Property Directed Reachability with Word-Level Abstraction

Yen-Sheng Ho, Alan Mishchenko, Robert Brayton
University of California, Berkeley
{ysho, alanmi, brayton}@eecs.berkeley.edu

Abstract—SAT-based Property Directed Reachability (PDR) has become the key algorithmic development for unbounded model checking of gate-level sequential circuits, but it can be inefficient when applied to word-level problems with heavy arithmetic logic. To address this issue, word-level abstraction is often performed by replacing a whole set of signals with unconstrained new primary inputs. This paper introduces PDR-WLA, a word-level abstraction-refinement algorithm integrated into a modified PDR implementation. The algorithm uses efficient refinement and re-uses reachability information across iterations of refinement. PDR-WLA was implemented in ABC and evaluated on a large set of industrial Verilog designs. Experimental results show significant speedups on hard problems compared to the original PDR and to a naive word-level abstraction-refinement method.

I. INTRODUCTION

Unbounded model checking (UMC) on a Register-Transfer-Level (RTL) circuit is hard but has important applications in the IC design industry:

1) Sequential equivalence checking (SEC). An RTL circuit is sequentially synthesized by retiming, clock-gating, pipelining etc., and UMC is required for proving the correctness of the synthesis.

2) Property checking. UMC is used to prove that a circuit always satisfies a set of given properties.

UMC is challenging at the bit level, and even more so at the word level, where complex arithmetic operators, such as multipliers, adders, and variable shifters, are involved.

IC3 [3] or Property Directed Reachability (PDR) [8] is considered the best algorithm for bit-level UMC. Abstraction has been a key development and is widely used. Different methods of abstraction include the following. Word-level abstraction [12], [1], [5], [4], [14], [11] can be effective by abstracting away heavy arithmetic logic. Localization abstraction [19] is a method where gates or signals are replaced by new unconstrained primary inputs. Counterexample guided abstraction and refinement (CEGAR) [7] is a framework for iterating abstraction and refinement, where refinement is based on the analysis of spurious counterexamples.

We propose PDR-WLA, an efficient CEGAR-based word-level localization algorithm integrated with PDR. Given a word-level design, PDR-WLA starts with the extreme abstraction with all hard signals (e.g., outputs of multipliers, adders, etc.) abstracted (i.e. replaced by new primary inputs). Next, the resulting word-level abstraction is bit-blasted and given to a modified PDR algorithm. If a counterexample (CEX) is found, PDR-WLA simulates it on the original design to check if it is real. If so, PDR-WLA reports it and terminates; otherwise, the CEX is spurious and is used to refine the current abstraction. Then a new iteration begins with the refined abstraction.

The main contributions embodied in PDR-WLA are that it

- integrates word-level abstraction with PDR efficiently,
- uses a new refinement strategy that takes advantage of structural and proof-based analysis of spurious counterexamples, and
- re-uses reachability information (reachability clauses) derived in previous iterations.

PDR-WLA is implemented and available in the public verification tool ABC [6] (command %pdra). It was evaluated on a set of 195 industrial Verilog RTL benchmarks. PDR-WLA is capable of solving 18 hard problems not solved by PDR. The results also show that 1) reusing previously derived reachability clauses improves performance significantly and 2) the new refinement strategy is the most effective compared to several others proposed and tested.

This paper starts with background material in Section II. PDR-WLA is presented in Section III. Various refinement strategies are given in Section IV. Related work is discussed in Section V. Experiments are presented in Section VI. Conclusions and future work are discussed in Section VII.

II. PRELIMINARIES

A. The UMC problem

The input is a word-level circuit given in structural Verilog containing bit-vector (BV) signals, including primary inputs (PIs), primary outputs (POs), flip flops (FFs), and internal signals. Flip flops have reset values as initial states. A design is modeled as a finite state machine (FSM).

Definition 1. An FSM is a tuple \( M = (I, O, S, \text{Init}, T) \) where \( I \) is the set of PIs, \( O \) is the set of POs, \( S \) is the set of FFs, \( \text{Init} \) is the set of initial states, and \( T \) is the set of (deterministic) transition relations where \( T \subseteq I \times S \times S \). If \( (i, s, s') \in T \), there exists a transition from \( s \) to \( s' \) under \( i \).

The input word-level circuit is assumed to contain a single FSM and a single output, \( out \), representing a property to be checked. If the problem is to prove equivalence between two designs, it is assumed that a miter circuit, \( M \), has been created by merging all PIs and merging FFs if their correspondences are known. The miter’s output, \( out \), is a Boolean signal, which

\footnote{Reset values are either constants or free variables (unknown value \( X \)).}
is the OR of the pairwise XORs of the corresponding outputs of the two designs. Thus \( \text{out} = 1 \) if the two designs are different. Similarly for property checking, \( \text{out} \) is the output of a monitor, and \( \text{out} = 1 \) if the property fails. In terms of linear temporal logic (LTL), the UMC problem is formulated as \( M \models \mathbf{G} \neg \text{out} \), i.e., \( \text{out} \) is never 1 if the property holds.

