A Complete Approach to Unreachable State Diagnosability via Property Directed Reachability

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

***Abstract*— In modern hardware design, substantial manual effort is required to fix a design when verification discovers a state unreachable. This paper addresses this growing pain where given an unreachable target state, a methodology is presented to return all design locations where a change can be implemented to make the target state reachable. In contrast to previous state reachability rectification techniques that use bounded model checking, our approach addresses the issue using unbounded model checking. It first enhances the circuit transition relation by inserting a novel error model construction at each suspect location. An unbounded model checking algorithm is then applied to the enhanced transition relation to find which of the suspect locations can be changed to make the target state reachable. The use of unbounded model checking allows it to identify the complete problem solution set. As an added benefit, it also returns a proof that no further solution(s) exist in the form of an inductive invariant. Empirical results on industrial designs confirm the theoretical and practical gains of this approach.**

I. Introduction

In modern hardware design, functional verification has grown to consume up to 70% of the total design effort [1]. Debugging, the process of localizing the error, accounts for up to half of the verification cycle [2]. Many verification and debugging tasks have a great deal of automation available to mitigate the substantial engineering effort they require. However, in cases where a state is erroneously unreachable, comparatively little automation is available to aid the engineer in correcting the error.

Traditionally, functional verification may reveal errors through such means as firing assertions, observation signal value mismatches, or scoreboard discrepancies. In all such cases, an error trace that exposes the bug is returned. An automated debugging tool [3]–[6] later uses this error trace to aid the engineer in discovering the root cause of the error. Conversely, when a state is shown to be unreachable in violation of the design specification, no error trace is available for a traditional debugging utility to be applied. Towards that end, the work in [7] develops a technique to facilitate diagnosis of unreachable states. This technique, while effective, has some key drawbacks that represent barriers to a fully automated debug process for this problem. First, it may only provide a subset of the possible error sources. Additionally, it requires the engineer to set ad-hoc parameter values that essentially dictate and limit exploring the complete solution space. In simple terms, this is a trade off between resolution and performance where an engineer must apply knowledge of the design to intelligently choose these parameters. These drawbacks present a barrier to a fully automated debug process that explores the complete solution space in an effort to maximally preserve the engineering effort already invested in the design.

Towards this goal, this paper presents a novel methodology to debug unreachable states that explores the complete set of the solution space. Given an unreachable target state, the methodology returns every 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 formulating the debugging problem as an unbounded model checking problem, in contrast to traditional techniques [7] that use bounded model checking (BMC) [8].

In detail, the proposed methodology works as follows. First, the transition relation of the circuit is enhanced by inserting error-select registers. Each error-select register is associated with a suspect location such that if it is active, the suspect location is effectively disconnected from its fanout and replaced by a free variable. Any non-active error-select register does not alter the behavior of the circuit. This construction allows each suspect location to be replaced with an arbitrary Boolean function by activating the respective error-select register. In this manner, the target state is reachable under the enhanced transition relation if and only if there is a suspect location where a fix can be implemented to make it reachable in the original circuit (i.e. a *solution*). In order to find these suspect locations, property-directed reachability (PDR) [9]–[11] is executed to check the reachability of the target state under the enhanced transition relation. When reachable, PDR returns a counter-example in which an error-select register is active, indicating that the associated suspect location is a solution. The error-select register is then de-activated and PDR is executed again. This process is repeated until the target state is unreachable under the enhanced transition relation. At this point, no further solutions exist and PDR returns an inductive invariant that proves this claim.

Experiments on industrial designs confirm the theoretical findings and the practicality of the approach. The technique is able to find an average of 2.3x as many solutions as traditional approaches. An additional performance optimization is presented that provides a 5.1x speedup compared to the initial approach.

The remainder of this paper is organized as follows. Section II presents background information regarding unbounded model checking and prior work. Section III defines the problem and presents an algorithm to diagnose unreachable states. Section IV presents a methodology to apply model checking incrementally in the algorithm, potentially improving runtime performance. Section V presents experimental results. Finally, Section VI concludes the paper.

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 represents 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'. 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 in this paper 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.

