Diagnosing Unreachable States Using Property Directed Reachability

Ryan Berryhill[1], Andreas Veneris[1,2]

Abstract— **In the modern design cycle, substantial manual effort is required to correct errors found when verification reveals an unreachable state. This work introduces two methodologies to automate this task. Given an unreachable target state, both methodologies return a set of design locations where changes can be implemented to make the target state reachable (*i.e.,* solutions). The first methodology uses intertwined steps of reachable state-space approximation, property checking, and traditional debugging to compute a subset of the solutions that make the target state reachable in some fixed number of clock cycles. The second methodology uses property-directed reachability with an enhanced version of the circuit's transition relation to compute the complete set of solutions to the problem. As an additional benefit, it returns an inductive invariant proving that no further solution(s) exist. The completeness of the approach comes at the cost of increased runtime when compared to the first methodology. Empirical results on industrial designs confirm the effectiveness of both approaches.**

I. Introduction

In modern hardware design, functional verification has grown to consume the majority of the design effort [1]. Debugging accounts for a substantial 60% of the verification cycle [2]. Most verification and debugging tasks are now partially or fully automated, somewhat mitigating the substantial engineering effort they demand. However, when verification reveals an error in the form of an unreachable state, identifying the root cause remains a predominantly manual task.

When functional verification reveals an error such as a firing assertion, observation signal value mismatch, or a scoreboard discrepancy an error trace is returned that demonstrates the failure. An automated debugging tool [3]–[6] can use this error trace to aid the engineer in finding the source of the error. Conversely, when a state is shown to be unreachable in violation of the specification, an error is clearly detected, but no error trace exists to guide an automated debugging tool. As a result, traditional debugging tools cannot be applied to accelerate the debug process.

Towards the goal of alleviating this problem, the work in [7] introduces an automated technique to facilitate diagnosis of unreachable states. The approach consists of intertwined steps of state-space approximation, traditional property-checking, and traditional SAT-based automated debugging [3]. Given an unreachable target state and a set of suspect locations, it computes a subset of the suspect locations where a change can be implemented to make the target state reachable (*i.e.,* solutions). First, property-directed reachability [8]–[10] is used to compute an initial over-approximation of the set of states reachable in a fixed number of clock cycles. Subsequently, a traditional SAT-based debugger is used to debug a sequence of state transitions from any state in the approximation to the target state. Due to the inherent nature of the state space approximation the approach may find spurious solutions, which are identified using property-directed reachability and discarded. This has the beneficial side effect of refining the state space approximation, reducing the chances of finding further spurious solutions. The approach additionally has a configurable tradeoff between runtime and resolution.

Since this technique may only provide a subset of the possible error sources, it limits the engineer's confidence that the results maximally preserve the engineering effort invested in the design. To address this concern, this paper presents an additional methodology that computes the complete solution set of the problem, but typically requires more runtime than the previous approach. Given an unreachable target state and a set of suspect locations, it returns every suspect location that can be changed to make the target state reachable in any number of clock cycles. As an added benefit, upon termination it returns an inductive invariant that proves no further solution(s) exist. This is accomplished by modeling the debugging problem as an unbounded model checking problem.

In greater detail, the methodology works as follows. First, an enhanced transition relation of the circuit is constructed by inserting error-select registers. Each error-select register is associated with a suspect location, such that the suspect location is effectively replaced by an arbitrary Boolean function when its respective error-select register is active. Non-active error-select registers do not change the behavior of the circuit. Additional constraints are added to ensure that only one error-select register is active at a time. From this construction, the target state is reachable under the enhanced transition relation if and only if a change can be implemented at one of the suspect locations to make it reachable in the original circuit. These locations represent solutions to the problem. To find the solutions, reachability of the target state under the enhanced transition relation is checked using Property-Directed Reachability (PDR). When the target state is reachable, a counter-example is returned in which an error-select register is active, indicating that the respective suspect location is a solution. The error-select register is then deactivated and PDR is executed again. This process is repeated until the target state is unreachable, indicating that no further solutions exist. At this point, PDR returns an inductive invariant proving this claim.

As this approach trades increased run time for completeness, it provides a valuable tradeoff to the user. If the entire solution set is needed, the engineer can apply the exact approach at the cost of increased runtime. However, the user may have sufficient knowledge of the design and intuition about the source of the error to guide debugging. In these cases, it is possible to use the approximate approach with appropriate parameter values to find the desired set of solutions more quickly. This allows the user to choose the methodology best suited to their particular problem while remaining confident that the results can be applied to accelerate debugging.

Experiments on industrial designs confirm the theoretical findings and the practicality of the presented approaches. The complete solution set of the problem is found to represent an average of only 10% of the suspect locations, substantially narrowing down the source of the error. The approximation-based approach provides a 4.8x speedup relative to the complete approach, and is able to find an average of 43.4% of the complete solution set.

The remainder of this paper is organized as follows. Section II presents background information regarding unbounded model checking and traditional automated debugging. Section III defines the problem and presents the approximation-based algorithm. Section IV presents the complete approach. Section V presents experimental results. Finally, Section VI concludes the paper.