A UMC solver either reports a counterexample (CEX) that falsifies the property or produces an inductive invariant proving that the property holds globally.

**Definition 2.** A counterexample (CEX) is a sequence of PI assignments driving the design from an initial state into a state falsifying the property.

**Definition 3.** An inductive invariant (Inv) proving a property \( P(s) \) is a predicate function satisfying the properties below.

1. \( \text{Init}(s) \Rightarrow \text{Inv}(s) \)
2. \( \text{Inv}(s) \land T(i, s, s') \Rightarrow \text{Inv}(s') \)
3. \( \text{Inv}(s) \Rightarrow P(s) \)

**B. Property Directed Reachability**

It is assumed that the reader is familiar with the basic ideas underlying the PDR [8]. Algorithm 1 outlines a high-level view of PDR. It maintains a list of sets of clauses, called the PDR trace: \( \Omega = (R_0, R_1, \ldots, R_N) \). Every \( R_j \) is a set of clauses that over-approximates the set of states reachable from the initial states within \( j \) steps. These clauses in a PDR trace are called reachability clauses.

**Definition 4.** Given an FSM, \( M = (I, O, S, \text{Init}, T) \), and a property \( P \), a PDR trace is a sequence of predicate functions, \( \Omega = (R_0, R_1, \ldots, R_N) \), such that

1. \( R_0(s) = \text{Init}(s) \)
2. \( R_j(s) \Rightarrow R_{j+1}(s) \) for \( 0 \leq j < N \).
3. \( R_j(s) \land T(i, s, s') \Rightarrow R_{j+1}(s') \) for \( 0 \leq j < N \).
4. \( R_j(s) \Rightarrow P(s) \) for \( 0 \leq j < N \).

**Algorithm 1 PDR**

Input: \( G_M \Rightarrow G_M^\prime \) the bit-level input circuit
Output: status \( \in \{ \text{SAT}, \text{UNSAT} \} \)

1. \( \Omega \leftarrow \{ \text{Init} \} \Rightarrow \Omega \) the PDR trace
2. \( k \leftarrow 0 \Rightarrow k \) the PDR depth
3. while true do
4. \( \Omega, \text{cex} \leftarrow \text{RECBLOCKCUBE}(G_M, \Omega, k) \)
5. if \( \text{cex} \neq 0 \) then
6. return \( \text{SAT} \) \Rightarrow \text{Found a real CEX}
7. \( k \leftarrow k + 1 \)
8. \( \Omega \leftarrow \Omega \cup \{ \top \} \Rightarrow \text{Open a new frame}
9. \( \Omega \leftarrow \text{PROPAGATEBLOCKEDCUBES}(G_M, \Omega) \)
10. if \( \Omega \) contains a fixed point then
11. return \( \text{UNSAT} \)

PDR starts with the trace \( \Omega \) with only one element \( R_0 = \text{Init} \). It then tries to strengthen the trace by recursively blocking bad cubes\(^3\) (line 4). If a bad cube intersects with the initial states, then a CEX is returned. Otherwise, the last element \( R_k \) of the trace now satisfies the property \( P \). PDR then adds a new element \( \top \) (empty set of clauses) to \( \Omega \), and tries to propagate clauses (using induction) from \( R_1 \) to \( R_k \) (line 9). If a fixed point \( R_j \) is found, the problem is declared UNSAT and the inductive invariant \( R_j \) is returned.

The details of procedures \( \text{RECBLOCKCUBE} \) and \( \text{PROPAGATEBLOCKEDCUBES} \) can be found in [8].

**C. Word-level abstraction**

In this paper, localization abstraction [19] is used. Given a word-level circuit and a set of target signals (e.g., outputs of arithmetic operators), an abstraction is created by replacing the target signals with free variables called pseudo PIs (PPIs). Localization is not necessarily restricted to flip flops; any signal can be abstracted, similar to GLA [16].

**Example 1.** Consider the circuits in Figure 1. The PO, \( \text{out} \), in Figure 1a is constant-0, since both \( 2 \times x \equiv x + x \) and \( 2 \times y \equiv y + y \) are true. Figure 1b is the result of abstracting all 4 arithmetic operators by replacing their outputs with PPIs. Note that while the example is combinational for illustration purposes, the abstraction scheme applies generally to sequential circuits and UMC problems.

**Definition 5.** Given an original circuit \( M \) and an abstraction \( A \) of \( M \), a CEX of \( A \) is real if it can falsify the property on \( M \) (make \( \text{out} = 1 \)). Otherwise, it is spurious.

