to be selected to produce many ODC on the variables of interest, while the VSIDS scheme allows for a dynamic decision procedure able to adapt to the changing nature of the SAT problem.

7.2 Unbounded Model Checking

In this section, the SAT-based reachability analysis framework is evaluated on unbounded model checking problems. The UMC problems are of type EFp, where p is a reachable or unreachable state picked at random. The UMC algorithm uses a hybrid BFS and DFS approach where the next state for pre-image computation is chosen as the state with the smallest hamming distance to the initial state q_0 [29]. The smallest hamming distance is a greedy heuristic used to reach the initial state with the fewest number of iterations possible. The data structure used to find whether a newly found pre-image is already found by the reachability analysis engine is similar to the tree structure described in [29]. Instead of using an explicit timeout or memory limit, the number of state cubes visited by the
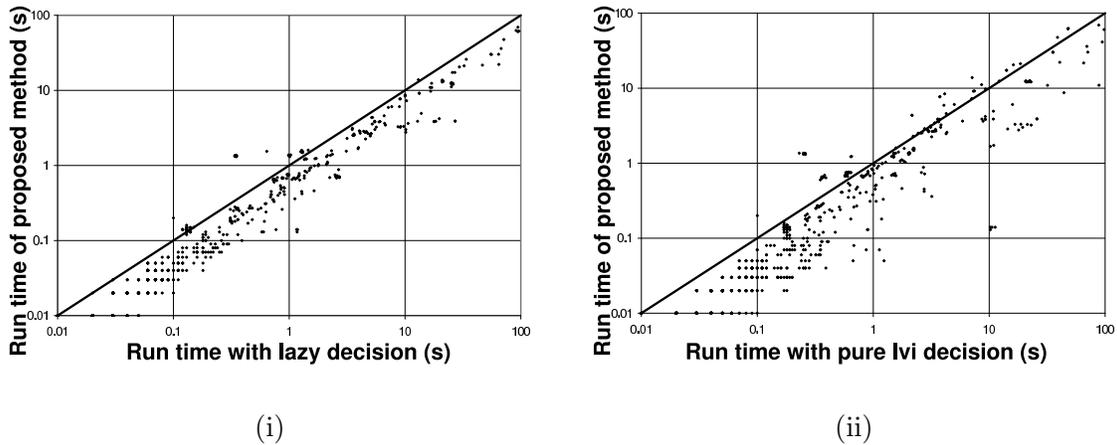
(i) (ii)

Figure 11. Comparison of different decision making heuristics

unbounded model checker is limited to 1000 since it is related to both memory requirement and performance.

Table 1 shows the results of the presented ODC-based SAT solver versus a justification engine similar to [15] on UMC problems. This justification approach is a state-of-the-art technique for creating small cubes from each SAT solver solution, which is shown to be competitive relative to other techniques such as [20]. Our implementation of [15] is built on top of zChaff and makes use of the circuit-based justification engine, as described in [1], each time the SAT solver finds a solution.

For each of the ISCAS'89 circuits shown in column one, the number of solution cubes found by the proposed approach and the justification approach is shown in columns two and three, respectively. Columns four and five give an insight into how the UMC problem is solved by showing the maximum depth visited by the reachability analysis engine for each respective approach. The run times required to solve the problems for the proposed and the justification approaches are shown in columns six and seven, respectively, with the faster approach shown in bold face. The final column present the average speed-up of the proposed approach with respect to the justification approach.

The results of Table 1 show a fairly similar behavior in terms of the number of solutions found and the depth visited by the UMC engine. For reachable problems that are solved within the alloted 1000 solutions, these numbers do not provide much insight in the effectiveness of the ODC techniques. This is due to the fact that a reachable state can be found quickly when a "lucky" state is selected for pre-image computation which leads to the initial state. The run time of the UMC tool (in columns five and six) allow us to evaluate the effectiveness of the different techniques.

