Implicit Search-Space Aware Cofactor Expansion: A Novel Preimage Computation Technique

Kameshwar Chandrasekar
Intel Corporation
Santa Clara, CA 95054
Email: kameshwar.chandrasekar@intel.com

Michael S. Hsiao
Virginia Tech
Blacksburg, VA 24061
Email: mhsiao@vt.edu

***Abstract*— In this paper, we introduce a novel preimage computation technique that directly computes the circuit cofactors without an explicit search for any satisfiable solution. We use an implicit search on the primary inputs of a sequential circuit to compute all the circuit cofactors for the target preimage. In order to alleviate the computational cost, aggressive learning techniques are introduced that reason on the search-states by analyzing the relations among circuit cofactors. Such analysis generates search-state induced clauses that directly help to prune the cofactor space during preimage computation and to perform non-chronological backtracking. Experimental results show that a significant improvement can be achieved in both performance and capacity as compared to the existing techniques.**

I. Introduction

Symbolic methods have been widely used for Model Checking finite transition systems. The representation and manipulation of Boolean expressions are critical for the scalability and robustness of Symbolic Model Checking. Traditionally, Reduced Ordered Binary Decision Diagrams (ROBDDs) have been used for this purpose, since they are canonical and can be efficiently manipulated. However, they suffer from potential memory explosion for large designs. On the other hand, Boolean Satisfiability (*SAT*) has received significant attention in recent years, and it offers a promising alternative to BDD-based methods. One of the important research problems in Symbolic Model Checking is to see if we can reach a given target state (could be a buggy state), from an initial state, in a sequential design. In general, *SAT*-based Model Checking can be classified into Bounded Model Checking (*BMC*) and Unbounded Model Checking (*UMC*).

In BMC [1], we look for traces/paths of a bounded length, say k, that violate the property. On the other hand, in UMC, we explore the entire reachable state space to prove the correctness of a property. The core computation step in *UMC* is image/preimage computation that performs state space traversal. In essence, image (preimage) computation requires that the set of all next (previous) states that can be reached from a given set of states, in one cycle, be computed. In *SAT*-based *UMC* [2] [3] [4], the preimage needs to be computed iteratively until either the desired state or a fixed point is reached. In order to handle large circuits, abstraction offers a feasible solution using the abstract-verify-refine loop [5] [6] [7]. We propose a novel circuit based preimage computation technique with the following features:

1) We propose a decision tree search technique that directly computes the circuit cofactors for preimage by quantifying the input variables.
2) We derive relations among the circuit cofactors based on the search states reached during the search.
3) We introduce search-state induced learning, based on the relations derived, to prune the cofactor space and to perform non-chronological backtracking.

We implemented the proposed technique and compared against three preimage computation techniques: (1) McMillan's blocking clause approach [2], (2) Sheng and Hsiao's ATPG (Automatic Test Pattern Generation) based approach [8], and (3) Ganai et al.'s cofactor blocking approach [4]. We implemented all the four techniques on top of a publicly available verification tool called ABC [9]. Our experimental results on publicly available ISCAS '85 and VIS-ITC '99 benchmark circuits, show that direct circuit cofactoring with search-state induced learning is competitive with the existing techniques and can lead to several orders of magnitude improvement in both run-time and memory.

A. Previous Work

Initially, certain variants to Binary Decision Diagrams were proposed for image/preimage computation. In [10] [11] [12], non-canonical structures such as Reduced Boolean Circuit (RBC), Boolean Expression Diagrams (BED) and AND-inverter graphs are used to represent the transition relation, and quantification rules are proposed for specific sub-structures. However, in the general case, the length of formulas may grow exponentially due to the quantification. In [13], *SAT* solvers are used to provide a disjunctive decomposition of clauses and BDDs are used subsequently to solve the disjunctions. In [2], a seminal preimage computation technique is proposed for unbounded model checking using a pure *SAT* solver. The transition relation and negation of the property are represented as a set of clauses, and variable quantification is performed during solution enumeration. After each solution is found, an enlarged cube is identified by re-building the implication graph. The negation of this cube is added as a *blocking clause* to the clause database. The set of all blocking clauses represent the preimage and they are stored in a Zero-suppressed Binary Decision Diagram (ZBDD) to perform light weight optimization. In [3], the authors use a simple circuit-based

1-4244-9707-X/06/$20.00 ©2006 IEEE

justification procedure to identify enlarged solution cubes (and thus avoid re-building the implication graph). In [14], the authors improved the ZBDD-to-clause conversion step for preimage iteration, by generating clauses for each node in the ZBDD rather than direct clause representation for the state-set. Recently, in [15] Jin et al. proposed to perform conflict analysis on the blocking clause and prune both the solution and conflict space efficiently. They further improvised the blocking clause approach by targeting only the prime clauses in [16] and showed that we can apply powerful pruning techniques, if we allow overlap of solution spaces across different SAT enumerations.