\(^3\)A cube of states containing one where the property fails (a bad state).
D. Simple CEGAR (S-CEGAR)

Algorithm 2 (S-CEGAR) is an example of a simple integration of CEGAR and PDR at the word level. The algorithm starts by abstracting all signals in the set $S$ (e.g., outputs of all specified arithmetic operators). Next, an abstraction-refinement loop is entered where each iteration begins by creating a word-level abstraction based on the current set $B$, the set of signals to be abstracted away. The abstraction is then bit-blasted and solved by a bit-level PDR. If the solver returns UNSAT, the property is proved. Otherwise a CEX to the abstraction, cex, exists and is then simulated on the original circuit ($W_M$) to check if it is real. If yes, the property is falsified and cex is returned; otherwise cex is analyzed to derive a set of signals ($\Delta B$) that, if un-abstracted, can block cex. A new abstraction, with $\Delta B$ un-abstracted, is then created and a new iteration begins.

Algorithm 2 Simple CEGAR (S-CEGAR)

Input: $W_M$ ⊳ $W_M$: the word-level input circuit
Input: $S$ ⊳ $S$: the initial set of targeted signals
Output: status ∈ { SAT, UNSAT }
1: $\text{Iterations} \leftarrow 1$
2: $B \leftarrow S$ ⊳ $B$: the set of abstracted signals
3: while true do
4:     $W_A \leftarrow \text{CREATEABSTRACTION}(W_M, B)$
5:     $G_A \leftarrow \text{BitBLAST}(W_A)$
6:     cex ← PDR($G_A$)
7:     if cex ≠ \emptyset then
8:         if IsREALCEX($W_M$, cex) then
9:             return SAT
10:         else
11:             $\Delta B \leftarrow \text{REFINE}(W_M, G_A, B, cex)$
12:             $B \leftarrow B \setminus \Delta B$
13:         end
14:     else
15:         return UNSAT
16: end

In each iteration of S-CEGAR, a new PDR solver is used and reachability clauses are recomputed from scratch. This is inefficient when the algorithm needs many iterations to find a final abstraction, i.e. one that proves the property.

III. PDR with Word-Level Abstraction

A. The algorithm

PDR-WLA uses an important insight; PDR traces can be re-used between iterations if abstractions are monotone. The idea is similar to previous work of PDR with abstraction [18], [10], extending it to the word level.

Similar to PDR, PDR-WLA starts with the trace $\Omega$ containing only $R_0 = Init$. One difference is that PDR-WLA works on an abstraction instead of the original circuit. Similar to S-CEGAR, it begins by abstracting all targeted signals $S$, resulting in a word-level abstraction ($W_A$), which is then bit-blasted into a circuit ($G_A$). As with PDR, PDR-WLA tries to recursively block bad cubes at depth $k$ with the abstract model $G_A$ and the trace $\Omega$. If a bad cube intersects with the initial states, then a CEX, cex, is returned and checked on the original circuit ($W_M$). If cex is also a CEX on $W_M$, the property is falsified; otherwise cex is used to compute a subset ($\Delta B$) of $B$ to refine the current abstraction ($\Delta B$ will be un-abstracted). Note that a nonempty $\Delta B$ exists because cex can always be blocked by un-abstracting some signals. Set $B$ is updated by removing $\Delta B$. A new abstraction is derived for the next iteration of recursive blocking. If PDR-WLA successfully blocks bad cubes at the current depth $k$, then it increments the depth by one and adds a new element ($\top$) to $\Omega$. It then tries to propagate the clauses in $\Omega$ using induction. If a fixed point is found, then the property holds; otherwise, blocking bad cubes at the new depth will be tried (line 10).

Note that PDR-WLA can be viewed as a PDR algorithm with on-the-fly word-level abstraction. The same trace $\Omega$ is re-used throughout the computation, even though the current abstraction is continuously refined. Thus, important reachability information derived in previous iterations is re-used, resulting in a significant speedup over S-CEGAR.

Algorithm 3 PDR with Word-Level Abstraction (PDR-WLA)

Input: $W_M$ ⊳ $W_M$: the word-level input circuit
Input: $S$ ⊳ $S$: the initial set of targeted signals
Output: status ∈ { SAT, UNSAT }
1: $\text{Iterations} \leftarrow 1$
2: $\Omega \leftarrow \{\text{Init}\}$ ⊳ $\Omega$: the PDR trace
3: $k \leftarrow 0$ ⊳ $k$: the PDR depth
4: $B \leftarrow S$ ⊳ $B$: the set of abstracted signals
5: $W_A \leftarrow \text{CREATEABSTRACTION}(W_M, B)$
6: $\text{while $\Omega$ true do}$
7:     $G_A \leftarrow \text{BitBLAST}(W_A)$
8:     $\text{while true do}$
9:         $\Omega$, cex ← RecBlockCube($G_A$, $\Omega$, $k$)
10:        if cex ≠ \emptyset then
11:            if IsREALCEX($W_M$, cex) then
12:                return SAT
13:            else
14:                $\Delta B \leftarrow \text{REFINE}(G_A, B, cex)$
15:                $B \leftarrow B \setminus \Delta B$ ⊳ Un-abstract some signals
16:                $W_A \leftarrow \text{CREATEABSTRACTION}(W_M, B)$
17:                $G_A \leftarrow \text{BitBLAST}(W_A)$
18:                $\text{Iterations} \leftarrow \text{Iterations} + 1$
19:            end
20:        else
21:            break
22:        end
23:     $\Omega \leftarrow \Omega \cup \{\top\}$ ⊳ Open a new frame
24: $\Omega \leftarrow \text{PROPAGATEBLOCKEDCUBES}(G_A, \Omega)$
25: if $\Omega$ contains a fixed point then
26:     return UNSAT