A. Property-Directed Reachability

The work presented here uses extensively the unbounded model checking algorithm of Property-Directed Reachability (PDR) [10].

[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

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.,* the set of i-step reachable states). Each F_i is represented by a formula in Conjunctive Normal Form (CNF) and each clause in F_i over-approximates the set of i-step reachable states. As such, F_i and each of its clauses are referred to as *i-step invariants* [11]. Additionally, each clause c in every F_i includes every initial state, *i.e.*, $I \subseteq c$. In other words, c satisfies *initiation*.

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. Prior Work

Traditional automated debugging tools [3]–[6] require an error trace for each failure to constrain the problem. But when a state is unreachable due to some error, no such error trace exists and existing techniques are no longer applicable. To tackle this problem, the authors in [7] present an automated diagnosis methodology for unreachable states that relies on a set of intertwined steps of reachable state-space approximation and SAT-based debugging.

To operate, that algorithm requires three parameters. The first parameter is the the target state and the second is the maximum number of clock cycles K in which to reach this target state. Finally, it asks for a window size N, which represents the number of clock cycles that are debugged. Increasing K may find additional solutions that reach the target state in a greater number of clock cycles, while increasing N may find a greater portion of the solution set for a fixed value of K. When the algorithm begins, PDR is executed to over-approximate the set of $(K - N)$-step reachable states. Then, SAT-based debugging is applied to debug a sequence of N state transitions starting at any state in the approximation and ending at the target state. When a solution is found, PDR is used to determine if it is a real solution or a spurious one. Using larger values of N and K may result in finding larger numbers of solutions, but it may also increase runtime.

While this approach indeed diagnoses unreachable states, it requires the engineer to intelligently select the parameters in an ad-hoc manner so as to balance runtime with completeness of the solution. Essentially, this is a trade off between performance and resolution. As a result the method does not give the engineer confidence whether adjusting the values of parameters will provide more solutions or merely increase the runtime. Evidently, being able to explore the complete solution space presents more options for the engineer to correct the design and preserve the engineering effort invested in it. Ultimately these issues present barriers to a fully automated process for debugging unreachable states, which is the aim of the methodology presented in the next section.

III. DIAGNOSING UNREACHABLE STATES

Given an erroneous circuit C, a set of suspect locations denoted as $L = \{l_1, ..., l_{|L|}\}$, and an unreachable target state condition $\mathcal{S}$ the proposed methodology aids the engineer in finding suspect locations where a fix can be implemented to make at least one of the states of $\mathcal{S}$ (*i.e.,* an $\mathcal{S}$-state) reachable. The set of suspect locations L is provided

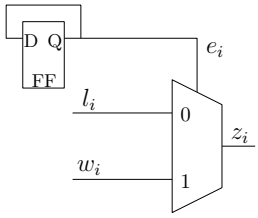

Fig. 1. Error select register and multiplexer at suspect location l_i

by the user and in the worst case it can include every location in the circuit. Of course, the larger the suspect set L is the more time the algorithm is expected to take. In practice, this is a favorable characteristic for the proposed algorithm because the engineer may 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. This is also a parameter provided by the user. All target states are unreachable in C.

A *solution* is defined as a location in the circuit that can be replaced by some Boolean function to make any target state(s) reachable. The function may be arbitrarily complex and may require the addition of new state elements, but does not require modifying any other locations in the circuit. The purpose of the proposed methodology is to determine which of the suspect locations are indeed solutions to the problem. It returns $L_{sol} \subseteq L$ which is the subset of locations in L that are solutions. The approach finds every solution in the set L and is complete by nature. As an additional feature, it terminates with a proof demonstrating that no further solution(s) exist in L. Note that the methodology is intended only to indicate locations where a functional change makes the target state reachable. The engineer is responsible for deciding how to implement the actual fix.

A. Methodology

The proposed methodology involves solving a series of unbounded model checking problem instances by generating an enhanced FSM model of the circuit. For simplicity of our presentation, this section treats the model checker as a "black box."

