Fast Illegal State Identification for Improving SAT-Based Induction *

Vishnu C. Vimjam and Michael S. Hsiao
{vvimjam, mhsiao}@vt.edu
ECE Department, Virginia Tech
Blacksburg, VA, 24061

ABSTRACT

In this paper, we propose a novel framework to quickly extract illegal states of a sequential circuit and then use them as constraints during the SAT-based induction runs. First, we employ a low-cost combinational ATPG to identify unreachable partial-states among groups of related flip-flops. Second, we propose the concept of necessary-assignment looping to identify additional unachievable partial-states. Third, we extend the above unachievability theory to capture new non-trivial sequential logic dependencies among the circuit signals. Finally, we use a unified framework that utilizes all the above information and aims at maximizing the learning. All the learned illegal states are converted into constraint clauses and are replicated at all the unrolled transition relations to prune the search-space. Experimental results show that, due to the added constraints, many safety properties can be proved at earlier depths and the induction run-times can be significantly reduced.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids - *Verification*

General Terms

Algorithms, Verification

Keywords

Induction, SAT, ATPG, Learning

1. INTRODUCTION

In recent years, Model Checking [1] has shown promise in finding subtle bugs in complex designs. In model checking, the design to be verified is modeled as a Kripke structure (or an FSM) and the properties are written in temporal logic. The states of the design are then traversed to verify these properties formally. Unbounded model checking, such as Symbolic Model Checking [2], and SAT-based Bounded Model Checking [3] are two efficient, but different approaches of state-space traversal in model checking. If the property being checked is an invariant, however, the complexity of both of these approaches can increase significantly, since a complete reachability analysis is needed to prove such a property.

Induction-based techniques [4, 5] have been proposed to enable complete proofs for invariants. In contrast to the standard model checking techniques, induction-based techniques attempt to prove a property without performing state-space traversals. For example, in [5], equivalence of certain internal signals is first assumed, and BDDs are used to check if those equivalences hold indefinitely, *i.e* if they are inductive invariants. A complete technique for induction using SAT has been introduced in [6] and has been effectively used to verify FPGA cores. In [7], the technique of [5] has been generalized for safety property checking using SAT and several new improvements are also proposed. A new scheme called Explicit induction has been proposed in [8] to prove temporal safety properties. Although SAT-based induction has been effectively applied in the past, it suffers from one main limitation, namely the lack of knowledge of illegal state-spaces. Incremental learning techniques such as [10–12] aid in speeding up the induction runs but do not have the capabilities to prove a property at earlier depth than required otherwise.

In this paper, we propose new and efficient techniques to address the above limitation in SAT-based induction. In particular, we employ low-cost learning techniques based on (i) combinational ATPG, (ii) necessary assignment looping and (iii) static sequential logic implications to efficiently capture illegal partial-states[1] of a design. These are then converted into constraint clauses and replicated throughout the unrolled transition relations during the induction runs. Experiments conducted on ISCAS89 and ITC99 benchmarks show that, for many safety properties, our learning can aid in completing the proofs at earlier depths, whereas the conventional runs failed. This lead to significant savings in the execution times.

The rest of the paper is organized as follows: Section 2 provides the background on SAT-based Induction followed by the motivation behind this work. In Section 3, we present all our learning techniques and algorithms followed by the experimental results in Section 4. In Section 5, we conclude the paper.

2. SAT-BASED INDUCTION

Due to the recent advances in Boolean Satisfiability, induction using SAT procedures has been explored [6, 7]. Let $I(s)$ be the initial state(s) of a system, $T(s, s')$ be its transition relation, and ϕ be a property to be verified. Given an unrolled circuit instance of length k, let T_i and ϕ_i respectively denote the transition relation and the property assertion at step i. Then, the Boolean formula for the complete unrolled instance with the property can be constructed

*supported in part by NSF Grants 0305881, 0417340, 0524052, and SRC Grant 2005-TJ-1359.001.

DAC 2006, July 24–28, 2006, San Francisco, California, USA.

[1]Boolean value assignments to a subset of flip-flops

by $\{\phi^*(1,2,..,k) \wedge T^*(1,2,..,k)\}$, where $\phi^*(1,2,..,k) = \phi_1 \wedge \phi_2 \wedge .. \wedge \phi_k$ and $T^*(1,2,..,k) = T_1 \wedge T_2 \wedge .. \wedge T_k$. In SAT-based induction [6], the base case $I(s) \rightarrow \phi$ and the induction step $\phi_1 \wedge T_1 \rightarrow \phi_2$ are translated to Boolean formulas $I(s) \wedge \neg\phi$ and $\phi_1 \wedge T_1 \wedge \neg\phi_2$, respectively. These two formulas are checked for satisfiability using a SAT solver. If the base case formula is satisfiable, then one can readily conclude ϕ is not an invariant. Otherwise, the induction step is performed. If this is unsatisfiable, ϕ can be concluded as an invariant. Else, nothing can be concluded about ϕ. For most properties, however, this *simple induction* scheme is insufficient, and a stronger induction scheme called *induction with depth* can be used [6]. In this new scheme, the base case is modified to $I(1) \wedge T^*(1,2,..,k) \rightarrow \phi^*(1,2,..,k)$, and the induction step is modified to $\phi^*(1,2,..,k) \wedge T^*(1,2,..,k+1) \rightarrow \phi_{k+1}$, both of which are again converted into respective Boolean formulas. (Here, $I(1)$ denotes the initial state assertions at the present-state elements of T_1). If nothing can be concluded about ϕ at a depth k, it is increased to a higher depth k' and the SAT checks are repeated. To make the procedure complete, the search needs to be limited to simple paths. This is conducted by adding *unique state constraints* [6]. For finite-transition systems, if ϕ is an invariant, there exists a finite k such that the formula for the induction step is unsatisfiable.