[1]University of Toronto, ECE Department, Toronto, ON M5S 3G4 ({ryan, veneris}@eecg.toronto.edu)

[2]University of Toronto, CS Department, Toronto, ON M5S 3G4

II. Preliminaries

The following notation is used throughout this paper. Given a sequential circuit C, $S = \{s_1, ..., s_{|S|}\}$ denotes the set of state elements (registers) of C. Each assignment $t \in \{0,1\}^{|S|}$ to the state elements is a state of C. The transition relation of C is denoted $T \subseteq \{0,1\}^{|S|} \times \{0,1\}^{|S|}$. For a state pair $\langle t, t' \rangle$, $\langle t, t' \rangle \in T$ if and only if there exists an assignment to the primary input that causes C to transition from t to t'. Additionally, the propositional formula $t \wedge T \wedge t'$ is satisfiable if and only if $\langle t, t' \rangle \in T$. The set of initial states of C is denoted $I \subseteq \{0,1\}^{|S|}$. For a predicate P over the set of state variables S, any state $t \in P$ is referred to as a P-state.

For the purpose of model checking, C can be modeled by a Finite State Machine (FSM) $M = (S, I, T)$. A sequence of states $t_0, ..., t_n$ is a *trace* of M if and only if $\langle t_i, t_{i+1} \rangle \in T$ for all $0 \leq i < n$ and $t_0 \in I$. A state t is *reachable* under M if it appears in a trace of M. It is also *i-step reachable* if it appears in a trace of i cycles or less. In this paper, the predicate R_i denotes the set of i-step reachable states.

A. Property-Directed Reachability

The work presented here uses extensively the unbounded model checking algorithm of Property-Directed Reachability (PDR) [9]. Given an FSM $M = (S, I, T)$ and a safety property $P \subseteq \{0,1\}^{|S|}$ representing the set of safe states, PDR attempts to prove that P holds for M. It either returns an *inductive invariant* proving that no unsafe states are reachable or a *counter-example* showing that an unsafe state is reachable.

To achieve the above, PDR computes a sequence of predicates over the state elements $F = \langle F_0, ..., F_k \rangle$. The set of F_i-states over-approximates the set of states reachable in i or fewer clock cycles (*i.e.,* R_i). Each F_i is represented by a formula in Conjunctive Normal Form (CNF) and each clause in F_i over-approximates R_i.

The algorithm proceeds through a series of iterations $k = 1, 2, ...$ in which iteration k attempts to find a k-step counter-example. This process will either result in new clauses being added to some or all of the formulas for $F_0, ..., F_k$, or in a counter-example being found. If P indeed holds, the algorithm will eventually reach a point at iteration k where $F_i = F_{i-1}$ for some $i \leq k$. At this point, F_i is an inductive invariant proving the property P holds. The algorithm returns Reachable if an unsafe state is reachable under M and Unreachable otherwise.

B. Traditional SAT-based Debugging

The work presented here also utilizes the SAT-based automated debugging framework of [3]. Given an error trace exposing erroneous behavior in a circuit, the framework returns a set of locations where a fix can be implemented to correct the behavior. Letting $L = \{l_1, l_2, ..., l_{|L|}\}$ denote the suspect locations in the circuit, the transition relation is enhanced by the addition of a set of *error-select lines* $e = \{e_1, e_2, ..., e_{|L|}\}$. If $e_i = 0$ then the behavior at location l_i is unchanged. Setting $e_i = 1$ replaces l_i with a free variable w_i.

Subsequently, for a k cycle error trace, the enhanced transition relation is unrolled into a k frame ILA representation. Additional constraints are derived to set the primary input values to the values from the error trace in each time-frame and to force the primary output values to the reference values. An additional constraint is added to ensure that the circuit begins in a particular initial state. Finally, a cardinality constraint ϕ_n ensures that exactly n error-select lines are simultaneously active. The constrained ILA is converted into a propositional formula in CNF. The formula is such that each satisfying assignment indicates an n-tuple of suspect locations that can be simultaneously modified to correct the erroneous behavior in the circuit. As such, all satisfying assignments to the formula are found using an all-solutions SAT solver.

III. Approximation-Based Approach

This section presents an algorithm that localizes bugs that cause unreachable states. Given an erroneous circuit C, a set of suspect locations $L = \{l_1, ..., l_{|L|}\}$, and an unreachable target state condition $\mathcal{S}$ the proposed methodology finds suspect locations where a fix can be implemented to make at least one $\mathcal{S}$-state reachable. The set of suspect locations L is provided by the user. In the worst case it can include every location in the circuit. In practice, a larger suspect set is expected to increase the runtime of the algorithm. The engineer may therefore apply knowledge regarding the source of the error to reduce the size of L to *e.g.,* all locations within a block that is suspected to be the error source. The target state condition $\mathcal{S}$ is a predicate representing the set of target states, all of which are assumed to be unreachable in C. This is also a parameter provided by the user.