In [8], an ATPG engine is used for circuit based preimage computation. The transition relation is represented as a Boolean circuit and an ATPG engine is invoked to enumerate all the solutions that represent the complete preimage. Success-driven learning is proposed to avoid re-searching overlapping solution sub-spaces, thereby accelerating the search. The preimage is stored as a free BDD which shares all the common subgraphs and is generally more compact than ROBDD. Success-driven learning was further augmented in [17] to prune larger search-spaces. Success-driven learning was also integrated into a hybrid *SAT*-solver to take advantage of both circuit-based success-driven learning and clause-based conflict-driven learning in [18].

In [4], the transition relation is represented by an OR-inverter graph and efficient solution enumeration is performed using a specialized hybrid *SAT* solver. After obtaining each solution, the authors use the circuit cofactors, with respect to the input assignment, as the enlarged state space. Overall, their experiments show a run-time improvement over the technique in [2], since they are able to capture a larger state space at each enumeration.

II. Preliminaries

Let us consider a completely specified, deterministic Finite State Machine (FSM) of Mealy type that is represented by the 6-tuple: $< I, O, S, \delta, \lambda, S_0 >$

- I/O represents the primary inputs/outputs
- S/S_0 represents the state set / initial state set
- δ/λ represents the next state/output function

For a synchronous sequential circuit, the transition relation $T(X,I,X')$ is defined by:

$$T(X,I,X') = \bigwedge_{i=1}^{i=n} (x_i' \equiv \delta_i(X,I))$$

- X/X' is the set of present/next state variables
- δ_i is the next state function for x_i'

Symbolically, the preimage for a given set of target states, $S(X')$, is given by: $Preimage(X) = \exists_{I,X'} T(X,I,X') \wedge S(X')$

Please note that $T(X,I,X') \wedge S(X')$ can be represented as a single output Boolean circuit as shown in Figure 1, which is our model for preimage computation. We refer the model as a circuit and the circuit gates as Boolean variables when the meaning is clear from the context. Further, the Boolean value assignment to a variable or a gate is referred as a literal.

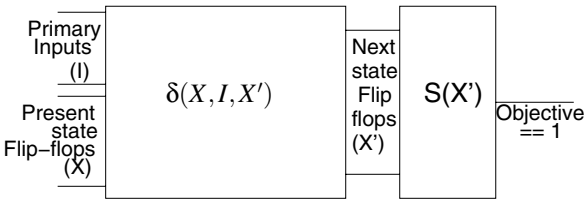

Fig. 1. Preimage computation model

A. Motivation

SAT-based approaches existentially quantify *all* the variables in the given CNF to find a solution. In preimage computation, we need to quantify the *primary input variables alone*. It is sufficient to branch on primary input variables alone and compute the generalized cofactors obtained with respect to each input assignment to the circuit. If we branch on the primary input variables only in a decision tree like fashion, then each terminal of the decision tree is a generalized cofactor w.r.t an input assignment (instead of a solution or a conflict). We can thus reduce the number of decision variables and directly compute the generalized cofactors. However, the practical limitations of this approach are the (1) lack of cofactor-driven learning techniques and (2) potentially huge cofactor sizes that might be computed at each terminal.

In order to overcome the aforementioned limitations, we propose search-state driven learning techniques that can deduce clauses from different search-states of the circuit. In order to block each cofactor, we deduce a **single clause** that can account for that cofactor space. Based on these clauses, we show that it is possible to perform non-chronological backtracking in the decision tree. Further, in order to contain the size of the preimage, we use the well-known AND-inverter graphs for the circuit and incorporate elegant DAG-aware rewriting rules [19] while constructing the preimage.

