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[1]. A design is modeled as a finite state machine (FSM).

Definition 1. An *FSM* is a tuple $M = (I, O, S, Init, T)$ where I is the set of PIs, O is the set of POs, S is the set of FFs, $Init$ 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

[1]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 $out = 1$ if the two designs are different. Similarly for property checking, out is the output of a monitor, and $out = 1$ if the property fails. In terms of linear temporal logic (LTL), the UMC problem is formulated as $M \models \mathbf{G}\neg out$, i.e. 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) $Init(s) \implies Inv(s)$
2) $Inv(s) \wedge T(i, s, s') \implies Inv(s')$
3) $Inv(s) \implies P(s)$

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, $M = (I, O, S, Init, T)$, and a property P, a *PDR trace* is a sequence of predicate functions, $\Omega = (R_0, R_1, \ldots, R_N)$, such that

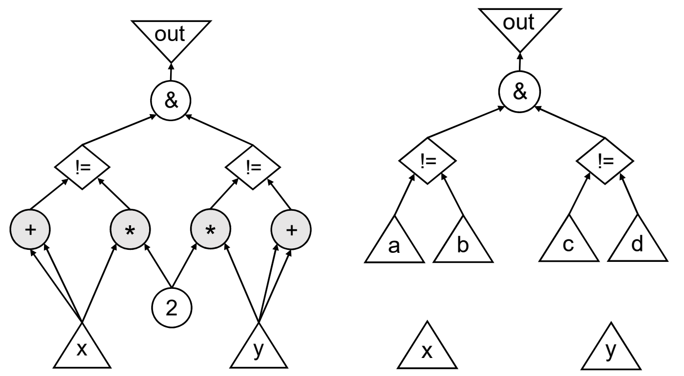
1) $R_0(s) = Init(s)$
2) $R_j(s) \implies R_{j+1}(s)$ for $0 \le j < N$.
3) $R_j(s) \wedge T(i, s, s') \implies R_{j+1}(s')$ for $0 \le j < N$.
4) $R_j(s) \implies P(s)$ for $0 \le j < N$. [2]

Algorithm 1 PDR

```
Input: $G_M$                          ▷ $G_M$: the bit-level input circuit
Output: status ∈ { SAT, UNSAT }
 1: $\Omega \leftarrow \{Init\}$                        ▷ $\Omega$: the PDR trace
 2: $k \leftarrow 0$                                ▷ $k$: the PDR depth
 3: while true do
 4:     $\Omega$, cex ← RecBlockCube($G_M$, $\Omega$, $k$)
 5:     if cex $\neq \emptyset$ then
 6:         return SAT                   ▷ Found a real CEX
 7:     $k \leftarrow k + 1$
 8:     $\Omega \leftarrow \Omega \cup \{\top\}$                 ▷ Open a new frame
 9:     $\Omega$ ← PropagateBlockedCubes($G_M$, $\Omega$)
10:     if $\Omega$ contains a fixed point then
11:         return UNSAT
```

PDR starts with the trace Ω with only one element $R_0 = Init$. 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 Ω, and tries to propagate clauses (using induction) from R_1 to R_k (line 9). If a fixed point ($R_j = R_{j+1}$) is found, the problem is declared UNSAT and the inductive invariant (R_j) is returned.

The details of procedures RecBlockCube and PropagateBlockedCubes can be found in [8].



(a) The original circuit with four arithmetic operators, where x and y are primary inputs, 2 is a constant, $! =$ is the complement of a comparator, & is a bit-wise AND, and out is the negation of the property.

(b) An abstraction derived from the original by replacing the 4 arithmetic operators with 4 new primary inputs, a, b, c, and d.

Fig. 1: A combinational circuit illustrating word-level abstraction. $out \equiv 0$, UNSAT, since $2 \times x \equiv x + x$, which forces out to be constant 0.

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, 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 $out = 1$). Otherwise, it is *spurious*.

[2] $R_N(s)$ does not necessarily imply $P(s)$, i.e. $R_N(s)$ can contain bad states. Recursive blocking (line 4) tries to remove bad states from $R_N(s)$.