B. Analysis of PDR-WLA

PDR-WLA represents a general framework for word-level abstraction. It is complementary to other abstraction tech-
niques. The only requirement for soundness is that the derived sequence of abstractions (line 17) is monotone:

**Definition 6.** Let \( \{A_j\} \) be a sequence of abstractions, let \( \{T_j\} \) be their transition relations, and let \( \{\text{Init}_j\} \) be their initial states. \( \{A_j\} \) is monotone if \( \text{Init}_{j+1}(i, s, s') \implies T_j(i, s, s') \) and \( \text{Init}_{j+1}(s) \implies \text{Init}_j(s) \).

**Theorem 1.** Let \( M \) and \( A \) be FSMs where \( T_M \implies T_A \) and \( \text{Init}_M \implies \text{Init}_A \). Given a property \( P \), if \( \Omega = (R_0, R_1, \ldots, R_N) \) is a PDR trace of \( A \) with \( P \), then \( \Omega' = (\text{Init}_M, R_1, \ldots, R_N) \) is a PDR trace of \( M \) with \( P \).

**Proof.** Since \( \Omega \) is a PDR trace of \( A \) with \( P \), we have

\[
R_j(s) \implies R_{j+1}(s) \quad \text{for } 0 \leq j < N
\]
\[
R_j(s) \land T_A(i, s, s') \implies R_{j+1}(s') \quad \text{for } 0 \leq j < N
\]
\[
R_j(s) \implies P(s) \quad \text{for } 0 \leq j < N
\]

Note that \( \Omega' \) is the same as \( \Omega \), except that \( R_0 \) is replaced by \( \text{Init}_M \). Since \( \text{Init}_M \implies R_0 \) and \( T_M \implies T_A \), we have

\[
\text{Init}_M(s) \implies R_1(s)
\]
\[
\text{Init}_M(s) \land T_M(i, s, s') \implies R_1(s')
\]
\[
R_j(s) \land T_M(i, s, s') \implies R_{j+1}(s') \quad \text{for } 1 \leq j < N
\]
\[
\text{Init}_M(s) \implies P(s)
\]

Therefore by Definition 4, \( \Omega' \) is a PDR trace of \( M \) with \( P \). \( \square \)

**Theorem 2.** Algorithm 3 is sound and complete.

**Proof. Soundness.** It is sound to start a new iteration with the previous trace (line 10) because each iteration makes the current abstraction tighter by removing signals from \( B \). Note that \( R_0 \) is the initial states of the original circuit \( (W_M) \) and is shared by all abstractions. Similarly any state variable in clauses from a previous abstraction must remain in the next abstraction because abstractions are monotone. Thus, a trace can be safely copied over to the next abstraction (Theorem 1).

Finally, Algorithm 3 is sound because it returns UNSAT only if it finds an inductive invariant proving the property.

**Completeness.** The algorithm returns SAT only if a CEX is real. Convergence follows because, in each iteration, the size of \( B \) decreases by at least one (otherwise the CEX must be real). The number of iterations is bounded by \( |S| \). \( \square \)

IV. REFINEMENTS

Given a spurious CEX, \( cex \), the goal of refinement is to identify a subset of signals \( \Delta B \) in \( B \), such that if \( \Delta B \) is removed from \( B \), then \( cex \) is blocked in the next iteration. We say that \( \Delta B \) is *un-abstracted*.

A. Simulation-based refinement (SBR)

A simple refinement strategy is to simulate \( cex \) on the original circuit \( (W_M) \) and compare the PPI values (in \( cex \)) with their counterparts in \( W_M \). If the values of a signal \( s \) do not match, then \( s \) is a refinement candidate, i.e., a candidate for un-abstraction. If all such candidates are un-abstracted, the property must hold; thus \( cex \) is blocked. However, this approach often results in too many candidates being un-abstracted, and thus is not a good strategy.