III. Basic Cofactor Expansion

Consider a circuit with m inputs, say $I = \{i_1, i_2, ...i_m\}$, n latches, say $X = \{x_1, x_2, ..., x_n\}$ and p outputs, say $O = \{o_1, o_2, ..., o_p\}$. Suppose we wish to compute the preimage for a target state-set $S(X')$. We first construct the single output circuit $C(I,X)$, as in Figure 1. The preimage on C is:

$$\begin{aligned}
Preim(X) &= \exists_I C(I,X) \\
&= \exists_{I \setminus i_1} [C_{(i_1=0)} \vee C_{(i_1=1)}] \\
&= \exists_{I \setminus \{i_1,i_2\}} [C_{(i_1=0,i_2=0)} \vee C_{(i_1=0,i_2=1)} \\
&\qquad \vee C_{(i_1=1,i_2=0)} \vee C_{(i_1=1,i_2=1)}] \\
&\quad \ldots \\
&= \exists_{i_m} [C_{(i_1=0,i_2=0,\ldots,i_{m-1}=0)} \vee \ldots \\
&\qquad \vee C_{(i_1=1,i_2=1,\ldots,i_{m-1}=1)}] \\
&= C_{(i_1=0,i_2=0,\ldots,i_{m-1}=0,i_m=0)} \vee \ldots \\
&\qquad \vee C_{(i_1=1,i_2=1,\ldots,i_{m-1}=1,i_m=1)}
\end{aligned}$$

It may be noted that all the generalized cofactors that are derived in the above equations are circuit cofactors. Specifically,

each term in the *last equation* of $Preim(X)$ is a generalized cofactor of $C(I,X)$ w.r.t. an input assignment. We refer to each term or a disjunction of such terms as the preimage cofactor. A brute-force approach to compute $Preim(X)$ is to simulate the circuit with all the 2^m input patterns one by one. Each simulation leads to a preimage cofactor and the disjunction of all these circuit cofactors is the preimage.

A systematic way of exploring all the 2^m input enumerations is to use a decision tree search. We branch only on the primary input variables in I and a cofactor can be computed at the terminal node of the decision tree (also called as terminal cofactor). Each terminal of the decision tree will be a preimage cofactor that represents exactly one term in above equation for $Preim(X)$. In the decision tree search, the disjunction of all the terminal cofactors is the preimage.

In order to reduce the number of cofactor enumerations, we choose to branch on a primary input variable (as a decision) only if it has an X-path (a chain of unspecified/unassigned gates) to the objective of the model in Figure 1. Therefore, we no longer have to compute all the 2^m preimage cofactors, as we may skip many of the redundant cofactors. Each terminal cofactor represents *at least* one term (or a disjunction of terms) in the aforementioned equation for $Preim(X)$. It may be noted that the disjunction of all terminal cofactors is the complete preimage.

A. Circuit Search-State Relations

In order to prune the cofactor space during preimage computation, it is possible to learn from the search-state relations that exist at different scenarios during preimage computation. We first review a few terms that are demonstrated in Figure 2, to explain the following discussion.

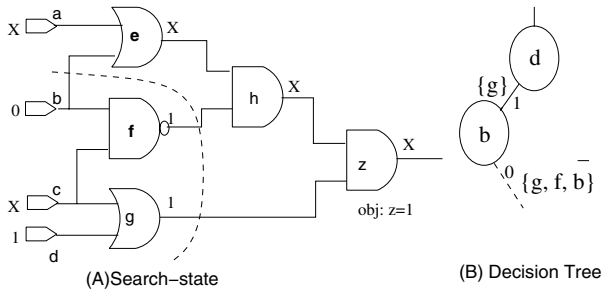

Fig. 2. Cut-sets in the Search Space.

For the circuit, $C(a,b,c,d)$, shown in Figure 2(A), a partial decision tree is shown in Figure 2(B).

- **Search-State** SS**:** the internal state of the circuit at each branch in the decision tree. It simply captures the set of all implied gate assignments at each branch in the decision tree. Ex: Figure 2(A) is the search-state for branch $b = 0$ in the decision tree (in Figure 2(B)).
- **Cut-set for search-state** $CS(SS)$**:** the set of specified gates in a search-state that has at least one X-path (a chain of unspecified/unassigned gates) to the circuit output. Ex: the dashed line in Figure 2(A) is the cut-set for the current search state.