A *solution* is defined as an n-tuple of locations in the circuit that can be replaced by different Boolean functions to make any target state(s) reachable. The functions may be arbitrarily complex and may require the addition of new state elements, but do not require modifying any other locations in the circuit. In practice, we are usually interested in solutions with $n = 1$, *i.e.,* those that consist of only a single suspect location. The purpose of the proposed methodology is to find suspect locations that are indeed solutions to the problem. It returns $L_{sol} \subseteq L$, which is a set of solutions from L. Note that the methodology is intended only to indicate locations where a functional change makes a target state reachable. The engineer is responsible for deciding how to implement the fix. As required for traditional SAT-based debugging using a set of error traces [3], a full verification step is needed to confirm the correctness of the modified design.

The methodology of this section also requires two parameters that dictate the portion of this solution space it explores. The first parameter is the *window size* N which determines how many clock cycles are modeled explicitly by the debugging step. The second is the *cycle limit* K, which dictates the number of clock cycles modeled by the state space approximation. Their exact rationale and usage within the algorithm are explained later in this section.

The algorithm models and debugs a sequence of state transitions from an arbitrary reachable state to $\mathcal{S}$. As calculating the exact set of reachable states is an intractable problem, an over-approximation is used to model the potentially reachable states. Spurious solutions that may arise from the use of the approximation are detected and discarded. Detecting spurious solutions has the beneficial side effect of refining the reachable state space approximation, potentially reducing the number of spurious solutions found later.

In greater detail, the approach consists of three main steps: reachability analysis, debugging, and spurious solution detection. The reachability analysis step computes an over-approximation of the set of K-step reachable states (*i.e.,* R_K), where K is the cycle limit parameter described earlier. Towards this end, it executes PDR with $\mathcal{S}$ as its unsafe state set. In doing so, PDR computes the sequence $F = \langle F_0, F_1, ..., F_k \rangle$ where each F_i over-approximates the set of i-step reachable states and has the property that $F_i \cap \mathcal{S} = \emptyset$ (*i.e.,* no $\mathcal{S}$-states are in F_i). As such, F_K over-approximates R_K. PDR may terminate before computing F_K. In this case, $F_K = \neg\mathcal{S}$ is used. As $\mathcal{S}$ is unreachable, its negation clearly over-approximates the set of K-step reachable states for any value of K.

The debugging step solves a SAT-based debugging instance designed to find suspect locations that can be changed to allow for a sequence of state transitions from some state in F_K to a target state. The window size parameter N defined earlier determines the maximum length of the sequence of added transitions. As such, the debugging instance creates an ILA representation of N time-frames using the enhanced transition relation described in Section II-B. Intuitively, the ILA is constrained using F_K at its initial states and $\mathcal{S}$ as the next state of its final time-frame. The primary input and output variables are left unconstrained, allowing the solver to find solutions

Algorithm 1 APPROXIMATEUNREACHABILITY($C, \mathcal{S}, K, N$)

1: $L_{sol} = \emptyset$
2: $T =$ transition relation of C
3: $PDR(T, \mathcal{S}, K)$
4: $U =$ DEBUGGINGINSTANCE($C, \mathcal{S}, F_K$)
5: **while** $(Solution = SAT(U)) \neq UNSAT$ **do**
6: $t =$ current state of $Solution$
7: **if** PDR(T, t, K) == REACHABLE **then**
8: $L_{sol} = L_{sol} \cup \{Solution\}$
9: **else**
10: $U = U \wedge F_K$
11: **end if**
12: **end while**
13: **return** L_{sol}

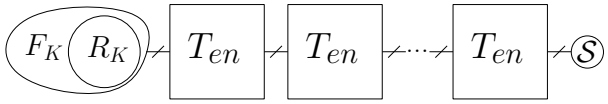

Fig. 1. Model used in the debugging step

for any input assignment. Finally, since exactly one error-select line must be active, a cardinality constraint [3] is applied to the error-select lines. The cardinality constraint ϕ_1 enforces that exactly one error-select line is 1, and is defined as follows:

$$\phi_1 = (e_1 \vee ... \vee e_{|L|}) \cdot \bigwedge_{\substack{1 \leq i < |L| \\ i \leq j \leq |L|}} (\bar{e_i} \vee \bar{e_j}) \quad (1)$$

In Eq. 1, the first clause ensures that at least one error-select line is active. The remaining clauses require that no pairs of error-select lines are active simultaneously. A similar cardinality constraint ϕ_n can be constructed for higher cardinalities. Such a constraint ensures that exactly n suspect locations are active. The resulting debugging instance is depicted in Figure 1 and can be expressed as follows:

$$F_K \wedge T_{en}^1 \wedge ... \wedge T_{en}^N \wedge \mathcal{S} \wedge \phi_n \quad (2)$$

Where T_{en}^i is the i^{th} copy of the enhanced transition relation in the ILA. A solution to Eq. 2 may indicate an n-tuple of locations where a change can be implemented to make a target state reachable. Due to the over-approximate nature of F_K, the debugging step may find spurious solutions if the chosen current state is a member of the set $F_K \setminus R_K$ (*i.e.,* it is not K-step reachable). Such a result may not indicate a design location where a change can be implemented to make a target state reachable. This necessitates a spurious solution detection step. When the SAT instance is solved, the solution is verified by using PDR to check if the current state is K-step reachable. If it is, the solution is proven to be non-spurious and added to the solution set. Conversely, if the current state is shown not to be K-step reachable, the solution is discarded. As a side effect of proving its unreachability, F_K is refined such that it does not include the unreachable current state. It may additionally exclude other states proven not to be K-step reachable. This is shown in Figure 2 where state t has been shown not be K-step reachable. As a result, a more accurate approximation shown in Figure 2(b) is derived.