2.1 Motivation

There is a main drawback for SAT-based induction, which is the lack of knowledge about the illegal state-space of the design. This limitation manifests itself in two ways. First, since the SAT-solver relies on a branch-and-bound procedure, the SAT engine may make poor decisions and later learn that the decisions lead to a functionally impossible (or illegal) space. Hence, the knowledge of illegal states before-hand can constrain the SAT-search. Second, if the formula for the induction step at a depth k is satisfiable, the satisfying solution forms a simple path s_1, s_2, ..., s_{k+1} such that the first k states satisfy ϕ but the last state s_{k+1} does not. In such a scenario, two cases are possible: (i) the first state s_1 is a reachable state, and hence ϕ is definitely not an invariant; (ii) the first state s_1 is an illegal state in which case the counter-example obtained is spurious and nothing can be concluded about ϕ. However, it is difficult to know the reachability of s_1. In [9], the authors obtain a new property $\phi \wedge \neg s_1$ with the intention that the new property is stronger than the original and hence might be proved at an earlier induction depth. However, if the induction step run has several such spurious counter-examples (which could be exponential), the new property might still require a large depth to be proved.

Due to the reasons discussed above, SAT-based induction can be improved by learning many illegal states and using them to constrain the search as close as possible to the reachable space. Since a thorough reachability analysis (even over-approximate) can be prohibitively expensive, we try to gather under-approximate illegal state information of a design, via an alternative, low-cost analysis.

3. ILLEGAL STATE IDENTIFICATION

In this section, we describe our techniques for learning illegal states. We use the term *time-frame* to represent one transition relation of a sequential circuit. Let B={0, 1} represent the set of Boolean logic values. The notation $X_{v.\tau}$ is used to represent a signal X set to value $v \in B$ in a given time-frame τ. If the parameter τ is not relevant, we simply use X_v. Also, we assume all the designs are initializable (either by an initialization sequence or via an explicit reset signal). This ensures that the system has only one terminally strongly connected component (TSCC).

We refer to logic simulation as 3-valued logic propagation done using the three values {0, 1, U}, where U is the unknown logic value. A signal X is said to be *specified* if it attains a known logic value $v \in B$; otherwise it is *unspecified* (or unknown). A composite signal assignment, $F = \{X1_{v1} \wedge X2_{v2} \wedge ... \wedge Xn_{vn}\}$, where $v1, v2, ..., vn \in B$, is said to be *achievable* if starting from an all unknown initial state, there exists at least one *finite* input vector sequence that can set $F = 1$ in a finite time-frame τ (*i.e.* to specify $X1 = v1, X2 = v2, ..., Xn = vn$ in τ). Otherwise, F is said to be *unachievable*. For an initializable design, at least one of F or $\neg F$ will be achievable. If F is unachievable, $\neg F$ will be an invariant.

3.1 Learning Using An ATPG

In this subsection, we will explain our ATPG based illegal state identification technique. The following definition forms the basis of our learning.

DEFINITION 1: (**n-Cycle-Unreachable**) A state s is said to be n-cycle-unreachable if the n^{th}-level preimage of s is empty.

In other words, if there does not exist any state that can reach s in n cycles, then s is unreachable. Note that this definition is a simpler form of the complete induction seen in section 2. If a state is n-cycle-unreachable, it is also (n+1)-cycle-unreachable, and so on. We employ a combinational ATPG-based method (using the PODEM[2] algorithm [13]) to quickly identify a subset of such states. The main advantage is that the learned information can be used repeatedly to constrain all the state-spaces without any additional analysis. Another added benefit is that, states that can be reachable only from the learned unreachable states can also be avoided by the SAT-solver during its search.