The enhanced FSM model behaves like the original circuit with certain suspect locations replaced by arbitrary Boolean functions. Which suspect locations are replaced depends on value assignments to *error-select registers*, which are new circuitry added to the original FSM and depicted in Figure 1. Their exact rationale and functionality is described later in this section. Each model checking problem is crafted so that the result 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 FSM $M = (S \cup E, I_{en}, T_{en})$, by adding new hardware in the form of error-select registers $E = \{e_1, ..., e_{|L|}\}$ and constructing an enhanced initial state condition I_{en} as well as transition relation T_{en}. A trace of the circuit $t_{C,0}, ..., t_{C,n}$ is *equivalent* to a trace of the model $t_{M,0}, ..., t_{M,n}$ if and only if the original registers in the set S have the same value assignment in states $t_{M,i}$ and $t_{C,i}$ for all $0 \leq i \leq n$.

The enhanced transition relation is constructed from that of the original circuit by adding hardware to facilitate diagnosis. For each suspect location l_i, an associated free variable w_i and error-select register e_i are added. The error-select register is made immutable (*i.e.,* its value can never change during the operation of the circuit) by feeding its output back to its input. As explained later, this is necessary to associate the reachability of particular states under M with a suspect location being a solution. Subsequently, new hardware is added such that l_i is effectively replaced by an arbitrary Boolean function when $e_i = \texttt{1}$. Error-select registers assigned to `0` have no effect on the behavior of the circuit. A multiplexer where the `0`-input is l_i, the `1`-input is the free variable w_i, and the select line

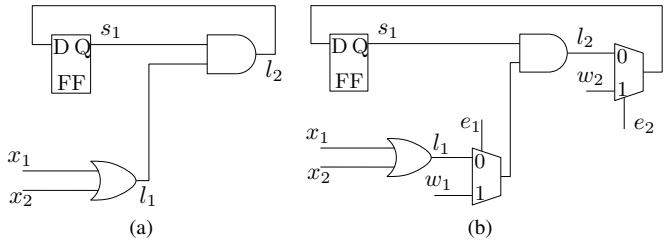

Fig. 2. (a) Original circuit (b) Circuit of T_{en} (error-select registers omitted)

is e_i implements this functionality. The output of the multiplexer is denoted z_i and connected to the original fanout of l_i. This multiplexer construction is depicted in Figure 1 while the equation below shows its CNF representation:

$$mux = (e_i \vee \bar{l_i} \vee z_i)(e_i \vee l_i \vee \bar{z_i})(\bar{e_i} \vee \bar{w_i} \vee z_i)(\bar{e_i} \vee w_i \vee \bar{z_i}) \quad (1)$$

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 2(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 2(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.

To diagnose unreachable states, it is necessary to associate the reachability of particular states with a specific location being a solution. Consider a trace of the enhanced model. All states in the trace have the same active error-select registers, as they are immutable. Assume $e_1, ..., e_n$ are active. 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.

For the trace to indicate a solution it must satisfy additional properties. Specifically, it must start from an initial state, end on a target state, and have exactly one active-error select register e_j. Using the argument in the previous paragraph, replacing l_j with an unknown function makes an equivalent trace exist for the original circuit. Since the trace starts from initial state and ends at target state, replacing l_j makes a target state reachable. This implies l_j is a solution. It is therefore possible to find solutions by finding traces with these three properties.

This motivates the construction of an enhanced set of initial states denoted as I_{en}. The original registers of the circuit are constrained using I, ensuring the initial states of the enhanced model correspond to initial states of the circuit. Since exactly one error-select register must be active, a cardinality constraint [3] is applied to the error-select registers. The cardinality constraint ϕ enforces that exactly one error-select register is assigned to 1, and is defined as follows:

$$\phi = (e_1 \vee ... \vee e_{|L|}) \cdot \bigwedge_{\substack{1 \le i < |L| \\ i \le j \le |L|}} (\bar{e_i} \vee \bar{e_j}) \quad (2)$$

In Eq. 2, the first clause ensures that at least one error-select register is assigned to 1. The remaining clauses ensure that no two error-select registers are simultaneously 1. The enhanced initial state condition is $I_{en} = I \wedge \phi$. This completes the construction of the enhanced FSM $M = (S \cup E, I_{en}, T_{en})$.