A more advanced way is to use a minimized CEX [15], in which some inputs are assigned to \( X \) (don’t care) while the minimized CEX still falsifies the property using ternary simulation. Those remaining concrete assignments are called *care-set* signals, meaning that, if any assignment in the set is changed, the output would be changed also. This provides a set of good candidates for refinement. If all signals in the care set are un-abstracted, then \( cex \) is very likely to be blocked.

B. Limitations of simulation-based refinement

While simulation-based methods are often good enough in many applications [16], [10], frequently they do not find a minimal set to un-abstract.

Example 2. Consider the original circuit and its abstraction in Figure 1. Suppose a CEX to the abstraction is found (Fig. 1b), where the assignments of PIs and PPIs are

\[
(x, y, a, b, c, d) = (0, 0, 1, 0, 1).
\]

For this example, the care set \( C \) returned by counterexample minimization would be all PPIs, \( C = \{a, b, c, d\} \). If any PPI is assigned an \( X \), the PO would become \( X \) as well; thus all PPIs are in the care set. However, it is clear that the set is not *minimum* because only \( \{a, b\} \) or \( \{c, d\} \) needs to be un-abstracted to get a final abstraction that results in UNSAT.

Therefore, a more effective proof-based strategy for refinement is proposed.

C. Proof-based refinement (PBR)

The proposed refinement is an enhanced version of the one used in UFAR [11]. The procedure is presented, followed by an analysis and comparison with other proof-based methods in the next subsection.

The main idea is that if \( cex \) is spurious, then if the original circuit \( (M) \) is simulated with \( cex \), the property holds in all time frames. This implies that the BMC Formula (1) below is UNSAT, where \( \text{Init}_M \) and \( T_M \) are the initial state and the transition function of \( cex \), and \( \text{out} \) is the output signal which, in general, can depend on the input \( i \) and the current state.

\[
\text{Init}_M(s) \land \bigwedge_{t=0}^{k-1} T_M(i_t, s_t, s_{t+1}) \land \bigwedge_{t=0}^{k} \text{out}(i_t, s_t)
\]

\[
\land \bigwedge_{t=0}^{k} (i_t = \beta(i_t))
\]

(1)

Next, multiplexers are introduced to select between the concrete version and the abstracted version of a signal. If assumptions are made such that all the concrete versions are selected initially, then the resulting BMC formula is still UNSAT and

\footnote{It is possible that each care-set signal is fed by a tree, without overlaps with the trees of other care-set signals. Even if all the care-set signals are un-abstracted, this will not provide enough constraints, and therefore \( cex \) is not blocked.}
a modern SAT solver, such as MiniSat [9], returns the final conflict clause. This contains a subset of the assumptions sufficient for UNSAT. This is an efficient variation of finding an unsat core, and the subset returned is a candidate for $\Delta B$.

The procedure operates in four steps:

1) Starting with the original circuit ($W_M$), for each signal $s$ in $B$, introduce two new PIs, $sel$ and $ppi$, where $sel$ is a Boolean signal and $ppi$ is a bit-vector signal consistent with the signal $s$. Replace $s$ with $s' = ITE(sel, s, ppi)$ where $ITE$ is the if-then-else operator. Depending on the value of $sel$, either the concrete signal ($s$) or the abstracted one ($ppi$) becomes the new signal $s'$.

2) Denote the circuit created in Step 1 as $N$ and unroll it with the values of $cex$ plugged in, and keep $sel$ and $ppi$ as the remaining PIs. The $cex$ values plugged in are initial states and PIs at each time frame.

3) Solve the BMC query (2) below, which is guaranteed to be UNSAT. Note that $\beta(\cdot)$ is the assignment function of $cex, pi_t$ is the original PIs at time $t$, $X_t$ is the set of sel inputs at time $t$, and $x_{tn}$ is the sel input for the $n$-th replaced signal at time $t$. By propagating $x_{tn} = 1$ for all $t$ and $n$, Query (2) is reduced to (1) by construction ($sel = 1$ means that the concrete version is chosen).

$$init_N(s) \wedge \bigwedge_{t=0}^{k-1} \bigwedge_{i=0}^{k} \bigwedge_{t=0}^{k} out(i_t, s_t) \wedge \bigwedge_{t=0}^{k} \bigwedge_{i=0}^{k} x_{tn}$$ (2)

4) Derive a subset $\Delta X$ of $X$ using the assumption interface of a modern SAT solver, and determine $\Delta B$ from $\Delta X^*$. This procedure is different from conventional proof-based methods. Details will be discussed in Section IV-D.

**Example 3.** Consider the circuits in Figure 1. Suppose a CEX to the abstraction (Fig. 1b) is obtained, where the assignments of PIs and PPIs are

$$(x, y, a, b, c, d) = (0, 0, 0, 1, 0, 1)$$

Circuit ($N$), derived by introducing $ITE$s for each PPI, is shown in Figure 2. If all $sel$ PIs $\{s_1, s_2, s_3, s_4\}$ are 1, then the circuit is reduced to the original. Next, PI values ($x = 0, y = 0$) are plugged in, and PPIs $\{a, b, c, d\}$ are left unconstrained. The SAT solver is called to determine if $out$ can be 1. The result must be UNSAT with the assumptions of the $sel$ PIs being all 1. In this case, the subset returned would be either $\{s_1, s_2\}$ or $\{s_3, s_4\}$, which is the minimum set needed. This example demonstrates that PBR can pinpoint a precise set for refinement while a simulation-based approach only gives a rough approximation.