We unroll a circuit for n time-frames (similar to an Iterative Logic Array expansion) as shown in Figure 1(a), where the unrolled frames are numbered 1, 2, ..., n. The lines PI(i), PO(i), PS(i), NS(i) represent the primary inputs, primary outputs, present-state elements, next-state elements for frame i respectively. All the present-state elements for frames 2 to n are simply treated as buffers. We can embed a state s at PS(n + 1) and try to justify it while treating the state elements at PS(1) as pseudo-primary inputs (PPIs). In such a scenario, if no input assignment can justify the state s, it is n-cycle-unreachable. This is because, allowing the flip-flops at PS(1) to be fully controllable assumes all the states are possible at that boundary and hence guarantees $\neg s$ is an invariant. However, pre-image computations usually have exponential complexity and it will be infeasible to compute them even for a

[2]PODEM backtraces from a given objective (say, X_v) and makes decisions on the primary inputs to justify it (*i.e.* to set $X = v$)

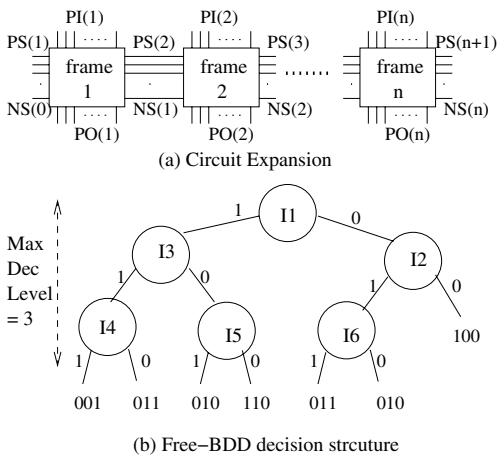

Figure 1: ATPG-based learning

```
0: Given n, MDL;  unroll circuit n times
1: Order flip−flops at PS(n+1) and unmark them
2: Compute SCOAP values (C0, C1) for each flip−flop
3: while (not done) do
4:    currDL = 0;  Initialize all signals to value U
5:    ModifiedPODEM();    // build a free−BDD
6:     LearnIllegalStatesFromFreeBDD();
7:    Mark flip−flops for which free−BDD is complete
8:    If all flip−flops marked, then done;
9: end while

10: ModifiedPODEM()
11:    if (currDL = MDL) then return;
12:    currDL = currDL + 1
13:    X = Next unspecified flip−flop in the order
14:    if (C0(X) > C1(X)) then obj=0; else obj=1
15:    (pi, val) = Backtrace(X,obj);
16:    LogicSim(pi=val);
17:    ModifiedPODEM();        // recurse
18:    LogicSim(pi=not(val));
19:    ModifiedPODEM();        // recurse
20:    LogicSim(pi=U);
21:    currDL = currDL−1
22: return;
```

Figure 2: n-cycle-unreachable learning algorithm

small subset of states. In order to identify unreachable states in an efficient manner, we propose a *state-independent* procedure as explained below.

Without loss of generality, consider the flip-flops at PS($n+1$) be arranged in an order $\{X1, X2, X3, .., Xn_{ff}\}$, where n_{ff} is the number of flip-flops. Our main goal is to perform an ATPG search without targeting any particular state. We start with the first flip-flop $X1$ set to an objective logic value (say, 1) and start the search process (*i.e.* justifications via PODEM). If $X1_{1.n+1}$ is justified during the decision process, we proceed to justify $X2$ to a value (say, 0) and continue. In this way, the entire decision tree is constructed like a free-BDD[3] as shown in Figure 1(b), where the nodes represent the decision variables ($I1$, $I2$, .., $I6$ can be any primary input or a PPI) and the leaves indicates the logic values attained by the flip-flops at PS($n+1$).

Consider a group of 3 flip-flops ($X1$, $X2$, $X3$) at PS($n+1$) were *all* specified at *all* leaves of the decision structure. Figure 1(b) shows some example values. It can be seen that the partial-states 000, 101 and 111 were not achieved at any of the leaves. Since the decision tree is complete for *specifying* all these 3 flip-flops, these three states can be concluded as illegal. In other words, $\{(\overline{X1} \wedge \overline{X2} \wedge X3) \vee (\overline{X1} \wedge X2 \wedge \overline{X3}) \vee (\overline{X1} \wedge X2 \wedge X3) \vee (X1 \wedge \overline{X2} \wedge \overline{X3}) \vee (X1 \wedge X2 \wedge \overline{X3})\}$ is an invariant. One way of strengthening this invariant is by increasing n, due to which more illegal states can be ruled out. Another way is to eliminate the already known illegal paths (and the corresponding leaves) of the decision structure. For example, if $I1$ and $I2$ of Figure 1(b) are PPIs and if we know that $\{I1_0 \wedge I2_0\}$ is an illegal partial-state, then $\{X1_1 \wedge X2_0 \wedge X3_0\}$ is also an illegal partial-state. This is because the partial-state 100 can be obtained via only illegal paths of the free-BDD (in this case, only one such path). In our implementation, we store the illegal states learned in the iterations $n = 1, 2, .., i-1$ and use them to eliminate the illegal paths of the free-BDDs in the iteration $n = i$. All the illegal partial-states accumulated till the end of the last iteration N (user-defined) are converted into constraint clauses.

Figure 2 shows our learning algorithm for a given n. Due to space limit, we do not show the above invariant strengthening technique. To keep the computational cost low, we limit the maximum decision level (MDL) to a user-defined value (currently set to 20). After each ATPG run, we obtain a free-BDD from which unreachable partial-states are extracted. We use a technique similar to that of [14] to form the free-BDD. We refer the reader to [14] for details. All the flip-flops that are specified at *all* the leaves of the free-BDD are removed from the order and are not considered in the remaining ATPG runs. Within a given MDL limit, if no such flip-flop exists, we forcefully remove the first flip-flop in the order. This is done to ensure that the algorithm terminates within a maximum of n_{ff} calls to the ModifiedPODEM() procedure.