The entire procedure is shown in Algorithm 2. Line 3 executes PDR to compute the initial approximation F_K. Line 4 constructs the debugging instance of Eq. 2. Subsequently, line 5 finds a solution to the debugging instance. On line 6, the current state of the solution is extracted and line 7 checks if it is K-step reachable. If so, line 8 adds it to the solution set. Otherwise the solution is found to be spurious and the updated F_K is conjuncted onto the debugging formula on line 10, preventing the algorithm from finding any further solutions with this current state.

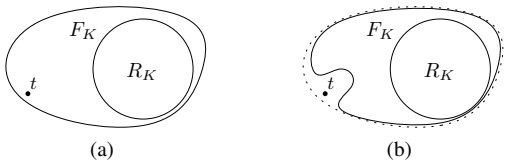

Fig. 2. Set F_K (a) initially (b) after proving state $t \notin R_K$

The following theorem proves the correctness of the algorithm.

Theorem 1 *The solution set of Algorithm 1 is exactly the set of all solutions that make a target state $\mathcal{S}$ reachable N cycles after a K-step reachable state.*

Proof: The debugging instance of Eq. 2 uses F_K as its current state set. Throughout the algorithm's execution, F_K is updated, but always includes every K-step reachable state. Therefore, the current state set of the debugging instance always includes all K-step reachable states, implying that the algorithm finds every solution that reaches the target state and has a K-step reachable current state. Furthermore, solutions where the current state is not K-step reachable are rejected, ensuring that all solutions found have a K-step reachable current state. This implies that it finds exactly the set of all solutions that allow for a sequence of N transitions from a K-step reachable state to a target state in $\mathcal{S}$. ■

Theorem 1 proves that the algorithm works when $\mathcal{S}$ can be reached with N transitions from a state that is K-step reachable. While this represents a useful subset of the solutions, in practice many other solutions may exist. The following section extends the algorithm to find the complete solution set.

IV. EXACT APPROACH

The approach of the previous section finds a particularly useful subset of the solutions to the problem of state unreachability. However, it has two key drawbacks. First, in order to use it the engineer must apply design knowledge to set parameters that dictate and limit the portion of the solution space it can explore. The second is that it may not provide confidence that it returns the actual error source, as it does not return every solution. This section presents a methodology to debug unreachable states that eliminates these drawbacks. Similar to the previous methodology it accepts an erroneous circuit C, a set of suspect locations $L = \{l_1, ..., l_{|L|}\}$, and an unreachable target state condition $\mathcal{S}$ as its input. It finds every solution in the set L and is complete by nature. As an additional feature, it terminates with a proof that no further solution(s) exist in L.

The proposed methodology essentially merges the debugging and reachability analysis steps of the previous section into a single process. As will be seen, this eliminates the possibility of finding spurious solutions. This is accomplished by solving a series of unbounded model checking problems using an enhanced FSM model of the circuit. The enhanced FSM model behaves like the original circuit with particular suspect locations replaced by unknown Boolean functions. Which suspect locations are replaced depends on assignments to *error-select registers*, which is new circuitry depicted in Figure 3 that behaves similarly to the error-select lines used in the debugging step of the previous section. Their exact rationale and functionality is explained later in this section. Each model checking problem either indicates a solution $l_i \in L$ or proves that no location in $L \setminus L_{sol}$ is a solution, at which point the algorithm terminates.

Towards this end, the algorithm constructs an enhanced transition relation T_{en} by adding new hardware in the form of error-select registers $E = \{e_1, ..., e_{|L|}\}$. It then constructs an enhanced FSM model of the circuit denoted by $M = (S \cup E, I_{en}, T_{en})$. A trace of the circuit $t_{C,0}, ..., t_{C,n}$ is *equivalent* to a trace of the FSM model $t_{M,0}, ..., t_{M,n}$ if and only if the registers in the set S (those registers

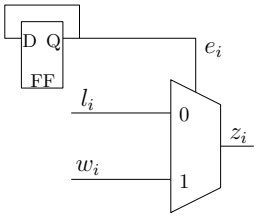

Fig. 3. Error select register and multiplexer at suspect location l_i

present in the original circuit) have the same assignment in states $t_{M,i}$ and $t_{C,i}$ for every $0 \leq i \leq n$.