Example 2: Consider again the example from Figure 2. The enhanced initial state condition I_{en} is the conjunction of $I = (\bar{s_1})$ and the cardinality constraint ϕ. 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 certain traces of the enhanced model indicate that specific locations are solutions. PDR is used to find traces that meet these requirements. The enhanced initial state condition ensures that the traces PDR finds begin on an initial state with exactly one active error-select register, as required. The enhanced transition relation ensures that the same single error-select register is active in every state of the trace. All that remains is to force the trace to end on a target state. To do so, PDR is executed with $\mathcal{S}$ as its unsafe state set. If any target state is reachable, it will return REACHABLE and a counter-example trace that meets the requirements previously described. As such, if e_j is the active error-select line in the counter-example then l_j 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.

Example 4: For the circuit of Figure 2, after finding the solution l_2, the enhanced initial state condition becomes $I_{en} = (\bar{s_1}) \wedge (e_1 \vee e_2) \wedge (\bar{e_1} \vee \bar{e_2}) \wedge (\bar{e_2})$, 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 1. 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 11 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 M' in which solution l_j is blocked. The distinction between M and M' in the algorithm is included to simplify the discussion in Section IV. 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 12). Finally, L_{sol} and the proof of solution completeness are returned in line 13.

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

Algorithm 1 UNREACHABILITY($C, \mathcal{S}, L$)

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

B. Soundness and Completeness

Two theorems are presented to demonstrate that Algorithm 1 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 to the problem. Completeness implies that every solution from the set L is indeed found. Theorem 1 below uses the nature of traces of M to show that the approach is sound.

Theorem 1 *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 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 1, Theorem 1 proves that the algorithm is sound. Further, Theorem 2 below shows that the methodology is also complete, that is, it returns all solutions from the input set of suspect locations L.

Theorem 2 *Upon termination L_{sol} contains every solution from L*

Proof: Lines 6 to 11 are executed to find solutions until all target states are unreachable. First, consider the case when $L_{sol} = L$ at the termination of Algorithm 1. Clearly, this includes every solution in L, and the theorem holds in this case.

Assume the opposite case, Algorithm 1 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.

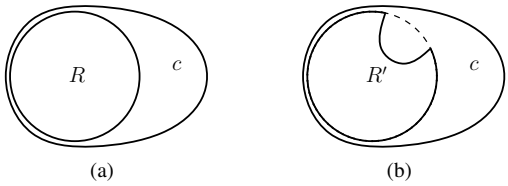

Fig. 3. State space representation of (a) M and (b) M'

Therefore, none of the suspect locations in L_{rem} are solutions which implies that when Algorithm 1 terminates L_{sol} contains every solution from L. ■

As the algorithm examines solutions strictly from the suspect set L, it obviously cannot find solutions that are not in that set. If every solution in the circuit is needed, the user may choose L to include every circuit location. As a larger input set may increase runtime, the algorithm offers a favorable trade off where one can limit the initial suspect set L to locations that are considered to be likely error sources. For instance, an engineer may introduce 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 indeed include the actual error source.

IV. PERFORMANCE OPTIMIZATION

In the methodology of the previous section, each solution results in an additional call to PDR. In the worst case, this will require a total of $|L|$ calls. We note that the only difference between consecutive PDR calls is that a single error-select register is set to 0. For such a small change in the model, many of the invariants may remain valid [11]. In fact, as shown later in this section, all of the invariants generated by PDR remain valid in the problem of interest. This allows each call to PDR to reuse the entire sequence of formulas $F = \langle F_0, ..., F_k \rangle$ which can potentially result in substantial runtime savings.