We use the SCOAP measures [15] to identify if a flip-flop is most controllable to logic 0 or logic 1. Heuristically, we always try to justify a flip-flop to its most controllable value. This is done to specify a value at that flip-flop with as few decisions as possible (recall the MDL limit). The MLP procedure [16] is used to obtain the initial ordering of the flip-flops. This procedure computes the input supports for the flip-flops and clusters the ones with closer supports. For our purpose, note that an initial ordering is important, because, once a flip-flop is specified, the next flip-flop that we choose should be co-justifiable easily. The overall complexity of the above algorithm is $O(n_{ff}.2^{MDL})$, which can be be adjusted according to the user-specified value for MDL.

Dynamic Regrouping. Since MLP uses only structural analysis to order flip-flops, we have observed that the ordering may not always help us in justifying a suitable group of flip-flops together. For example, if flip-flops $X1$ and $X2$ are adjacent in the order produced by MLP, it is possible that within MDL, there exists several paths of the free-BDD, at whose leaves $X1$ is specified but not $X2$. This happens when the flip-flop $X2$ does not group well with $X1$ and more decisions are needed along those paths to specify $X2$. In such cases, we will not learn anything illegal involving $X1$ and $X2$. We use a procedure to dynamically regroup the flip-flops as our analysis progresses. Essentially, if we were not able to specify $X2$ at the leaf of a path p during the decision process, we check if any other flip-flops in the lower order have been specified at *all* the leaves of the previously constructed $p-1$ paths and as well at the current path p. If so, all such flip-flops are moved to the position next to $X1$, and $X2$ is moved down the order. If no such flip-flop exists, however, we continue as shown in our regular algorithm (see Figure 2). The main intuition behind this dynamic approach is to group flip-flops which are closely related in the Boolean space rather than via circuit structure. The final ordering obtained at the end of unroll depth $n-1$ is used as the initial ordering for the next iteration n.

At each leaf, L, of a free-BDD built in Figure 2, we obtain a set of assignments to the flip-flops at PS($n+1$). Let this be partial-state $F1$. Let the flip-flop assignments along the path of the free-BDD for leaf L form a partial state $F2$. We take the intersection of the assignments in $F1$ and $F2$ (say, $F3$) and store them as a list, $LIST$. All such stored partial-states are used in our unified framework as explained in section 3.4.

3.2 Unachievable Partial-State Learning

In this subsection, we explain our necessary assignment looping theory to learn additional unachievable partial-states. First, consider the concept of a general logic implication, which describes the logical dependencies among the signals in a circuit. For example, the implication $X_{0.\tau} \rightarrow Y_{1.\tau+1}$ means that whenever signal X is set to logic 0 in frame τ, the only possible value at signal Y in frame $\tau+1$ is a logic 1. We define the concept of a *sufficient* implication as follows.

DEFINITION 2: (**Sufficiency**). Starting at an all unknown initial state, let I be an input vector sequence of length τ. The implication

[3] a BDD where all the paths need not have the same variable order

$X_{v.\tau} \rightarrow Y_{w.\tau-k}$ $(v, w \in B,\ k \geq 0)$ is termed *sufficient* if any input vector sequence I that implies $X_{v.\tau}$, also implies $Y_{w.\tau-k}$. Formally, $\forall I : I \rightarrow X_{v.\tau} \Rightarrow I \rightarrow Y_{w.\tau-k}$.

Not all the logical implications inside a circuit are sufficient. For example, consider the simple circuit shown in Figure 3 where E_0 is a constant assignment. The implication $D_1 \rightarrow E_0$ holds true but is not sufficient. This is because D_1 can be achieved by an input vector $\{A = U, B = 1\}$ which doesn't cause E_0 via logic simulation. The figure shows a few other sufficient and non-sufficient example implications.

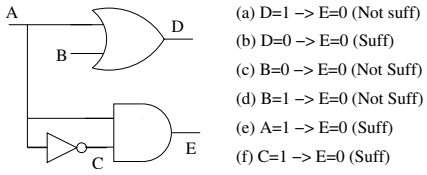

Figure 3: Sufficient implication examples

Next, we use the above sufficiency condition and propose the following theorem for obtaining unachievable partial-states.

Theorem 1: *A partial-state* $F = \{X1_{v1} \wedge X2_{v2} \wedge .. \wedge Xn_{vn}\}$, *(*$v1$, $v2$, .., $vn \in B$*), consisting of* n *flip-flop assignments is unachievable, if for any* τ*, the following sufficient implication exists:* $(X1_{v1.\tau} \wedge X2_{v2.\tau} \wedge ... \wedge Xn_{vn.\tau}) \rightarrow (X1_{v1.\tau-k} \wedge X2_{v2.\tau-k} \wedge ... \wedge Xn_{vn.\tau-k})$, $k > 0$.