In greater detail, the enhanced transition relation is constructed from that of the original circuit as follows. For each suspect location l_i, an associated free variable w_i and error-select register e_i are added. The hardware shown in Figure 3 is then constructed. Note that the error-select register is immutable (*i.e.,* its value can never change during the operation of the circuit) because its output is fed back to its input. As explained later, this associates the reachability of particular states under M with particular suspect locations being solutions. The hardware behaves such that l_i is effectively replaced by an arbitrary function when $e_i = 1$. Error-select registers assigned to 0 clearly do not alter the behavior of the circuit. This construction can also be represented efficiently in CNF as shown in Eq. 3 below:

$$mux = (e_i \vee \bar{l_i} \vee z_i)(e_i \vee l_i \vee \bar{z_i})(\bar{e_i} \vee \bar{w_i} \vee z_i)(\bar{e_i} \vee w_i \vee \bar{z_i}) \quad (3)$$

The enhanced transition relation is then constructed from the circuit with the added hardware. As the error multiplexer can be represented by four clauses, the CNF encoding of T_{en} has $O(|L|)$ more clauses than that of the original transition relation of C.

Example 1: To illustrate the behavior of T_{en}, consider the circuit of Figure 4(a). It has a single state element s_1, two primary inputs x_1 and x_2 and the two suspect locations are l_1 and l_2. Assume that the initial state is $s_1 = 0$ (*i.e.,* $I = (\bar{s_1})$). It is easily verified that it is impossible for the circuit to reach a state where $s_1 = 1$. This unreachability can be diagnosed using the target state condition $\mathcal{S} = (s_1)$. In doing so, the enhanced transition relation is constructed from the circuit shown in Figure 4(b). When $e_1 = e_2 = 0$, this circuit behaves the same as the original circuit. When $e_1 = 1$, l_1 is set to the free variable w_1, allowing it to assume any value during model checking. Similar behavior applies to e_2 and l_2. It can be seen that when any $e_i = 1$, this circuit behaves like the original circuit with l_i replaced by some unknown Boolean function.

As mentioned earlier, the reachability of certain states under the FSM is associated with particular suspect locations being solutions. Consider a trace of the enhanced model. The same error-select registers are active for every state in the trace, as these registers are immutable. Let $e_1, ..., e_n$ denote the active error-select registers. The enhanced model therefore behaves like the original circuit with locations $l_1, ..., l_n$ replaced by arbitrary Boolean functions. It can be concluded that the original circuit has an equivalent trace if $l_1, ..., l_n$ are simultaneously replaced by unknown functions. If this trace contains a target state, then simultaneously replacing $l_1, ..., l_n$ makes the target state reachable.

Now consider a trace that starts from an initial state, ends on a target state, and has exactly n active-error select registers $e_1, ..., e_n$. Using the argument in the previous paragraph, the original circuit has an equivalent trace when $l_1, ..., l_n$ are replaced with unknown functions. The trace starts from an initial state and ends at a target state, so replacing these n locations makes a target state reachable (*i.e.,* they are a solution). This applies to any trace satisfying these three properties, and we therefore tailor the algorithm to find such traces.

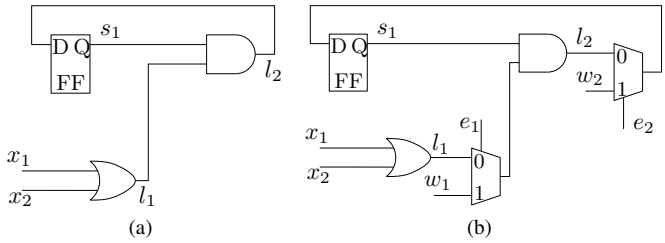

Fig. 4. (a) Original circuit (b) Circuit of T_{en} (error-select registers omitted)

This motivates the construction of the enhanced initial state constraint I_{en}. The formula for I_{en} constrains the original registers of the circuit using I, which ensures the initial states of the enhanced model correspond to initial states of the circuit. Since exactly n error-select registers must be active, the cardinality constraint ϕ_n is applied to the error-select registers. The enhanced initial state condition is therefore $I_{en} = I \wedge \phi_n$. This completes the construction of the enhanced FSM $M = (S \cup E, I_{en}, T_{en})$.

Example 2: Consider again the example from Figure 4. The enhanced initial state condition I_{en} is the conjunction of $I = (\bar{s_1})$ and the cardinality constraint ϕ_1. Therefore, $I_{en} = (\bar{s_1}) \wedge (e_1 \vee e_2) \wedge (\bar{e_1} \vee \bar{e_2})$. The set of states in I_{en} is $\{(\bar{s_1} \wedge e_1 \wedge \bar{e_2}), (\bar{s_1} \wedge \bar{e_1} \wedge e_2)\}$. Notice that these are all states in which $s_1 = 0$, which is the initial state condition. Additionally, every state of I_{en} has one active error-select register, matching the requirements for traces that indicate solutions.

It has been established that a trace of the enhanced model must have three properties to indicate a solution. Specifically, it must start on an initial state of C (an I-state), have exactly n error-select registers active in every state, and end on a target state. PDR is used to find such traces. The enhanced initial state condition ensures that it finds traces beginning on an I-state with exactly n active error-select registers, as required. The immutability of error-select registers in the enhanced transition relation ensures that the same n error-select registers are active error-select register in every state of the trace. The only remaining requirement is that the trace must end on a target state. To meet this requirement, PDR is executed with $\mathcal{S}$ as its unsafe state set. If any target state is reachable, PDR returns a counter-example trace that meets the requirements previously described. As such, if $e_1, ..., e_n$ are the active error-select registers in the counter-example then $l_1, ..., l_n$ is a solution.