5Signals are consistent if they have the same widths and signedness.

6In our implementation, there is only one free variable $x_n$ associated with the replaced signal, i.e. $x_m \equiv x_{m0} \equiv x_{m1} \equiv \ldots \equiv x_{mn}$ for $1 \leq n \leq |B|$. This way, we have $|B|$ assumptions (instead of $(k+1)|B|$) and the returned $\Delta X$ is exactly our candidate for $\Delta B$.

**D. Comparison of refinement strategies**

Two additional proof-based refinement strategies, PBR-A and PBR-B, are presented compared with SBR (Sec. IV-A) and PBR (Sec. IV-C).

Given a spurious CEX, $cex$, there are at least two more ways to formulate an UNSAT query that can be used for proof-based refinements. $\beta(\cdot)$ is the assignment function of $cex$.

**PBR-A.** This considers Formula (3) below. The idea is that if the values in $cex$ are plugged into the abstraction $T_A$, then $out$ must be 1 at some time frame $t$. Therefore, the formula asserting that $out$ is 0 for all time frames, with $cex$ plugged in, must be UNSAT. One can then compute the subset of PPIs sufficient for UNSAT, deriving a refinement. Note that PBR-A does not use the information of the original circuit and can be considered as a proof-based version of SBR.

$$init_A(s) \wedge \bigwedge_{t=0}^{k-1} \bigwedge_{i=0}^{k} \neg out(i_t, s_t)$$ (3)

**PBR-B.** This uses Formula (4) below. Let $pi_t$ and $ppi_t$ be the original PIs and the PPIs at time $t$, respectively. Similar to PBR (Formula 2), it takes the original circuit into account by introducing MUXes selecting between PPIs and the original signals, creating a circuit $N$. The only difference with PBR is that PBR-B also plugs in the assignments of the PPIs in $cex$ into the formula.

$$init_N(s) \wedge \bigwedge_{t=0}^{k-1} \bigwedge_{i=0}^{k} out(i_t, s_t)$$

$$\wedge \bigwedge_{t=0}^{k} \bigwedge_{i=0}^{k} \bigwedge_{t=0}^{k} x_{tn}$$ (4)
The four refinement strategies are compared using the two examples below.

**Example 4.** Consider the circuits in Figure 1. Suppose a CEX is obtained with the assignments of PIs and PPIs
\[(x, y, a, b, c, d) = (0, 0, 0, 1, 0, 1)\].

SBR and PBR-A would refine all PPIs \(\{a, b, c, d\}\). PBR-B would refine only either \(\{a, b\}\) or \(\{c, d\}\) to obtain a final abstraction. This shows that PBR can get a smaller final abstraction by refining fewer PPIs compared to using SBR and PBR-A.

**Example 5.** Consider slightly different circuits from those in Figure 1: the AND gates \((\land)\) are now replaced by OR gates \((\lor)\) in both the original circuit and its abstraction. Suppose a CEX is obtained with the assignments of PIs and PPIs
\[(x, y, a, b, c, d) = (0, 0, 0, 1, 0, 0)\].

SBR, PBR-A, and PBR-B all would refine \(\{a, b\}\), which is not a final abstraction, requiring another iteration. PBR would refine all PPIs \(\{a, b, c, d\}\), which is a final abstraction. This shows that PBR is able to converge with less iterations than the other three. The insight is that PBR refutes all spurious CEXes under the same assignments of original PIs in cex, while the others only refute CEXes with the same values of both PIs and PPIs.

**E. Proposed refinement (PBR and MFFC)**

From previous analysis, PBR provides a good set of candidate signals that, if un-abstracted, would block CEXes. However, we observed that in many cases, the signals in the fanin cones of those candidate signals would appear in the next iteration of refinement, implying that an additional structural analysis can further improve the speed of convergence.

The main idea is to use the maximum fanout free cones (MFFC) of those candidate signals. The MFFC of a signal \(s\) is a subset of its fanin cone, where each path from a signal in the MFFC to the POs passes through \(s\), i.e. the MFFC of a signal contains all the logic used exclusively by the signal. If a signal is abstracted, its MFFC would be abstracted. On the other hand, if a signal is un-abstracted, its MFFC is better unabstracted also; otherwise, additional iterations may be needed.