Proof: We prove this by contradiction. Starting from an all unknown initial state at frame 1, let a finite input vector sequence I of length l achieves $F = 1$ in frame l. According to our sufficient implication above, I should also set $F = 1$ in frame $l - k$, which in turn requires $F = 1$ in frames $l - 2k$, $l - 3k$, ..., $l - nk$ and so on, where $n \rightarrow \infty$. For some value of n, $nk \geq l$ which means that F has to be true in a frame ≤ 1. This, however, requires I to have a length $> l$.

[(length(I) = l) $\rightarrow$ (length(I)> l) $\Rightarrow$ (length(I) = ∞)].

In other words, no finite input vector sequence can achieve F. $\diamond$

For example, let for any τ, $(X_{0.\tau} \wedge Y_{1.\tau}) \rightarrow (X_{0.\tau-1} \wedge Y_{1.\tau-1})$ be a sufficient implication. Then, by the contrapositive law, $(X_{1.\tau-1} \vee Y_{0.\tau-1}) \rightarrow (X_{1.\tau} \vee Y_{0.\tau})$. In other words, flip-flops X and Y can be initialized and remain only in the following states: {00, 10, 11}. Note that Theorem 1 holds true only if the implication is a *sufficient* one. Otherwise, we might not conclude unachievability. For example, if the combinational logic shown in Figure 3 is a part of a sequential circuit, then $E_{0.\tau} \rightarrow E_{0.\tau-k}$ holds true for any τ, any $k > 0$. However, it *cannot* be a *sufficient* implication according to definition 2. Simply put, an invariant cannot sufficiently imply itself backward. Otherwise, it will not be an invariant.

In a naive manner, we can inject a partial-state onto the circuit, imply its necessary logic values and check if it satisfies the criteria given in Theorem 1. However, checking this for all possible partial-state assignments is impractical even for medium-sized designs. While Theorem 1 is proposed with respect to partial states, it is equally applicable for any composite signal assignment in a circuit. Hence, we apply it efficiently during the static implication computation process and in our unified framework, as explained in the following sub-sections.

3.3 Exploiting Static Implications

Since the circuit information is encoded into a Boolean formula for performing the induction runs, it becomes beneficiary to extract non-trivial relations existing among the circuit signals (*i.e.,* those that cannot be deduced by the SAT solver directly through Boolean Constraint Propagation(BCP)) and inject them as learned clauses into the formula. This has been put to use in [17], where the implications learned via direct logic simulation are converted to clauses for enhancing the SAT runs. In our framework, we make use of the Extended Backward learning (EBL) procedure described in [18]. There are two main advantages behind this: First, the learning procedure used in [17] might miss many complex relations existing inside a circuit. Although the time consumed by the EBL procedure can be higher than that of [17], it has the potential to identify new non-trivial relations. Second, the unachievability theory described in the above sub-section can be efficiently employed during the EBL computation process. Note that the use of EBL is not a main contribution of this paper, but the means of applying our technique via EBL is.

```
0:  Given N(odd), unroll circuit N times;  n=(N+1)/2
1:  Order signals in frame n from PIs to POs
2:  for each signal X in order, for each v in {0,1} do
3:    LogicSim(X=v)
4:    UG = set of unjustified gates due to X=v
5:    for each gate G in UG with controlling value cv do
6:      for each unspecified fanin Fi of G do
7:        Si = set of assignments due to LogicSim(X=v, Fi=cv)
8:        if (X=v, Fi=cv) hold Thm 1, add (X=v)–>(Fi=not(cv))
9:      end for
10:     Add (X=v) –> (intersection of all Si)    // EBL step
11:   end for
12:   if (X=v) holds Thm 1, store X=not(v) as invariant
13: end for
```

Figure 4: Unachievability learning algorithm

Figure 4 shows our unachievability learning algorithm. Given N, the circuit is unrolled for N time-frames. Each signal assignment X_v $(v \in B)$ in the middle frame n is logic simulated and the set of unjustified gates[4] due to X_v are identified. If G is an unjustified gate with controlling value cv, each of its unspecified fanin Fi is set to value cv and logic simulated together with X_v. The EBL procedure (step 10) computes the intersection of signal assignments in all such logic simulations and stores them as implications of X_v. As soon as the logic simulation of $X_v \wedge Fi_{cv}$ is done, we check if it holds the criteria of Theorem 1. If so, the implication $X_v \rightarrow Fi_{\overline{cv}}$ is learned. At the same time, the contrapositive relation $Fi_{cv} \rightarrow X_{\overline{v}}$ is also learned. Note that all such implications learned via Theorem 1 are highly non-trivial since a SAT engine might not be able to deduce them during its decision making process. In our implementation, we store all the implications learned and use them to identify more signal assignments during logic simulations (at steps 3 and 7). The advantage of doing so is to learn more non-trivial implications when the algorithm proceeds for the remaining signals.