Example 3: Continuing the illustration of the methodology from Example 2, recall that the target state condition is $\mathcal{S} = (s_1)$ and the initial state condition is $I = (\bar{s_1})$. The enhanced model has the following counter-example trace: $\langle t_0, t_1 \rangle = \langle (\bar{s_1} \wedge \bar{e_1} \wedge e_2), (s_1 \wedge \bar{e_1} \wedge e_2) \rangle$. Notice that t_0 corresponds to an initial state of the original circuit, t_1 is a target state, and e_2 is the active error-select register. In states t_0 and t_1 the model behaves identically to the original circuit with l_2 replaced by an unknown function. Since t_0 is an initial state and t_1 is a target state, replacing l_2 with a different function makes a target state reachable in the original circuit. This indicates that location l_2 is a solution. Indeed, the reader can confirm that replacing the AND-gate that drives l_2 with an OR-gate makes the target state reachable. Other corrections to the problem are also possible.

After finding a solution, it is blocked so the algorithm finds the remaining solutions if any. For a solution l_j, this is accomplished by conjoining the unit literal clause $\neg e_j$ to I_{en}, so that PDR will not return any additional counter-examples in which e_j is active. If a cardinality $n > 1$ is used, solutions will have n active error-select registers $e_1, ..., e_n$. In this case, a clause $(\neg e_1 \vee ... \vee \neg e_n)$ is conjuncted instead, ensuring that this same n-tuple is not found again as a solution.

Example 4: For the circuit of Figure 4, after finding the solution l_2, the enhanced initial state condition becomes $I_{en} = (\bar{s_1}) \wedge (e_1 \vee$

Algorithm 2 UNREACHABILITY($C, \mathcal{S}, L$)

1: $L_{sol} = \emptyset$
2: $S =$ state element set of C
3: $T_{en}, E =$ CONSTRUCTMODEL(L, C)
4: $I_{en} = I \wedge \phi_n$
5: $M = (S \cup E, I_{en}, T_{en})$
6: **while** $PDR(M, \mathcal{S}) ==$ REACHABLE **do**
7: $e_1, ..., e_n =$ active error-select registers in counter-example
8: $L_{sol} = L_{sol} \cup \{l_1, ..., L_n\}$
9: $M = (S \cup E, I_{en} \wedge (\neg e_1 \vee ... \vee \neg e_n), T_{en})$
10: **end while**
11: $invariant =$ inductive invariant extracted from PDR
12: **return** $(L_{sol}, invariant)$

$e_2) \wedge (\bar{e_1} \vee \bar{e_2}) \wedge (\bar{e_2})$, leaving $(\bar{s_1} \wedge e_1 \wedge \bar{e_2})$ as the only remaining initial state. It is easily verified that this state cannot reach any target states, implying that l_1 is not a solution. This is indeed the case. To reach a state where $s_1 = 1$ the output of the AND-gate must be 1. In the initial state $s_1 = 0$, so regardless of the value at l_1 the AND-gate will never output 1. Therefore, there is no way to modify the circuit at l_1 to rectify the unreachability of the target state.

The steps of the methodology are shown in Algorithm 2. In that description, algorithm CONSTRUCTMODEL receives input L and C and returns the enhanced transition relation and error-select register set. Lines 3 to 5 construct the enhanced FSM model that is used by PDR. Lines 6 to 10 contain the main loop in which solutions are found. If a solution exists, it is extracted (line 7) and added to L_{sol} (line 8). Line 9 constructs a new model in which the solution is blocked. As the number of suspect locations is finite, the loop will eventually terminate. At this point, PDR indicates $\mathcal{S}$ is unreachable and an inductive invariant is extracted (line 11). Finally, L_{sol} and the proof of solution completeness are returned in line 12.

In the following subsection, the soundness and completeness of this algorithm are discussed.

A. Soundness and Completeness

Two theorems are presented to demonstrate that Algorithm 2 is both sound and complete w.r.t. its input set. In the context of this paper, soundness implies that every location returned is a solution. Completeness implies that every solution from the set L is indeed found. Theorem 2 shows that the approach is sound. For simplicity, the theorems are presented in terms of an error cardinality of $n = 1$ but are similarly valid for other values of n.

Theorem 2 *Every location in L_{sol} is a solution.*

Proof: Line 6 finds a counter-example trace $t_0, ..., t_n$ of M. As it is a counter-example trace, it starts at an initial state and ends at a target state, implying $t_0 \in I_{en}$ and $t_n \in \mathcal{S}$. As $I_{en} = I \wedge \phi$, the cardinality constraint ϕ_1 ensures that exactly one error-select register (e_j) is assigned to 1 in state t_0. Additionally, $t_0 \in I$.