In our experience, un-abstracting all candidate signals as well as those in their MFFCs often converges faster, i.e. reaching a final abstraction after fewer iterations. Thus, the proposed refinement operates in three steps:

1. Compute \(\Delta B_{PBR}\), a set of candidate signals, using PBR.
2. Compute \(\Delta B_{MFFC}\), the set of signals in the intersections of the MFFCs of \(\Delta B_{PBR}\) and \(B\).
3. Derive set \(\Delta B\): \(\Delta B = \Delta B_{PBR} \cup \Delta B_{MFFC}\).

**V. RELATED WORK**

**A. Word-level abstraction and model checking**

Most previous work is bounded in that it requires unrolling a circuit to a certain depth \(k\), and then they use SMT solvers \([12], [1], [5], [4], [13]\). These methods rely on Bounded Model Checking (BMC) \([2]\) and/or \(k\)-induction \([17]\). This becomes inefficient when deep unrolling is needed. In practice, BMC- and induction-based approaches are efficient in finding CEXes, but often incapable of producing an inductive invariant, which is required for UMC problems. PDR-WLA addresses unbounded problems and does not require unrolling.

Welp and Kuehlmann proposed a generalization of PDR to the theory of quantifier free formulas over bit-vectors (QF-BV) \([21], [20]\). Hybrid simulation and mixed types of atomic reasoning units are used for inductive and CEX generalization. However, they do not re-use PDR traces nor do they perform word-level abstractions.

The closest work to PDR-WLA is AVERROES \([14]\), a word-level algorithm integrating CEGAR and PDR. It abstracts wide data-paths into uninterpreted predicates, constants, terms, and functions, and solves the abstraction with an SMT-based PDR (where SMT solvers are used instead of SAT). The main differences between PDR-WLA and AVERROES are:

- PDR-WLA re-uses PDR traces derived in previous iterations; AVERROES does not.
- PDR-WLA uses PBR and MFFC as the main refinement strategy; AVERROES uses strategies similar to SBR, PBR-A, and PBR-B.

UFAR \([11]\) is a word-level algorithm that combines CEGAR and bit-level model checking. It abstracts arithmetic operators with black boxes as well as uninterpreted function constraints, and solves the abstraction with a portfolio of tools, including BMC and PDR. However, UFAR does not re-use PDR traces nor does it perform MFFC refinement.

**B. PDR with abstraction**

Vizel et al. proposed L-IC3 \([18]\), a bit-level IC3 with localization abstraction, where state variables are the targeted signals and different abstractions are used in different time frames. Fan et al. showed that gate-level abstraction (GLA) \([16]\) can be integrated with PDR \([10]\). However, both approaches consider only bit-level problems. At the word-level, abstracting only state variables may result in aggressive refinement where the entire logic cone of a flip flop would be refined, limiting scalability. On the other hand, GLA cannot be applied directly to the word level. In particular, it mainly uses SBR without considering MFFC, which could be ineffective as discussed in Section IV-D.

In contrast, PDR-WLA considers not only flip flops, but any type of signals, resulting in a finer-grained abstraction and refinement. Also, it uses specific procedures, PBR and MFFC, to find a final abstraction faster than the bit-level GLA.

**VI. EXPERIMENTAL RESULTS**

Experiments were done to evaluate PDR-WLA using different settings. PDR-WLA is part of the public verification tool, ABC \([6]\) (command %pdr Hình), which can parse word-level Verilog and transform the resulting design into a bit-level circuit by bit-blasting. For comparison, S-CEGAR (Sec. II-D, Algorithm 2) was implemented in ABC (command %abs).
PDR-WLA outperforms S-CEGAR in all but one case. The results are shown in Figure 3b, where PDR-WLA is able to outperform S-CEGAR in all but one case. Note that PDR-WLA and S-CEGAR would be the same if no PDR traces can be re-used. Therefore, only 29 cases with non-zero re-used PDR traces are shown.

The benchmarks used for evaluating PDR-WLA were a set of 195 industrial Verilog RTL designs. Large arithmetic operators and multiplexers were the signals targeted for possible abstraction (set S). A workstation with Intel Xeon E5504 CPUs clocked at 2.0 GHz with 24 GB of RAM was used. A time-out of 3600 seconds was used on all experiments.

First, we compare PDR-WLA to the original PDR [8], in which the input Verilog circuit is immediately bit-blasted. Given a 1-hour time-out, PDR-WLA solves 22 fewer test-cases than PDR (89 vs. 111), but PDR-WLA manages to solve 18 hard cases not solved by PDR. It is likely that many of the 22 cases can’t be abstracted, so trying such is a waste of time. Together they can solve 129 out of 195 benchmarks. Thus PDR-WLA complements PDR and would work well in a portfolio-based word-level model checker like [11].