Finally, after all the implications are computed for a given signal assignment X_v, we check if X_v satisfies the criteria of Theorem 1. If so, $X_{\overline{v}}$ is stored as an invariant. All such invariants and learned implications are converted to clauses. For example, for a learned implication $X_0 \rightarrow Y_1$, the clause $(X \vee Y)$ is created. The original EBL algorithm has quadratic complexity in terms of the number of signals of the circuit. Checking our unachievability criteria only requires a small fraction of the run-time needed by EBL.

Figure 5 shows 2 time-frames of an example sub-circuit. The copy of a gate X in frame n is named $X^{'}$ in frame $n - 1$. The implications $X1_{0.n} \rightarrow H^{'}{}_{1.n-1}$ and $F_{0.n} \rightarrow I_{1.n}$ already exist in the circuit (shown by dotted lines). For simplicity, we do not show all the gates in both the frames. Consider the assignment $E_{1.n}$ due

[4] a gate is *unjustified* if it has a specified value but the current logic values of its fanins do not justify it

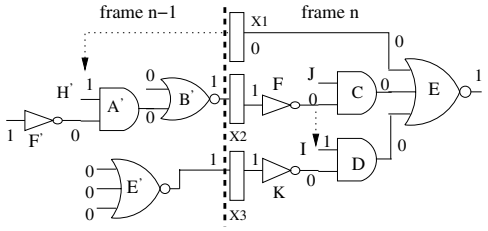

Figure 5: Necessary assignment looping example

to which the gate C (with logic value 0) is unjustified. The two possible justifications for $C_{0.n}$ are $J_{0.n}$ and $F_{0.n}$ (since both fanins J and F are unspecified currently). When $F_{0.n}$ is logic simulated along with $E_{1.n}$, the figure shows how the assignments $F'_{0.n-1}$ and $E'_{1.n-1}$ are also implied. Since we are implying the necessary values in the circuit, these are sufficiently implied and hence we can conclude $\{E_1 \wedge F_0\}$ as unachievable according to Theorem 1. Encoding this as a clause $(\neg E \vee F)$ into the induction formula helps in rejecting states that can cause these unachievable signal combinations. In the example shown, $\{X1_0 \wedge X2_1 \wedge X3_1\}$ is one such illegal state.

3.4 Unified Framework

In this sub-section, we provide a unified framework which helps us in extracting more illegal states by using the information learned so far. First, for each partial state $F3$ stored in $LIST$ (obtained from the ATPG procedure), we check if it satisfies Theorem 1. If so, $F3$ is marked unachievable and the corresponding clause is learned, which can be used to constrain the induction runs.

Second, we do the following: By performing the ATPG procedure in Section 3.1, we obtain groups of flip-flops and impossible combinations among the flip-flops in each group. From the procedure described in Section 3.3, we obtain non-trivial implications (from EBL as well as via Theorem 1). For each group of flip-flops obtained at the end of our ATPG analysis, we construct an ordered BDD as shown in Figure 6, where each BDD node shows a flip-flop with logic assignments 1 and 0 respectively. At each decision along a path, all the logic implications of the corresponding assignment are injected onto the circuit and logic simulated.

If the new signal assignments due to that decision are not consistent in the circuit, we store the partial-state (so far) as illegal and continue with the opposite assignment for that flip-flop. Otherwise, we check if the partial-state satisfies the criteria of Theorem 1. If so, we store the partial-state as illegal and continue as before. This process is repeated until the whole BDD is enumerated. In Figure 6, let the three flip-flops $X1$, $X2$, $X3$ be in one group (in that order) and let $\{X1_0 \wedge X2_1 \wedge X3_0\}$ be an unreachable partial-state learned via our ATPG analysis. As shown in the figure, let the assignment combination $\{X1_1 \wedge X2_1 \wedge X3_0\}$ cause a conflict during logic simulation where as the combination $\{X1_0 \wedge X2_0\}$

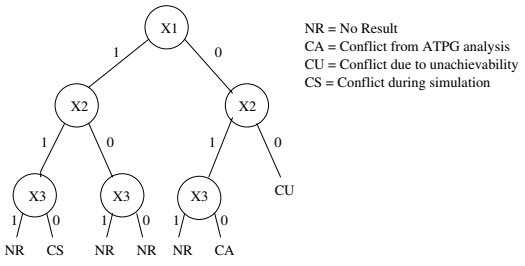

Figure 6: Enumeration BDD example

satisfies the unachievability criteria. Converting each illegal partial-state to clause form will lead to the three clauses $(X1 \vee \neg X2 \vee X3)$, $(\neg X1 \vee \neg X2 \vee X3)$ and $(X1 \vee X2)$ respectively.

Finally, we simplify all the clauses learned so far. Since we utilized BDD structures, there might be redundancies present in the resulting clauses. For example, from Figure 6, we can see that the partial-state $\{X2_1 \wedge X3_0\}$ itself is illegal (irrespective of $X1$). Thus the first two clauses learned above has a redundant literal in them. We utilize binary resolution [19] to remove such redundancies. For example, resolving on the variable $X1$ using those two clauses would lead to the clause $(\neg X2 \vee X3)$ which captures the unreachable core and maximizes the conflicting probability. Similarly, given the ATPG learning from Figure 1(b), we would learn the clause $(\neg X1 \vee \neg X3)$.