In order to facilitate the discussion on search-states, we use the following abbreviations that correspond to a particular search-state, SS (examples from Figure 2(A)):

- **Gate Value Assignment** $GVA(SS)$**:** set of gate value pairs for all gates in C. Ex: $\{(a = X),(b = 0),(c = X),(d = 1),(e = X),...,(z = X)\}$.
- **Cutset Assignment** $CSA(SS)$**:** set of gate value pairs for all gates in the cut-set for a search-state. Ex: $\{(b = 0),(f = 1),(g = 1)\}$.
- **Cutset Conjunction** $CSC(SS)$**:** Boolean formula that represents the conjunction of literals that are obtained from all the gates in $CSA(SS)$. Ex: $\neg b \wedge f \wedge g$.
- **Input Value Assignment** $IVA(SS)$**:** set of gate value pairs for all the input gates in $GVA(SS)$. Ex: $\{(a = X),(b = 0),(c = X),(d = 1)\}$. We also say that SS is the search-state *induced* by the input value assignment $\{(a = X),(b = 0),(c = X),(d = 1)\}$.
- **Specified Input Value Assignment** $SIVA(SS)$**:** set of all specified gate value pairs in $IVA(SS)$. Ex:$\{(b = 0),(d = 1)\}$.
- **Input Value Conjunction** $IVC(SS)$**:** Boolean formula that represents the conjunction of literals that are obtained from all the gates in $SIVA(SS)$. Ex: $\neg b \wedge d$.
- **Logic decomposition / Circuit Cofactor for search-state** $Cof(SS)$**:** reduced circuit obtained by projecting the input value conjunction $IVC(SS)$ on the circuit, C. Ex: $C(a,0,c,1) \equiv a$. This can also be referred as $C \downarrow_{IVC(SS)}$ or $C \downarrow_{CSC(SS)}$.
- **Preimage Circuit Cofactors** $PreimCof(SS)$**:** disjunction of all terminal circuit cofactors that can be obtained for the circuit $Cof(SS)$.
- **Preimage Cofactor Space** $PreimCofSpace(SS)$**:** Boolean space that refers to $PreimCof(SS)$.

In order to perform learning on the search-states, we define the following relations among search-states:

- **Conformable Search states:** A search-state A is said to be *conformable* with search-state B, if $CSA(A) \subseteq GVA(B)$.
- **Extendable Search states:** A search-state A is said to be *extendable* to search-state B, if we can induce a search state D, such that $SIVA(D) \supseteq SIVA(A)$ and $CSA(D) = CSA(B)$.

Lemma 1: *If search-states A and B are such that* $CSA(A) = CSA(B)$*, then* $PreimCofSpace(A)$ *is equivalent to* $PreimCofSpace(B)$.

Proof: We know that $IVC(A) \rightarrow CSC(A)$ and $IVC(B) \rightarrow CSC(B)$ due to logic simulation. Since a cut-set disconnects its fanin-cone from the remaining portion of the circuit, $C \downarrow_{IVC(A)} = C \downarrow_{CSC(A)}$ and $C \downarrow_{IVC(B)} = C \downarrow_{CSC(B)}$. Although, the assignment to the gates in the fanin-cone of $CSA(A)$ and $CSA(B)$ may be different, $CSA(A) = CSA(B)$ and all gates in the remaining portion of the circuit are unspecified in both A and B. Therefore, the logic decomposition: $C \downarrow_{CSC(A)} = C \downarrow_{CSC(B)}$. Since $C \downarrow_{IVC(A)} = C \downarrow_{CSC(A)} = C \downarrow_{CSC(B)} = C \downarrow_{IVC(B)}$,

terminal cofactors computed after A and B will be equivalent.$\diamond$

Lemma 2: *If search-state A is conformable with search-state B, then A is extendable to B.*

Proof: We will construct a search-state D such that $SIVA(D) \supseteq SIVA(A)$ and $CSA(D) = CSA(B)$.

First, let us construct an implication graph $IG(A,B)$ as follows: Start from the literals in $CSC(B)$ and find its antecedents in the search-state of B. Recursively, find the antecedents until, we reach the gates in $CSA(A)$ or $IVA(B)$, whichever is earlier. The root nodes of $IG(A,B)$ will be the gates in $CSA(A)$ or $IVA(B)$ (see Figure 3).

Let $SIVA(IG)$ be the set of specified input value assignments in the root nodes of the implication graph, which are not in $CSA(A)$ or its fanin cone (see Figure 3). Let $IVC(IG)$ be the conjunction of all the literals in $SIVA(IG)$. Since $IVC(IG) \wedge CSC(A)$ is the conjunction of all literals in the root nodes of IG(A,B), $IVC(IG) \wedge CSC(A) \rightarrow CSC(B)$.