As explained in Section II-A, PDR computes a sequence of predicates $F = \langle F_0, ..., F_k \rangle$ represented by CNF formulas, where each F_i and each clause of F_i are i-step invariants. Each clause c additionally has the property of initiation, *i.e.,* $I_{en} \subseteq c$. In order to reuse clauses and formulas after changing the model, they must remain i-step invariant and satisfy initiation with respect to the new model. The work of [11] presents an invariant finder, which extracts the portion of the invariants computed for one model that remain valid in another. This provides a means for the reuse of clauses after changing the model in the general case. However, it is possible to exploit the structure of the model updates in Algorithm 1 to reuse every clause without verifying i-step invariance or initiation.

To further elaborate this point, consider the state of Algorithm 1 immediately after executing line 9. At this point, M is the FSM model used to find solution l_j and M' is the FSM model after blocking solution l_j. It is immediately obvious that $I'_{en} \subseteq I_{en}$. For any clause c, if $I_{en} \subseteq c$, it is trivially true that $I'_{en} \subseteq c$. Therefore, all clauses that satisfy initiation under M satisfy initiation under M'.

This leaves only to show that all i-step invariant formulas and clauses of M are i-step invariant for M'. If every clause in F_i is i-step invariant then F_i itself is also i-step invariant [9]. Therefore, it is sufficient to show that the i-step invariant clauses of M are i-step invariant for M'. This ultimately arises from the fact that the reachable state set of M' is a subset of that of M. Any over-approximation of the set of states reachable under M will also over-approximate the reachable states of M'. An i-step invariant is simply an over-approximation of the set of i-reachable states, so intuitively, the i-step invariant clauses of M are also i-step invariant clauses of M'. Lemma 1 provides a first step towards proving this claim, by showing that conjoining the clause $\neg e_j$ to I_{en} does not make any previously-unreachable $\neg e_j$-states reachable.

TABLE I
EXPERIMENTAL RESULTS

Benchmark				Traditional [7]		Initial		Optimized		
benchmark	#gates	#registers	$\|L\|$	#solutions	time (s)	#solutions	time (s)	#solutions	time (s)	speedup
mrisc_core	8165	1328	1000	10	15.9	10	430	10	111	3.9x
design1	1070	147	314	4	16.0	14	128	14	21.7	5.9x
divider	3555	360	57	1	1.5	53	2.2	53	1.2	1.8x
spi	1009	132	544	40	19.0	40	598	40	76.6	7.8x
wb	390	61	346	247	38.7	261	9983	261	211	47.3x
usb_core	4856	534	1000	4	17.5	4	1065	4	492	2.2x
ac97_ctrl	12607	2325	126	10	1.4	18	44.8	18	16.8	2.7x
rsdecoder	4856	534	1000	40	371	-	-	40	951	-
GEOMEAN										5.1x

Lemma 1 *All $\neg e_j$-states that are not i-step reachable under M are not i-step reachable under M' for all $i \geq 0$.*

Proof: Consider a $\neg e_j$-state t that is not i-step reachable under M. Assume towards a contradiction that it is i-step reachable under M'. M' must have at least one trace $t_0, ..., t_n$ where $t_0 \in I'_{en}$ and $t_n = t$. As error-select registers are immutable and t is a $\neg e_j$-state, t_0 is also a $\neg e_j$-state.

As M and M' have the same transition relation, each state transition in the trace is valid under M. Therefore, t is only unreachable under M if $t_0 \notin I_{en}$. This is a contradiction as $t_0 \in I'_{en}$ and it has already been shown that $I'_{en} \subseteq I_{en}$. Therefore, all $\neg e_j$-states that are not i-step reachable under M are not i-step reachable under M'. ■

As shown, the model updates in Algorithm 1 do not make any unreachable $\neg e_j$-states reachable. Further, they clearly make all e_j-states unreachable. These two facts together imply that no states unreachable under M' are reachable under M. Letting R (R') denote the set of states reachable under M (M'), we have $R \subseteq R'$. It only remains to show that this implies all i-step invariants of M are i-step invariant for M'.

To visualize this fact, assume c is an invariant clause under M. As an invariant clause of M, it must over-approximate R. This is shown in Figure 3(a), where the set of c-states contains R. As R' is a subset of R, Figure 3(b) shows that the set of c-states also over-approximates R'. Clause c therefore remains invariant for M'.