Thus, once we obtain all the constraint clauses, we resolve on each variable at a time and eliminate the redundant literals in them. The final set of clauses maximizes the learned constraints. Recent clause simplifying techniques such as NiVER [20] can as well be used for further simplification, however are not currently employed in our tool.

4. EXPERIMENTAL RESULTS

We have integrated all the above learning mechanisms into our framework IFILL (Induction using Fast ILLegal states). In IFILL, the procedures described in Sections 3.1, 3.3 and 3.4 are performed and the constraint clauses are obtained. Experiments for sequential circuits from the ISCAS89 and ITC99 benchmark suites were conducted using the zChaff [22] SAT solver (version 2004.11.15 taken from [23]) on a P4 3.2GHz machine with 1GB RAM and running Linux OS. We experimented with the hard safety properties, which are based on aborted state-justifications from a sequential ATPG. An aborted state is a state that the ATPG failed to justify within its resource limit. If s is an aborted state, the property was formulated as $AG(\neg s)$. We have neglected all those properties which can be proved easily using induction.

For the smaller circuits ($<$ 5000 gates), maximum unroll limit was set to 4 and 5 for the ATPG analysis and EBL procedure, respectively, whereas it was set to 3 and 3 for the larger ones. The induction depth k is increased by 25 frames per iteration up to a maximum of 250 frames for the smaller circuits. For the larger ones, these were set to 10 and 100 respectively. Time-out limit for each property was set to 30,000 seconds.

Table 1 reports the experimental results. For each instance shown, we report the induction run-times required by the original run (no learning), learning from [17], learning from EBL [18] and with our learning under columns *Orig*, *[17]*, *EBL* and *IFILL* respectively. In column *EBL*, only implications learned via EBL are added, whereas in ours, all the learned illegal states are added in addition. The sub-columns T_p and T_t denote the pre-processing and total times (T_p + solving time) in seconds. As a first observation, we can see that the pre-processing times needed by our method is only few seconds higher than that of EBL. In most cases, they are negligible when compared with the actual induction run-times.

If a property is verified via our technique, we report the corresponding bound in brackets under our sub-column T_t. Among the 26 properties shown, we were able to complete the proof at an early bound for 11 properties whereas the other three approaches could not. For example, for s13207.1, the original run, [17] and EBL took more than 30000, 28000, 10000 seconds respectively without any result, whereas we completed the proof within a total of 129 seconds. Note that, by being able to complete the proofs at smaller depths, the high run-times incurred in solving the higher depth instances are avoided.

Table 1: Induction runs for safety properties

Circuit.ϕ	Gates	Orig	[17]	EBL [18]		IFILL(Ours)	
		T_t	T_t	T_p	T_t	T_p	T_t
b10.1	206	83	118	1	8	2	**3**
b10.2	206	86	131	1	7	2	**3**
s526.1	223	3565	3434	1	1291	3	**3[50]**
s526.2	223	4034	4386	1	1599	3	**3[50]**
b13.1	362	534	89	1	16	2	**13**
b07.1	441	193	42	1	11	2	**8**
b07.2	441	219	43	1	20	2	**9**
b04.1	737	580	563	0	**491**	1	509
b04.2	737	768	491	0	**465**	1	507
s1423.1	753	7371	7616	1	8943	4	**7259**
s1423.2	753	7162	9874	1	7475	4	**6013**
b11.1	770	5154	4864	1	3405	2	**2[50]**
b11.2	770	5575	4800	1	2386	2	**2[25]**
b05.1	998	151	46	0	41	1	**1[50]**
b05.2	998	192	32	0	44	1	**1[50]**
s9234.1	5883	28015	21912	18	3033	25	**26[20]**
s9234.2	5883	21319	18682	18	6252	25	**26[20]**
s13207.1	8803	-TO-	28613	108	10712	129	**129[20]**
s13207.2	8803	-TO-	-TO-	108	8258	129	**130[20]**
b15.1	8922	**9037**	11985	490	18120	523	10188
b15.2	8922	**7547**	16851	490	10849	523	8398
b14.1	10098	597	505	48	101	57	**80**
b14.2	10098	1357	1110	48	594	57	**73**
s15850.1	10533	-TO-	-TO-	35	12124	46	**7387**
s15850.2	10533	-TO-	-TO-	35	16051	46	**8803**
s38417.1	23949	10211	7254	34	4787	48	**48[10]**

T_p: Preprocessing time T_t: Total time (with T_p) *TO*: $T_t > 30000$ sec

For most other properties for which nothing is concluded within the maximum depth, we were able to speed-up the induction runs. For example, for b15850.2, the original run, [17] and EBL took more than 30000, 30000 and 16000 seconds, respectively, whereas we were able to reduce it to 8803 seconds. For the two properties b15.1 and b15.2, adding the new clauses caused negative effects when compared with the original runs. This is because, all the added clauses increase the burden on the SAT engine during BCP and also possibly lead to a different decision ordering, which reduces its efficiency.