Since none of the gates in $SIVA(IG)$ are in $CSA(A)$ or its fanin cone and all the gates in $SIVA(A)$ are in $CSA(A)$ or its fanin cone (see Figure 3), $SIVA(IG) \bigcap SIVA(A) = \Phi$. Therefore, $SIVA(IG)$ and $SIVA(A)$ are compatible without any conflicting assignment. Let us induce the search-state D, such that $SIVA(D) = SIVA(IG) \bigcup SIVA(A)$. Then $IVC(D) = IVC(IG) \wedge IVC(A) \rightarrow IVC(IG) \wedge CSC(A) \rightarrow CSC(B)$, i.e., $CSA(D) = CSA(B)$. $\diamond$

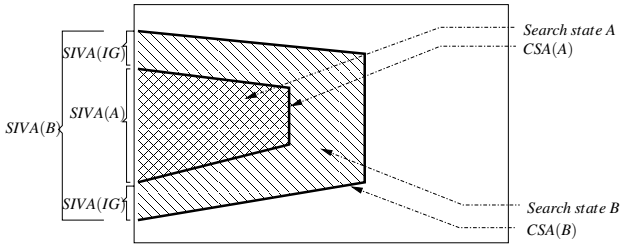

Fig. 3. Proof for Lemma 2

Theorem 1: If search-state A is conformable with search-state B, then $PreimCofSpace(B)$ implies $PreimCofSpace(A)$.

Proof: From Lemma 2, we can induce a search-state D such that $SIVA(D) = SIVA(A) \bigcup SIVA(IG)$ and $CSA(D) = CSA(B)$. Since $CSA(D) = CSA(B)$, $PreimCofSpace(D) = PreimCofSpace(B)$, from Lemma 1.

From search-state A, if we assign $SIVA(IG)$ to the primary inputs, we will reach search-state D and induce a cofactor $Cof(D)$. Other preimage cofactors are also present that can be obtained by different input value assignments, starting from A. A decision tree representation for the scenario is shown in Figure 4. Since the preimage cofactor space for A is a disjunction of terminal preimage cofactors, $PreimCofSpace(D) \subseteq PreimCofSpace(A)$. $\diamond$

IV. Search-space aware Cofactor Expansion

The relations among search-states discussed in the above section can be used to prune the redundant cofactor space that is encountered during preimage computation. In the following discussion, we analyze the situations where we can perform the search-state induced learning.

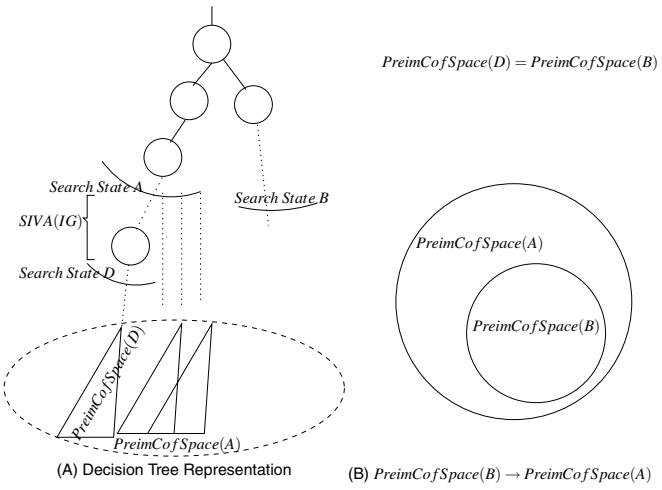

Fig. 4. Proof for Theorem 1

A. Search-state induced clauses

In the scenario shown in Figure 5(B), suppose we have obtained a search-state, A, at a branch E_1 in the decision tree. Then, we explore the branches under A to obtain the preimage cofactors for the logic decomposition C_{LA}. After backtracking from E_1, suppose we reach a branch E_2 and obtain a search-state B, such that A is conformable with B ($CSA(A) \subseteq GVA(B)$). Then, according to Theorem 1, the preimage cofactors that will be obtained from the logic decomposition, C_{LB}, is already contained in the preimage cofactor space under A. Therefore, we need not compute the preimage cofactors under C_{LB} and can immediately backtrack in the decision tree.