The above discussion focuses on invariant clauses, but it is similarly valid for i-step invariant clauses. The following theorem proves this claim.

Theorem 3 *All clauses that are i-step invariant under M are i-step invariant under M'.*

Proof: Let c be a clause that is i-step invariant under M. Assume towards a contradiction that c is not i-step invariant under M'. This implies that there is a state $t \notin c$ that is i-step reachable under M'. Additionally, t is not i-step reachable under M, as c is i-step invariant under M and $t \notin c$.

Since t is i-step reachable under M' and not M, by Lemma 1, it is an e_j-state. No e_j states are reachable under M', contradicting the assumption that c is not i-step invariant under M'. ■

It has already been shown that all clauses satisfying initiation for M do so as well for M', so Theorem 3 implies that it is possible to reuse $F = \langle F_0, ..., F_k \rangle$. That is, the execution of PDR on line 6 of Algorithm 1 can start with F already initialized to the sequence previously computed. This can potentially result in substantial performance optimization to the execution of Algorithm 1.

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 algorithm is developed on top of the PDR implementation of ABC release 1.01 [12]. For comparison the approach of [7] is implemented using a state-of-the art SAT-based debugger [3] and the same PDR implementation. Experiments are timed out after 14400 seconds (4 hours). Seven designs from OpenCores [13] and one commercial design from an industrial partner are utilized as benchmarks. Each problem instance is created by injecting a design error to make at least one state erroneously unreachable. This is accomplished by changing operators in expressions, complementing conditions in if-statements, introducing incorrect state transitions, etc. These are typical design errors observed in industry.

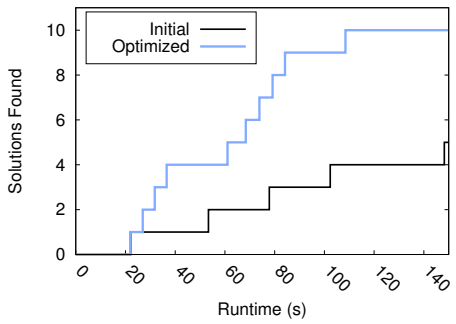

Fig. 4. Solutions vs. runtime for `mrisc_core`

Suspect locations for the set L are chosen algorithmically as follows. The fanin of all registers that appear in the target state predicate is traversed breadth-first and each location visited is made a suspect. This process continues until all locations in the cone of influence of the relevant registers are added or an upper limit of 1000 suspects is reached. All suspects chosen by this algorithm are in the cone-of-influence of a relevant register, and therefore may be able to rectify the error. Further, locations with greater spatial locality to the problem are visited first. This ensures that the suspects are those most likely to be relevant to the problem.

Table I shows comprehensive results. The first four columns show the name of the problem instance, the number of gates in the design, 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 when using the the approach of [7] with window size $N = 1$ and cycle limit $K = 50$. The next two columns show the runtime and number of solutions found using the unoptimized approach in which clauses are not reused. The final three columns show the runtime, number of solutions found, and speedup relative to the initial approach for the optimized approach.

The optimized and unoptimized approaches always find the same solutions. Over the entire set of experiments the optimized approach achieves a 5.1x geometric mean speedup relative to the unoptimized approach. Notice that the speedup is 47x for the circuit `wb`. This happens because of the large number of solutions. To demonstrate this effect, Figure 4 shows the number of solutions found versus runtime for the optimized and unoptimized approaches for `mrisc_core`. It can be seen that both approaches take roughly the same amount of time to find one solution. After finding the first solution, the optimized