Since the error-select registers are immutable, each state in the trace also has e_j active and all other error-select registers inactive. Further, the fact that $t_0 \in I$ ensures that t_0 corresponds to an initial state of C. Therefore, an equivalent trace also exists for C if l_j is replaced by an unknown Boolean function. As t_n is a target state, $\mathcal{S}$ can be made reachable in C by replacing l_j, indicating that l_j is a solution. All locations in L_{sol} are found in this manner, implying that every location in L_{sol} is a solution. ■

Because L_{sol} is the solution set of Algorithm 2, this proves that the algorithm is sound. Further, Theorem 3 below proves the algorithm's completeness by showing that it returns all solutions from the set L.

Theorem 3 *Upon termination L_{sol} contains every solution from L*

Proof: Lines 6 to 10 are executed to find solutions until all target states are unreachable. First, consider the case when $L_{sol} = L$ at the termination of Algorithm 2. Clearly, this includes every solution in L, and the theorem holds in this case.

Assume the opposite case, Algorithm 2 terminates when all target states are unreachable and $L_{sol} \subset L$. It suffices to show that the unreachability of all target states implies that no solutions are left. Consider the final call to PDR that returns UNREACHABLE. Denote the remaining suspect locations at this point as $L_{rem} = L \setminus L_{sol}$.

Assume all target states are unreachable. This implies that there are no traces of M that end in a target state. Consider a fixed valid initial state $\mathcal{I}_C$ of C. There are $|L_{rem}|$ corresponding initial states of M, each with a different active error-select register. Since all target states are unreachable, none of these states can reach a target state under M. This implies that for every suspect location in $l \in L_{rem}$, it is impossible to replace l with a different Boolean function such that $\mathcal{S}$ is reachable from $\mathcal{I}_C$ in C. Since $\mathcal{I}_C$ is an arbitrary initial state of C, this holds for every initial state of C.

Therefore, none of the suspect locations in L_{rem} are solutions which implies that when Algorithm 2 terminates L_{sol} contains every solution from L. ■

As the algorithm only considers solutions from the suspect set L, it cannot find solutions that are not in that set. If every solution in the circuit is needed, the user may choose L so it includes every location in the circuit. Since a larger input set may increase runtime, the algorithm offers a favorable trade off where one can restrict the set L to locations that are considered to be likely error sources. For instance, if an engineer introduces a bug when modifying a specific module it may be desirable in this case to restrict the suspect set to said module and treat the rest of the design as correct. In this manner, one remains confident that the results returned by the algorithm will include the actual error source.

V. EXPERIMENTAL RESULTS

All results presented in this section are run on a single core of an i5-3570K 3.4 GHz workstation with 16GB of RAM. The proposed algorithms are implemented on top of the PDR implementation within ABC release 1.01 [11] and the state-of-the-art SAT-based debugging framework of [3]. Experiments are timed out after 4 hours. The benchmarks consist of seven designs from OpenCores [12] and one design from an industrial partner. Each problem instance is created by injecting a common design error that makes a state erroneously unreachable. Common design error include changed operators, complemented if-statement conditions, added incorrect state transitions, etc. The set L of suspect locations is chosen as all locations in the cone-of-influence of registers that appear in the target state predicate. If more than 1000 such locations exist, the 1000 closest to the registers appearing in the target state predicate are chosen. This ensures that the suspect locations give the greatest resolution to the user. All experiments are run using an error cardinality of $n = 1$.

Table I shows comprehensive results. The first four columns show the name of the problem instance, the number of gates in the design's AND-INVERTER graph representation, the number of registers in the design, and the size of the suspect set, respectively. The next two columns show the number of solutions found and runtime for the exact approach of Section IV. The next four columns relate to the approximate approach of Section III, with cycle limit $K = 50$ and window size $N = 1$. They show the number of solutions found, percentage of solutions found compared to the exact approach, run time, and speedup relative to the exact approach, respectively.

As can be seen, the approximate approach has 4.8x geometric mean speedup when compared to the exact approach, but finds only 43% of the complete solution set. This provides the user with a valuable set of tradeoffs. If the complete solution set is needed, the user can use the exact approach and have confidence that the results can be applied to accelerate debuggging. Conversely, if the user is able to

TABLE I
EXPERIMENTAL RESULTS

Benchmark				Exact Approach		Approximate Approach			
benchmark	#gates	#registers	$\|L\|$	#solutions	time (s)	#solutions	% solutions	time (s)	speedup
mrisc_core	8165	1328	1000	10	111	10	100	15.9	7.0x
design1	1070	147	314	14	21.7	4	29	16.0	1.4x
divider	3555	360	57	53	1.2	1	1.9	1.5	0.8x
spi	1009	132	544	40	76.6	40	100	19.0	4.0x
wb	390	61	346	261	211	247	95	38.7	5.4x
usb_core	4856	534	1000	4	492	4	100	17.5	28.1x
ac97_ctrl	12607	2325	126	18	16.8	10	56	1.4	12.1x
rsdecoder	4856	534	1000	40	951	40	100	371	2.6x
AVERAGE							43		4.8x