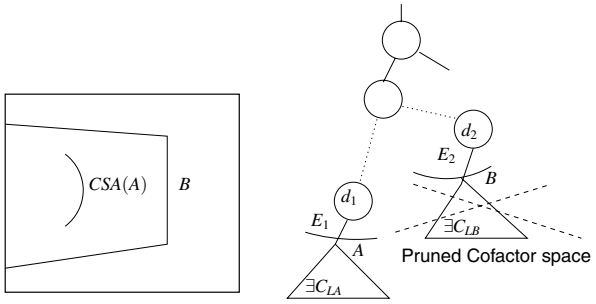

Fig. 5. Search-state induced learning

In order to take advantage of this kind of scenario, we derive a search-state induced clause from the cut-set of A. Let $CSA(A) = \{a_1, a_2, ..., a_k\}$ be the cut-set that induces the logic decomposition C_{LA}, i.e., $(a_1 \wedge a_2 \wedge ... \wedge a_k) \Rightarrow C_{LA}$. We can add the search-state induced clause $(\neg a_1 \vee \neg a_2 \vee ... \neg a_k)$ to block the preimage cofactor space in C_{LA}. Furthermore, when we reach any cut-set B, such that A is conformable with B (i.e.

$CSA(A) \subseteq GVA(B)$), this added search-state induced clause will be violated and it will help to backtrack immediately. However, it should be noted that this clause is not equivalent to C_{LA}. Therefore, it is only claimed that we will not reach the same logic decomposition structurally in the circuit. This is analogous to the fact that we can reach the conflict terminal multiple times during *SAT* search and each conflict terminal induces different conflict clauses.

We store the complement of cut-sets at each branch in the decision tree as search-state induced clauses in a clause database. We simply backtrack if any of these clauses are unsatisfied during the logic simulation of input decisions in our approach. We do not perform Boolean Constraint Propagation (BCP) on these clauses, since the unit implications on the internal variables in the circuit may violate the search-state analysis discussed in the previous section.

B. Non-chronological backtracking

The fundamental idea for our non-chronological backtracking is as follows: Let $L_I \subseteq (\alpha : I \mapsto \{0,1\})$ be a set of literals that is obtained by assigning Boolean values to a subset of the primary input variables. Consider partial input assignments, $\omega_1 = \bigwedge_{k1=1}^{n_1} l_{k_1}$ and $\omega_2 = \bigwedge_{k_2=1}^{n_2} l_{k_2}$ such that $l_{k_1}, l_{k_2} \in L_I$. Let $l_z \notin L_I$ be a literal which is a Boolean assignment to a primary input variable currently outside of L_I. If $(\omega_1 \wedge l_z) \Rightarrow C_1$ and $(\omega_2 \wedge \neg l_z) \Rightarrow C_2$, then by resolution, $(\omega_1 \wedge \omega_2) \Rightarrow (C_1 \vee C_2)$, where C_1, C_2 are circuit cofactors for the corresponding search-states.

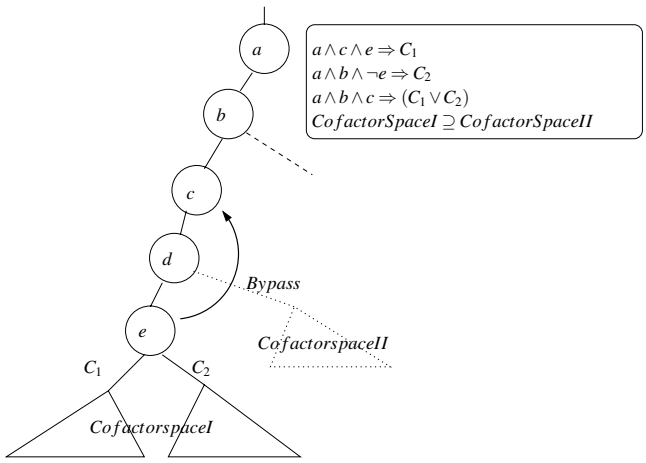

Fig. 6. Non-chronological backtracking

This scenario is shown in the decision tree in Figure 6. Suppose we make decisions a, b, c, d, e and compute the cofactor C_1. By implication analysis on the cutset, we find that $a \wedge c \wedge e \Rightarrow C_1$. Then we backtrack at e and compute the cofactor C_2. Again, by implication analysis, we find that $a \wedge b \wedge \neg e \Rightarrow C_2$. Then, by resolution, $a \wedge b \wedge c \Rightarrow C_1 \vee C_2$. $C_1 \vee C_2$ is the complete cofactor space available under the branch c. Therefore, after computing the circuit cofactors for C_1 and C_2, we can bypass the other branch of d and directly return to c because the cofactor space in the second branch of d is already contained in $C_1 \vee C_2$. We can thus avoid the redundant cofactor computation by backtracking non-chronologically.