To demonstrate the importance of re-using PDR traces in PDR-WLA, it was compared with S-CEGAR, which uses a fresh PDR solver in each iteration and does not preserve the reachability clauses across PDR runs. The results are shown in Figure 3, where the x and y axes represent the solving times of PDR-WLA and S-CEGAR, respectively. In Figure 3a, PDR-WLA outperforms S-CEGAR in all but eight cases. After investigation, it turns out that after several iterations of refinement, an abstraction can become combinationally UNSAT, implying that the circuit output can be proved UNSAT with all FFs un-initialized. In those cases, PDR-WLA would work hard to get a non-trivial inductive invariant while S-CEGAR proves that the problem is UNSAT after just one SAT call. To address this problem, PDR-WLA was enhanced to always check if the problem is combinationally UNSAT when an iteration begins. The results are shown in Figure 3b, where PDR-WLA beats S-CEGAR in all but one case.

20 out of the 195 designs were chosen in Table I to give an idea of details such as expected ranges of iterations needed, clauses in PDR traces re-used, and the sizes of B (signals to be abstracted way) in the final abstractions. All are UNSAT; each is characterized by the number of hard signals.

Definition 7. A hard signal is the output of
1) an adder, subtractor with width of at least 8, or
2) a multiplier, divider, modulus with width of at least 4, or
3) a multiplexer with width of at least 8.

The initial set of targeted signals (S) is chosen from hard signals with an upper bound of 50 for each of the three categories (e.g., there can be at most 50 adders in S). For each test-case, we show the runtime of six solvers: a) one PDR, b) one S-CEGAR (%abs), and c) four PDR-WLA versions (%pdra) with different refinement strategies (Sec. IV).

Observations from Table I are given below.

1) PDR-WLA vs. PDR. PDR-WLA generally is more efficient when proving hard problems for which small abstractions can be derived. On the other hand, if a problem cannot be abstracted well (e.g., case 20), PDR performs better.
2) S-CEGAR vs. PDR-WLA. An important factor in the comparison is the number of re-used clauses in all previous PDR traces. If the number is high, a high speedup in PDR-WLA is usually observed. Case 20 is an exception to this, where the re-use number is non-trivial but PDR-WLA is still slower. The reason is that the design becomes combinationally UNSAT after 3 iterations. This problem can be fixed by additional SAT calls as shown in Fig. 3. Note that there can be 0 re-used clauses (e.g., cases 16-19), since all refinements occur at k = 0 and no bad states are blocked at k = 1. If the trace contains only R₀ = Init, no clause can be re-used in the next iteration.
3) SBR (S2) vs. PBR (S5). PBR uses more iterations and derives smaller final abstractions (large |B|) in most cases, implying that PBR leads to more fine-grained and focused refinements.
4) PBR-B (S3) vs. PBR (S5). PBR uses less iterations to find a final abstraction, while PBR-B takes more iterations, which can be avoided by a proper analysis (see Example 5). PBR-B can derive a small final abstraction, but large numbers of iterations can cause poor performance. Note: comparison with PBR-A was not done due to its similarity to SBR.
5) Without MFFC (S4) vs. with MFFC (S5). MFFC can be crucial in preventing unnecessary refinement iterations. This is critical in cases 12, 13, 16, 18, and 19.

VII. Conclusions and Future Work

PDR-WLA efficiently integrates PDR with word-level abstraction. It re-uses PDR traces, or reachability clauses, derived in previous iterations of refinement. An effective refinement strategy, PBR with MFFC, was developed which was shown capable of deriving small final abstractions using fewer iterations. PDR-WLA was implemented in the public verification
TABLE I: Detailed experimental results for 20 unsatisfiable word-level test-cases. #HardSignals is the number of hard signals (Definition 7). |S| and |B| are sizes of the set of the initial targeted signals (S) and the set of signals to be abstracted away for each iteration (B) in Algorithm 3. #ReusedClauses is the number of clauses in PDR traces re-used by PDR-WLA. The number 0 if all refinements occur at k = 0. The details of SBR, MFFC, PBR, and PBR-B can be found in Section IV.

<table>
<thead>
<tr>
<th>ID</th>
<th>#Hard Signals</th>
<th>S</th>
<th>#ReusedClauses</th>
<th>CPU Time (seconds)</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1252</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1437</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1437</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1437</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1437</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1437</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1437</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1437</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1437</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1437</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1437</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1437</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1437</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1437</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>103</td>
<td>75</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1252</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>1252</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>1252</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>1252</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1252</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

References


Future work.

- Integrate BMC into Algorithm 3. The idea is that BMC can help PDR-WLA find spurious CEXes faster. Early prototypes suggest speedups in some benchmarks.
- Develop a good way to shrink abstractions. A shrinking procedure can be useful as shown in GLA [16]. One of the main challenges is that PDR traces cannot be re-used if abstractions are no longer monotone.
- Enhance the refinement strategies with constraints. For example, uninterpreted function constraints are known to be effective for SEC problems; partial interpretation constraints can also be useful. The challenge is to derive and apply constraints efficiently and automatically.