Next, we illustrate a cumulative run-time analysis (excluding T_p) for b14.2 with respect to the depth in Figure 7. Note that, we want the cumulative increase in run-time with depth to be as slow as possible. For smaller depths, all the runs seemed competitive (differing only little). But when the depth is increased beyond a certain limit (say, 60-70 in this case) the run-times for the other approaches increased almost exponentially, whereas we were still able to contain it. This demonstrates the efficiency of IFILL in enabling a deeper inductive search, under given resource limits.

5. CONCLUSION

We have proposed novel, low-cost learning techniques to extract under-approximate illegal state information of a design for improving SAT-based induction. All the learned illegal states are used as constraints in all transitions to focus the SAT search as close as possible to the reachable state space. Experiments revealed that our learning can verify properties at earlier depths whereas the conventional runs failed. We were also able to enhance the induction runs for a variety of other properties. Whereas IFILL can reduce the depths for true properties, it cannot improve those for false properties. This is because, the search in base cases is already restricted to the reachable space via the initial state constraints. In the future, we would like to explore in this direction.

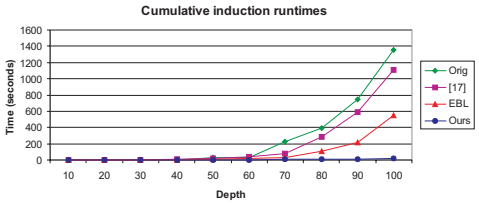

Figure 7: Cumulative run-times for b14.2

REFERENCES

[1] E. M. Clarke, O. Grumberg and D. A. Peled, "Model Checking". *The MIT Press*, 2000.

[2] K. McMillan, "Symbolic Model Checking: An Approach to the State Explosion Problem". *Kluwer Academic Publishers*, 1993.

[3] A. Biere, A. Cimatti, E. Clarke and Y. Zhu, "Symbolic Model Checking without BDDs". *Proc of Conf. on TACAS*, 1999.

[4] D. Deharbe and A. M. Moreira, "Using Induction and BDDs to Model Check Invariants". *Proc of Conf. on CHARME*, 1997.

[5] C. A. J. van Eijk, "Sequential Equivalence Checking without State-Space Traversal". *Proc of DATE*, 1998.

[6] M. Sheeran, S. Singh and G. Stalmarck, "Checking Safety Properties using Induction and a SAT Solver". *Proc of Conf. on FMCAD*, Nov 2000.

[7] P. Bjesse and K. Claessen, "SAT-based Verification without State-Space Traversal". *Proc of Conf. on FMCAD*, 2000.

[8] R. Armoni, L. Fix, R. Fraer, S. Huddleston, N. Piterman and M. Y. Vardi, "SAT-based Induction for Temporal Safety Properties". *Proc of BMC*, 2004.

[9] L. de Moura, H. Rueß and M. Sorea, "Bounded Model Checking and Induction: From Refutation to Verification". *Proc. of Conf. on CAV*, 2003.

[10] O. Strichman, "Accelerating Bounded Model Checking of Safety Properties", In *Formal Methods in System Design*, vol 24, pp. 5-24, Jan 2004.

[11] N. Een and N. Sorensson, "Temporal Induction by Incremental SAT Solving". *Proc of BMC*, July 2003.

[12] L. Zhang, M. R. Prasad and M. S. Hsiao, "Incremental Deductive & Inductive Reasoning for SAT-based Bounded Model Checking". *Proc of ICCAD*, pp. 502-509, 2004.

[13] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits". *IEEE Trans. on Comp.*, vol. C-30, pp. 215-222, March 1981.

[14] S. Sheng and M. S. Hsiao, "Efficient Preimage Computation Using a Novel Success-Driven ATPG". *Proc of DATE*, pp. 822-827, 2003.

[15] H. Goldstein and E. L. Thigpen, "Scoap: Sandia Controllability/Observability analysis program". *Proc of DAC*, 1980.

[16] In-Ho Moon, G. Hatchel and F. Somenzi, "Border-block Triangular Form and Conjunction Schedule in Image Computation". *Proc of Conf. on FMCAD*, Nov 2000.

[17] R. Arora and M. S. Hsiao, "Enhancing SAT-based Bounded Model Checking using Sequential Logic Implications". *Proc of VLSI Design Conf.*, pp. 784-787, Jan 2004.

[18] J. Zhao, M. Rudnick and J. Patel, "Static Logic Implication with Application to Fast Redundancy Identification". *Proc of VTS Symp.*, 1997.

[19] M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory". *ACM Journal*, vol 7, pp.201-215, 1960.

[20] S. Subbarayan and D.K. Pradhan, "NiVER: Non-Increasing Variable Elimination Resolution for preprocessing SAT instances". Proc of Conf. on SAT, May 2004.

[21] E. Goldberg and Y. Novikov, "BerkMin: a Fast and Robust SAT-Solver". *Proc. of DATE*, pp. 142-149, 2002.

[22] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang and S. Malik, "Chaff: Engineering an Efficient SAT-Solver". *Proc of DAC*, 2001.

[23] Website: http://www.princeton.edu/ chaff/zchaff.html