In our implementation, we use bit vectors to record the decision levels that are responsible for each gate assignment, similar to [20]. The length of the bit vector for a gate is generally equal to the decision level when it is assigned. The bit positions correspond to the antecedent decisions responsible for the gate assignment. For each decision, we assign a 1 to the bit in the position of the decision level. Then, during logic simulation we propagate the bit vectors to reflect the antecedents responsible for each gate assignment. The bits in the antecedent decision levels are assigned a 1 and the other bits are assigned a 0. The OR of all bit vectors of the gate assignments in a cut-set corresponds to the decisions that implied the cut-set. After computing each circuit cofactor, we directly backtrack to the highest decision level in the cut-set bit vector.

V. Experimental Evaluation

We implemented the proposed implicit cofactor expansion technique in C on top of a publicly available verification tool called ABC [9] to compute the preimage iteratively, until a fixed point is reached. For each iteration, we construct the preimage computation model shown in Figure 1 as an AND-inverter graph. We use DAG-aware rewriting and other low-cost structural optimization techniques to obtain a compact model. The preimage that is computed is also stored as an AND-inverter graph and used in the next iteration. On the same common platform, we implemented three other preimage computation techniques: all-solutions *SAT* solver on top of MINISAT *SAT* solver [2], all-solutions ATPG with success driven learning [8] and a hybrid *SAT* solver [4]. For the *SAT*-solver, we use circuit justification [3] to deduce the blocking clause and an efficient ZBDD-to-clause conversion technique proposed in [14]. For this approach, we limit the number of clauses to 1 million and the number of ZBDD nodes to 200K. For the all-solutions ATPG, we limit the number of the gates in the preimage state-set to 125K and the size of the hash table to 1 million. For hybrid *SAT* solver, we limit the number of conflict clauses to 1 million and the number of gates in the preimage state-set to 125K. For our approach, we limit the number of search-state induced clauses to 1 million and the number of gates in the preimage state-set to 125K. We conducted a set of experiments on ITC'99 and ISCAS'85 benchmark circuits for some of the hard-to-reach states and VIS invariant properties, on a Pentium IV, 3GHz machine running the Redhat Linux Operating System. We ran each technique for 20,000 seconds.

In Table I, it is seen that *SAT times out* for most of large circuits, with dense reachable states (many preimage state cubes per iteration), since they have to compute a large number of solutions and block them cube by cube. Furthermore, because they consider the latches during solution enumeration (and the number of latches generally is greater than the number of primary inputs), they have to explore a larger space. Similarly,

TABLE I

PREIMAGE COMPUTATION UNTIL CONVERGENCE

Ckt	# ltch	*SAT* [2]		ATPG [8]		Hybrid [4]		Our approach	
		dpth	Time(s)	dpth	Time(s)	dpth	Time(s)	dpth	Time(s)
s3384	183	1*	TO	1*	9(MO)	4	3.0	4	1.8
s1423	74	2	307.8	1*	11(MO)	2	0.20	2	0.09
s5378	164	1*	TO	1*	154(MO)	4*	2K(MO)	8	69.7
s4863	104	1*	TO	1*	94(MO)	1*	14K(MO)	2	1.7K
s1269	37	1*	19K(MO)	1*	302(MO)	8	229.7	8	272.3
s3271	116	1*	TO	1*	473(MO)	6	15	6	723
s420.1	16	51	11	51	27.1	19*	2.7K	51	107.1
b07	45	2*	TO	29	1160.4	10*	7K(MO)	29	8.7K
b08	21	19	4.2	19	6.3	19	990.3	19	173.3
b09	28	15*	TO	21	4.7	11*	TO	21	371.3

*MO - Memory Out; TO - Time Out; * - incomplete;*

the ATPG technique needs to store a huge number of search-states to perform learning, since they also branch on both the circuit's input and latch variables. This can lead to memory explosion in many of the larger circuits. On the other hand, the hybrid solver and our approach block the cofactors and prune a larger search space. However, the hybrid approach also searches the solution space, even though it blocks the cofactor space. Thus, such solution enumeration again leads to memory out in s5378 and s4863 due to the huge number of conflict clauses that were generated. On the other hand, we were able to compute the cofactors quickly and backtrack from that cofactor space due to search-state induced learning. Although we computed a larger number of cofactors that were overlapping in the solution space, we save the time required to find a particular solution as compared to the hybrid *SAT* solver. For circuits such as s5378, s4836, b07, and b09, our implicit circuit cofactoring technique can lead to one to several orders of magnitude improvement in both run-time and memory.