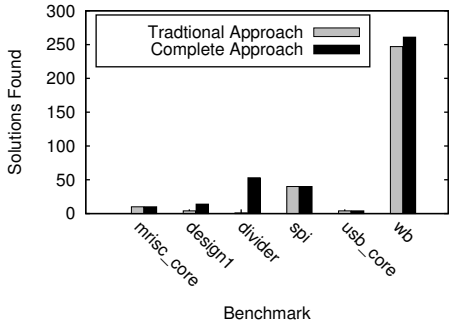

Fig. 5. Number of solutions found for each approach

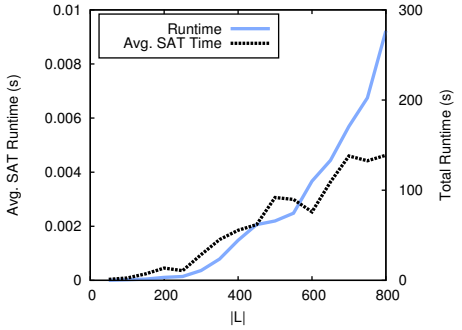

Fig. 6. Total and average SAT runtimes vs. $|L|$ for `usb_core`

approach is evidently able to find subsequent solutions more quickly. The unoptimized approach appears to take roughly the same amount of time to find each solution. Problems with large numbers of solutions are therefore expected to exhibit a larger speedup from this optimization. As can be seen, our approach requires somewhat more runtime than the traditional approach. This is expected yet it represents a worthwhile tradeoff, as its completeness offers the engineer greater insight for the root cause of the problem and flexibility for its rectification.

Figure 5 plots the number of solutions found by the approach presented in this work and the traditional approach of [7]. Across all experiments, the traditional approach is able to find only of 43.3% of the complete solution set. It can be seen that the design `divider` in particular has a large number of solutions not found by the traditional approach. Since the experiments use a window size of one, only one state transition is debugged. Any solutions found therefore have to be in the combinational fanin cone of a register that appears in the target state predicate, as the results of modifications at other locations would not propagate to the relevant registers in one clock cycle. The divider benchmark is a pipelined divider circuit and the target state is specified as a condition on registers in the final pipeline stage. The traditional approach can only find solutions in the combinational fanin cones of these registers, most of which is in the final pipeline stage. Since the problem can be corrected in other stages, it is evident that it finds only a restricted portion of the complete solution set.

Table II illustrates the effect of the suspect set L on the runtime of the algorithm for `usb_core` and `mrisc_core`. The first two columns show the benchmark and size of the suspect set. The next two columns show the number of SAT calls made by PDR and the overall runtime. It can be seen that larger suspect sets result in more SAT calls and longer runtime. Introducing more suspects increases the size and complexity of the enhanced transition relation, naturally making the SAT instances more expensive to solve. Each suspect location can also make additional states reachable. This can make approximating the state space more difficult, naturally increasing the number of times PDR must call the SAT solver.

TABLE II
EFFECT OF $|L|$ ON RUNTIME

benchmark	$\|L\|$	#SAT calls	SAT run-time (s)	total run-time (s)
mrisc_core	100	326	0.01	0.11
mrisc_core	250	1344	0.21	1.33
mrisc_core	500	2029	1.31	13.62
mrisc_core	1000	4029	13.3	111.1
usb_core	100	3371	0.30	0.65
usb_core	250	4438	1.59	4.24
usb_core	500	10818	33.2	65.9
usb_core	1000	22074	156.4	491.5

Figure 6 plots the total runtime and average runtime per SAT call for `usb_core`. It can be seen that increasing the size of the suspect set increases the total runtime and the average runtime of each SAT instance, as expected. Evidently, adding more suspect locations increases the number and complexity of the SAT instances solved. It seems as though the increase in runtime is not linear in the increase in $|L|$. 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. The total runtime of the algorithm therefore appears to be heavily-dependent on the suspect set it is given.

VI. CONCLUSION

This work presents an algorithm to diagnose design errors that result in unreachable states that returns the complete solution set of the problem. This is done by constructing an enhanced FSM model of the circuit and solving unbounded model checking problems on the enhanced FSM model. An optimization is presented to improve the performance. Experimental results confirm the practicality and effectiveness of the presented approach.

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. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," in *Advances in Computers*, vol. 58, 2003, pp. 118–149.
[9] A. Bradley, "Sat-based model checking without unrolling," in *Intl Conf. on Verification, Model Checking, and Abstract Interpretation*, 2011, pp. 70–87.
[10] 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.
[11] 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.
[12] 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/
[13] OpenCores.org, "http://www.opencores.org," 2007.