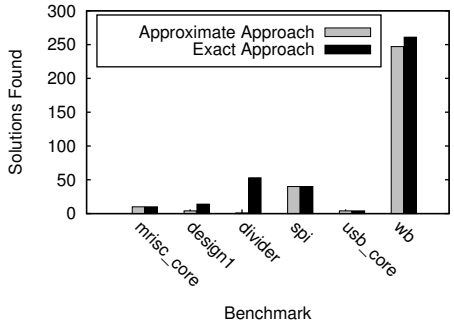

Fig. 5. Number of solutions found for each approach

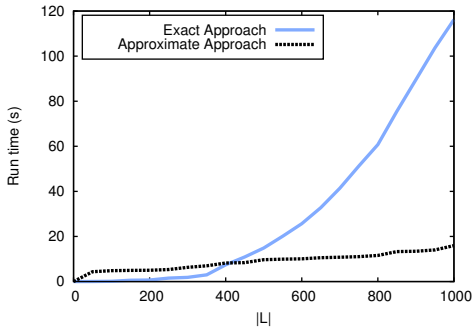

Fig. 6. Total and average runtime vs. $|L|$ for `mrisc_core`

appropriately set the parameter values for the exact approach it is possible to get a useful subset of solutions quickly.

Figure 5 plots the number of solutions found by the two approaches. Across all experiments, the approximate approach finds only of 43.3% of the complete solution set. In particular, for the design `divider`, only 2% of the solutions are found by the approximate approach. The experiments use a window size $N = 1$. Any solutions found therefore must be in the combinational fanin cone of a register that appears in the target state predicate, as the results of modifications at other locations cannot propagate to the relevant registers in one clock cycle. The `divider` benchmark is pipelined and the target state is specified on registers in the final stage. The experiment can therefore only find solutions in the combinational fanin cones of these registers, which represents a small part of the design. Since the problem can also be corrected in other pipeline stages, it is evident that it finds only a restricted portion of the complete solution set.

Furthermore, the exact approach provides an additional tradeoff when the user restricts the set of suspect locations L. Figure 6 plots the runtime of each approach versus $|L|$ for `mrisc_core`. It can be seen that increasing the size of the suspect set increases the total runtime for both algorithms as expected. However, increasing $|L|$ appears to have a much larger impact on the runtime of the exact approach. For the exact approach, increasing $|L|$ makes the problem more complex in multiple dimensions, as PDR must consider more states, the SAT instances it must solve become more complex, and the number of calls to PDR increases. However, the approximate approach spends much of its runtime computing the set of reachable states for the original circuit without error-select hardware. As a result, the size of the suspect set only substantially impacts the run time of the debugging step, making its runtime less dependent on $|L|$. By limiting the suspect locations to *e.g.,* the locations within a module expected to be the source of the error, the user can substantially improve the run time performance.

VI. CONCLUSION

This work presents two automated methodologies to debug design errors that manifest themselves in the form of unreachable states. The first uses steps of reachability analysis, SAT-based debugging, and spurious solution checking to compute a subset of the solutions. The second uses an enhanced FSM model of the circuit and PDR to compute the complete solution set at the cost of increased runtime. Experiments on industrial-level circuits confirm the practicality and effectiveness of both approaches.

REFERENCES

[1] H. Foster, "From volume to velocity: The transforming landscape in function verification," in *Design Verification Conference*, 2011.

[2] ——, "Assertion-based verification: Industry myths to realities (invited tutorial)," in *Intl Conference on Computer-Aided Verification (CAV)*, 2008, pp. 5–10.

[3] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, "Fault diagnosis and logic debugging using boolean satisfiability," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst*, vol. 24, no. 10, pp. 1606–1621, Oct. 2005.

[4] S.-Y. Huang and K.-T. Cheng, *Formal Equivalence Checking and Design DeBugging*. Norwell, MA, USA: Kluwer Academic Publishers, 1998.

[5] K.-H. Chang, I. Markov, and V. Bertacco, "Automating post-silicon debugging and repair," in *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, Nov 2007, pp. 91–98.

[6] G. Fey, S. Staber, R. Bloem, and R. Drechsler, "Automatic fault localization for property checking," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 27, no. 6, pp. 1138–1149, June 2008.

[7] R. Berryhill and A. Veneris, "Automated rectification methodologies to functional state-space unreachability," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15, 2015, pp. 1401–1406.

[8] A. Bradley, "Sat-based model checking without unrolling," in *Intl Conf. on Verification, Model Checking, and Abstract Interpretation*, 2011, pp. 70–87.

[9] N. Eén, A. Mishchenko, and R. Brayton, "Efficient implementation of property directed reachability," in *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, ser. FMCAD '11. Austin, TX: FMCAD Inc, 2011, pp. 125–134.

[10] H. Chockler, A. Ivrii, A. Matsliah, S. Moran, and Z. Nevo, "Incremental formal verification of hardware," in *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, ser. FMCAD '11. Austin, TX: FMCAD Inc, 2011, pp. 135–143.

[11] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification, Release 1.01." [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc/

[12] OpenCores.org, "http://www.opencores.org," 2007.