For the smaller circuits with sparse reachable states (few preimage state cubes per iteration), the solution-based techniques perform significantly better than the cofactor-based approaches. In circuits such as s420.1 (counter) and b08 (finds inclusions in sequence of numbers), *SAT* performs significantly better than our approach. This is mainly because the SAT approach can find the smaller subset of satisfiable solutions quickly. In general, the implicit cofactor expansion technique is more suitable for circuits with dense reachable states, since each cofactor may cover a huge number of preimage states (i.e. a large number of solution cubes).

VI. CONCLUSION

In this work, we presented a novel preimage computation technique that directly quantifies the primary inputs in the circuit and computes the cofactors required for preimage computation. We used the relations between the search-states and the circuit cofactors to prune the cofactor space and perform non-chronological backtracking. Experimental results show that the implicit cofactor expansion technique performs significantly better than existing approaches for large circuits, where a large number of preimage states needs to be computed at each iteration.

REFERENCES

[1] Biere, A. et al.: Symbolic Model Checking using SAT procedures instead of BDDs. Proc. Design Automation Conference, pp. 317–320, 1999

[2] McMillan, K.L.: Applying SAT methods in unbounded model checking. Proc. of Computer Aided Verification, pp. 250–264, 2002

[3] Kang, H.-J. and Park, I.-C: SAT-based unbounded symbolic model checking. Proc. of Design Automation Conference, pp. 840–843, 2003

[4] Ganai, M. et al.: Efficient SAT-based Unbounded Symbolic Model Checking using Circuit Cofactoring. Proc. of International Conference on Computer Aided Design, pp. 510–517, 2004

[5] Wang, D. et al.: Formal property verification by abstraction refinement with formal, simulation and hybrid engines. Proc. of Design Automation Conference, pp. 6, 2001

[6] Chauhan, P. et al.: Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis. Proc. of Formal Methods in Computer Aided Design, pp. 33–51, 2002

[7] McMillan, K.L. and Amla, N.: Automatic abstraction without counter examples. Proc. of TACAS, pp. 2–17, 2003

[8] Sheng, S. and Hsiao, M.S.: Efficient Preimage Computation using a novel success-driven ATPG. Proc. of Design Automation and Test in Europe, pp. 840–843, 2003

[9] Mischenko, A.: ABC: A System for Sequential Synthesis and Verification, Release 50905. http://www.eecs.berkeley.edu/ alanmi/abc/

[10] Abdulla, P.A. et al.: Symbolic Reachability Analysis based on SAT solvers. Proc. of TACAS, pp. 411–425, 2000

[11] Williams, P. et al.: Combining Decision Diagrams and SAT procedures for efficient Symbolic Model Checking. Proc. of Computer Aided Verification, pp. 124–138, 2000

[12] Cabodi, G. et al.: Circuit Based Quantification: Back to State Set Manipulation within Unbounded Model Checking, Proc. Design Automation and Test in Europe, pp. 688-689, 2005

[13] Gupta, A. et al.: SAT-based Image Computation with Application in Reachability Analysis. Proc. of Formal Methods in Computer Aided Design, pp. 354–371, 2000

[14] Chandrasekar, K. and Hsiao, M.S.: State set Management for SAT based Preimage Computation. Proc. of International Conference on Computer Design, pp. 585–590, 2005

[15] Jin, H. et al.: Efficient Conflict Analysis for Finding All Satisfying Assignments of a Boolean Circuit. Proc. of TACAS, pp. 750–753, 2005

[16] Jin, H. and Somenzi, F.: Prime Clauses for Fast Enumeration of Satisfying Assignments to Boolean Circuits.

[17] Chandrasekar, K. and Hsiao, M.S.: ATPG based Preimage: Efficient search space pruning using ZBDDs. Proc. of High Level Design Validation and Test, pp. 117–122, 2003

[18] Li, B., Hsiao, M.S. and Sheng, S.: A Novel All-solutions SAT solver for efficient Preimage Computation. Proc. of Design Automation and Test in Europe, pp. 272–277, 2004

[19] Bjesse, P. and Borälv, A.: DAG-aware Circuit Compression for Formal Verification. Proc. of International Conference on Computer Aided Design, pp. 42–49, 2004

[20] Kuehlmann, A. et al.: Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification. IEEE Trans. CAD, vol. 21, no. 12, pp. 1377–1394, 2002