The UMC results can be split into two sets, the easy reachable problems that are solved with few solution cubes and the harder problems that are not solved within finding 1000 solution cubes. The techniques are comparable on the easy problems, with the justification approach performing slightly better. However, for the more interesting harder problems, the proposed approach is overwhelmingly faster. Indeed, for the 13 cases where the solution limit is reached by both approaches ("1000" shown in columns two and three in Table 1), the

Table 1. Comparison of proposed and justification techniques for UMC problems

circuits	proposed # solutions	justification # solutions	proposed depth	justification depth	proposed time (s)	justification time (s)	speed-up (×)
s1196	2	2	1	1	0.01	0.01	1.00
s1238	1	1	1	1	0.01	0.01	1.00
s13207.1	170	2	88	1	0.10	**0.01**	0.10
s13207	37	1	1	1	0.03	**0.01**	0.33
s1423	9	1	1	11	0.01	0.01	1.00
s1488	1000	1000	21	21	**112.03**	398.90	3.56
s1494	1000	1000	21	21	**120.51**	397.49	3.30
s15850.1	1000	1000	434	373	**0.71**	3.13	4.41
s15850	1	1	1	1	**0.06**	0.07	1.17
s208.1	1	1	1	1	0.09	**0.08**	0.89
s27	8	1	1	1	0.01	0.01	1.00
s298	1000	1000	511	701	**122.17**	500.00	4.09
s344	1	1	1	1	0.01	0.01	1.00
s349	1000	1000	364	387	**0.72**	3.16	4.39
s35932	2	2	1	1	0.01	0.01	1.00
s35932	6	1	1	1	0.01	0.01	1.00
s382	28	1	1	1	0.02	**0.01**	0.50
s38417	1000	1000	463	504	**0.84**	2.70	3.21
s38584.1	1000	1000	462	512	**0.81**	2.57	3.17
s38584	1	1	1	1	0.01	0.01	1.00
s386	1000	65	24	18	1.31	**0.09**	0.07
s400	6	1	38	25	0.01	0.01	1.00
s420.1	1000	1	1	91	1.41	**0.18**	0.13
s444	11	1	1	1	0.01	0.01	1.00
s510	11	1	1	1	0.01	0.01	1.00
s526	1000	1000	543	617	**1.63**	3.41	2.09
s526n	1000	1000	511	511	**14.80**	114.19	7.72
s5378	1000	1000	511	511	**13.52**	114.14	8.44
s641	1000	44	511	26	1.38	**0.08**	0.58
s713	1	1	1	1	**0.01**	0.02	2.00
s820	1	1	1	1	0.02	0.02	1.00
s832	1000	1000	411	238	**2.43**	8.42	3.47
s838.1	102	1	49	1	0.18	**0.01**	0.06
s9234.1	101	1	48	1	0.18	**0.01**	0.06
s9234	1000	1000	511	511	**41.92**	312.42	7.45
s953	1000	1000	511	511	**33.95**	256.37	7.55
average	896.70	715.41	326.97	297.68	25.46	114.46	2.24

proposed technique is always faster. On average the ODC-based technique requires a run time of 25.46 seconds, while the justification-based technique requires a run time of 114.46 seconds for all experiments. These results demonstrate that the ODC approach is effective for BMC, as it is able to find more solutions quickly with an average of 4× performance improvement.

7.3 Trace Reduction

In this section the effectiveness and efficiency of the proposed reachability analysis framework is demonstrated on the trace reduction application. The traces are generated using directed random simulation where control inputs such as `enable` and `reset` are set appropriately, while other signals are assigned randomly for the entire trace sequence. The trace reduction approach is limited to find at most 10,000 state *cubes* via pre-image computation and an explicit timeout is not used. Since the compaction techniques of previous

works [4, 24, 32] are not publicly available, and due to the fact that the assertions and errors used are also unknown, the results cannot be compared.