[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 $\mathcal{S}$ (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 $\mathcal{B}$, 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 ($\Delta\mathcal{B}$) that, if un-abstracted, can block cex. A new abstraction, with $\Delta\mathcal{B}$ 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: $\mathcal{S}$                      ▷ $\mathcal{S}$: the initial set of targeted signals
Output: status ∈ { SAT, UNSAT }
 1: $Iterations \leftarrow 1$
 2: $\mathcal{B} \leftarrow \mathcal{S}$                   ▷ $\mathcal{B}$: the set of abstracted signals
 3: while true do
 4:     $W_A \leftarrow$ CREATEABSTRACTION($W_M$, $\mathcal{B}$)
 5:     $G_A \leftarrow$ BITBLAST($W_A$)
 6:     cex ← PDR($G_A$)
 7:     if cex $\neq \emptyset$ then
 8:         if ISREALCEX($W_M$, cex) then
 9:             return SAT
10:         else
11:             $\Delta\mathcal{B} \leftarrow$ REFINE($W_M$, $G_A$, $\mathcal{B}$, cex)
12:             $\mathcal{B} \leftarrow \mathcal{B} \backslash \Delta\mathcal{B}$
13:             $Iterations \leftarrow Iterations + 1$
14:     else
15:         return UNSAT
```

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 Ω 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 $\mathcal{S}$, 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 Ω. 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 ($\Delta\mathcal{B}$) of $\mathcal{B}$ to refine the current abstraction ($\Delta\mathcal{B}$ will be un-abstracted). Note that a *nonempty* $\Delta\mathcal{B}$ exists because cex can always be blocked by un-abstracting some signals. Set $\mathcal{B}$ is updated by removing $\Delta\mathcal{B}$. 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 Ω. It then tries to propagate the clauses in Ω 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 Ω 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: $\mathcal{S}$                      ▷ $\mathcal{S}$: the initial set of targeted signals
Output: status ∈ { SAT, UNSAT }
 1: $Iterations \leftarrow 1$
 2: $\Omega \leftarrow \{Init\}$                              ▷ $\Omega$: the PDR trace
 3: $k \leftarrow 0$                                    ▷ $k$: the PDR depth
 4: $\mathcal{B} \leftarrow \mathcal{S}$                 ▷ $\mathcal{B}$: the set of abstracted signals
 5: $W_A \leftarrow$ CREATEABSTRACTION($W_M$, $\mathcal{B}$)
 6:                                  ▷ $W_A$: the word-level abstraction
 7: $G_A \leftarrow$ BITBLAST($W_A$)
 8: while true do
 9:     while true do
10:         $\Omega$, cex ← RECBLOCKCUBE($G_A$, $\Omega$, $k$)
11:         if cex $\neq \emptyset$ then
12:             if ISREALCEX($W_M$, cex) then
13:                 return SAT
14:             else
15:                 $\Delta\mathcal{B} \leftarrow$ REFINE($G_A$, $\mathcal{B}$, cex)
16:                 $\mathcal{B} \leftarrow \mathcal{B} \backslash \Delta\mathcal{B}$    ▷ Un-abstract some signals
17:                 $W_A \leftarrow$ CREATEABSTRACTION($W_M$, $\mathcal{B}$)
18:                 $G_A \leftarrow$ BITBLAST($W_A$)
19:                 $Iterations \leftarrow Iterations + 1$
20:         else
21:             break
22:     $k \leftarrow k + 1$
23:     $\Omega \leftarrow \Omega \cup \{\top\}$                        ▷ Open a new frame
24:     $\Omega \leftarrow$ 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 techniques. 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 $\{Init_j\}$ be their initial states. $\{A_j\}$ is *monotone* if $T_{j+1}(i,s,s') \implies T_j(i,s,s')$ and $Init_{j+1}(s) \implies Init_j(s)$.

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

Proof. Since Ω is a PDR trace of A with P, we have

$$\begin{aligned} R_j(s) &\implies R_{j+1}(s) && \text{for } 0 \le j < N \\ R_j(s) \wedge T_A(i,s,s') &\implies R_{j+1}(s') && \text{for } 0 \le j < N \\ R_j(s) &\implies P(s) && \text{for } 0 \le j < N \end{aligned}$$

Note that Ω' is the same as Ω, except that R_0 is replaced by $Init_M$. Since $Init_M \implies R_0$ and $T_M \implies T_A$, we have

$$\begin{aligned} Init_M(s) &\implies R_1(s) \\ Init_M(s) \wedge T_M(i,s,s') &\implies R_1(s') \\ R_j(s) \wedge T_M(i,s,s') &\implies R_{j+1}(s') \text{ for } 1 \le j < N \\ Init_M(s) &\implies P(s) \end{aligned}$$

Therefore by Definition 4, Ω' is a PDR trace of M with P. □