Table 2 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 *orig* describes the original length of each trace (50, 100, or 1000). The next two columns under the *trace length* label of each section show the resulting trace length after applying the pre-image computation technique (*pre-img*) followed by the reachability analysis technique (*reach*). More specifically, the pre-image computation technique is the single step pre-image process described in Section 6.3 and the reachability analysis technique is the proposed method introduced in Section 6.1. The fourth and fifth columns of each section labeled with *run time* show the processing time in seconds required for *pre-img* and *reach*, respectively. Note that both the pre-image computation and reachability analysis are based on the proposed ODC-based SAT solver.

Table 2 shows how both techniques effectively work together to reduce the trace length. For many circuits, the original trace length is first reduced greatly by the single step pre-image (*pre-img*) technique and further reduced by the reachability analysis (*reach*). For example, the trace for circuit s344 is first reduced from 50 clock cycles to 33 clock cycles using *pre-img*, and then again from 33 clock cycles to 1 clock cycle using *reach*.

Analyzing the results in Table 2, notice that many traces are reduced to having a single clock cycle (length of 1), or a very short trace size after applying reachability analysis. This result can be partially attributed to the state selection heuristics of Section 5.1. These techniques result in smaller cubes, which increase the number of short paths created through the graph G and the likelihood that they will lead to the initial state.

Table 3 provides more insight into the results in Table 2 by providing the average length reductions achieved by the different components of the proposed approach for traces of size 50, 100, and 1000. Similar to Table 2, 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-img*), 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-img* 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 overall benefit of these reduced traces is that the subsequent debugging task is much easier. For manual debugging, the verification engineer can concentrate on these reduced traces in order to find the error source. For automated debugging, the impact is much greater since sequential debuggers are limited by the length of the traces [7]. Thus reducing the trace lengths can provide benefits to both manual and automated debugging.

Table 2. Results comparing reachability analysis approaches for trace reduction problems

circuits	trace length			run time (s)		trace length			run time (s)		trace length			run time (s)	
	orig	pre-img	reach	pre-img	reach	orig	pre-img	reach	pre-img	reach	orig	pre-img	reach	pre-img	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	0.00	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 3. Effectiveness of each component of the proposed trace reduction technique

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

JSAT

8. Conclusion

This paper demonstrated the benefits of ODCs for reachability analysis problems. We developed an efficient way to identify ODCs during the SAT solving process and proposed to leave such variables unassigned. As a result, SAT solutions can be found with fewer assigned variables, thus leading to smaller solution cubes. The advantages of small solution cubes make the proposed approach ideal for reachability analysis applications, such as unbounded model checking and trace reduction. Experiments on these applications demonstrated results of up to an order of magnitude better in terms of performance when using the proposed ODC techniques.

References

[1] M. Abramovici, M.A. Breuer, and A.D. Friedman. *Digital Systems Testing and Testable Design.* Computer Science Press, 1990.

[2] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. In *Advances In Computers*, 2003.

[3] A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Design Automation Conf.*, pages 317–320, 1999.

[4] Y. Chen and F. Chen. Algorithms for compacting error traces. In *ASP Design Automation Conf.*, pages 99–103, 2003.

[5] E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.

[6] G. De Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, Inc., 1994.

[7] M. Fahim Ali, A. Veneris, S. Safarpour, R. Drechsler, A. Smith, and M.S.Abadir. Debugging sequential circuits using Boolean satisfiability. In *Int'l Conf. on CAD*, pages 204–209, 2004.

[8] Z. Fu, Y. Yu, and S. Malik. Considering circuit observability don't cares in CNF satisfiability. In *Design, Automation and Test in Europe*, pages 1108–1113, 2005.

[9] M. Ganai, A. Gupta, and P. Ashar. Efficient SAT-based symbolic unbounded model checking using circuit cofactoring. In *Int'l Conf. on CAD*, pages 510–517, November 2004.

[10] A. Gupta, A. Gupta, Z. Yang, and P. Ashar. Dynamic detection and removal of inactive clauses in SAT with application in image computation. In *Design Automation Conf.*, pages 536–541, 2001.

[11] M. Iyer, G. Parthasarathy, and K.T. Cheng. SATORI - a fast sequential SAT engine for circuits. In *Int'l Conf. on CAD*, pages 320–325, 2003.

[12] N.K. Jha and S. Gupta. *Testing of Digital Systems.* Cambridge University Press, 2003.

[13] H. Jin, H. Han, and F. Somenzi. Efficient conflict analysis for finding all satisfying assignments of a boolean circuit. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 287–300, 2005.

[14] H. Jin and F. Somenzi. Prime clauses for fast enumeration of satisfying assignments to boolean circuits. In *Design Automation Conf.*, pages 750–753, 2005.

[15] H-J. Kang and I-C. Park. SAT-based unbounded symbolic model checking. *IEEE Trans. on CAD*, **24**(2):129–140, 2005.

[16] Th. Kropf. *Introduction to Formal Hardware Verification*. Springer, 1999.

[17] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Trans. on CAD*, **11**:4–15, 1992.

[18] B. Li, M.S. Hsiao, and S. Sheng. A novel SAT all-solutions solver for efficient preimage computation. In *Design, Automation and Test in Europe*, pages 272–277, 2004.

[19] J.P. Marques-Silva and K.A. Sakallah. GRASP – a new search algorithm for satisfiability. In *Int'l Conf. on CAD*, pages 220–227, 1996.

[20] K.L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Computer Aided Verification*, pages 250–264, 2002.

[21] A. Mishchenko and R. Brayton. SAT-based complete don't-care computation for network optimization. In *Design, Automation and Test in Europe*, pages 418–413, 2005.

[22] A. Mordago and J.P. Marques-Silva. Good learning and implicit model enumeration. In *IEEE Int'l Conf. on Tools with Artificial Intelligence*, 2005.

[23] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conf.*, pages 530–535, 2001.

[24] 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.*, pages 25–29, 2006.

[25] D. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *J. Symb. Comput.*, **2**(3):293–304, 1986.

[26] P. Rashinkar, P. Paterson, and L. Singh. *System-on-a-chip Verification: Methodology and Techniques*. Kluwer Academic Publisher, 2000.

[27] 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*, pages 1587–1590, 2006.

[28] S. Safarpour, A. Veneris, R. Drechsler, and J. Hang. Managing don't cares in Boolean satisfiability. In *Design, Automation and Test in Europe*, pages 260–265, 2004.

[29] S. Safarpour, A. Veneris, and H. Mangassarian. Trace compaction using SAT-based reachability analysis. In *ASP Design Automation Conf.*, pages 932–937, 2007.

[30] S. Sapra, M. Theobald, and E. Clarke. SAT-based algorithms for logic minimization. In *Int'l Conf. on Comp. Design*, pages 510–518, 2003.

[31] S. Shen, Y. Qin, and S. Li. A fast counterexample minimization approach with refutation analysis and incremental SAT. In *ASP Design Automation Conf.*, pages 451–454, 2005.

[32] S. Shen, Y. Qin, and S. Li. A faster counterexample minimization algorithm based on refutation analysis. In *Design, Automation and Test in Europe*, pages 672–677, 2005.

[33] G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part II*, pages 115–125. 1968.

[34] M. Velev. Encoding global unobservability for efficient translation to SAT. In *Int'l Conf. on Theory and Applications of Satisfiability Testing*, pages 197–204, 2004.

[35] M. Velev. Exploiting signal unobservability for efficient translation to CNF in formal verification of microprocessors. In *Design, Automation and Test in Europe*, pages 266–271, 2004.

[36] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli. Sat sweeping with local observability don't-cares. In *Design Automation Conf.*, pages 229–234, 2006.