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 $\mathcal{B}$. 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 $\mathcal{B}$ decreases by at least one (otherwise the CEX must be real). The number of iterations is bounded by $|\mathcal{S}|$. □

IV. Refinements

Given a spurious CEX, cex, the goal of refinement is to identify a subset of signals $\Delta\mathcal{B}$ in $\mathcal{B}$, such that if $\Delta\mathcal{B}$ is removed from $\mathcal{B}$, then cex is blocked in the next iteration. We say that $\Delta\mathcal{B}$ 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[4].

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

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

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 i_t is the input i at time t, s_t is the state variable at time t, k is the depth of cex, $\beta(\cdot)$ denotes the assignment function of cex, and out is the output signal which, in general, can depend on the input i and the current state.

$$\begin{aligned} Init_M(s) \wedge \bigwedge_{t=0}^{k-1} T_M(i_t, s_t, s_{t+1}) \wedge \bigvee_{t=0}^{k} out(i_t, s_t) \\ \wedge \bigwedge_{t=0}^{k} (i_t = \beta(i_t)) \end{aligned} \tag{1}$$

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

[4]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 $\Delta\mathcal{B}$.

The procedure operates in four steps:

1) Starting with the original circuit (W_M), for each signal s in $\mathcal{B}$, introduce two new PIs, sel and ppi, where sel is a Boolean signal and ppi is a bit-vector signal *consistent*[5] 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).

$$Init_N(s) \wedge \bigwedge_{t=0}^{k-1} T_N(i_t, s_t, s_{t+1}) \wedge \bigvee_{t=0}^{k} out(i_t, s_t) \wedge \bigwedge_{t=0}^{k} (pi_t = \beta(pi_t)) \wedge \bigwedge_{t=0}^{k} \bigwedge_{n=1}^{|X_t|} x_{tn} \tag{2}$$

4) Derive a subset ΔX of X using the assumption interface of a modern SAT solver, and determine $\Delta\mathcal{B}$ from ΔX[6].

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 ITEs 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.

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

[6]In our implementation, there is only one free variable x_n associated with the replaced signal, i.e. $x_n \equiv x_{0n} \equiv x_{1n} \equiv \ldots \equiv x_{kn}$ for $1 \leq n \leq |\mathcal{B}|$. This way, we have $|\mathcal{B}|$ assumptions (instead of $(k+1)|\mathcal{B}|$) and the returned ΔX is exactly our candidate for $\Delta\mathcal{B}$.

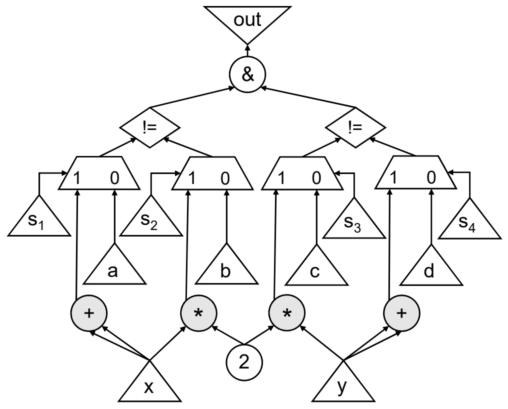


Fig. 2: Example for proof-based refinement, where x and y are original PIs, a-d are pseudo PIs, s_1-s_4 are *sel* PIs. If the assignments of x and y in cex are plugged in, and assumptions are made that s_1-s_4 are all 1, then out is constant-0 (UNSAT).

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.

$$Init_A(s) \wedge \bigwedge_{t=0}^{k-1} T_A(i_t, s_t, s_{t+1}) \wedge \bigwedge_{t=0}^{k} \neg out(i_t, s_t) \wedge \bigwedge_{t=0}^{k} (i_t = \beta(i_t)) \tag{3}$$

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.

$$Init_N(s) \wedge \bigwedge_{t=0}^{k-1} T_N(i_t, s_t, s_{t+1}) \wedge \bigvee_{t=0}^{k} out(i_t, s_t) \wedge \bigwedge_{t=0}^{k} (pi_t = \beta(pi_t) \wedge ppi_t = \beta(ppi_t)) \wedge \bigwedge_{t=0}^{k} \bigwedge_{n=1}^{|X_t|} x_{tn} \tag{4}$$

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 and PBR 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 (&) are now replaced by OR gates (|) 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 un-abstracted 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 abstracion after fewer iterations. Thus, the proposed refinement operates in three steps:

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

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 reuse PDR traces nor does it perform MFFC refinement.

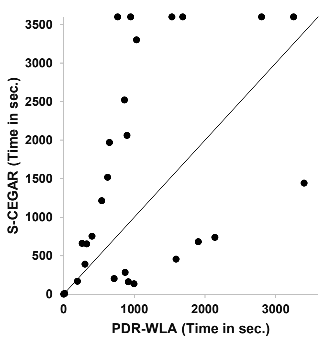
B. PDR with abstraction

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.

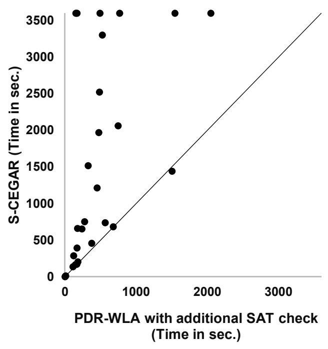
In contrast, PDR-WLA considers not only flip flops, but any type of signals, resulting in a finer-grained abstraction and refinement. Also, it uses specific procedures, PBR and MFFC, to find a final abstraction faster than the bit-level GLA.

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 *%pdra*), 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*).



(a) Running with the default settings, PDR-WLA outperforms S-CEGAR in many cases but not all of them.



(b) With appropriate additional SAT checking, PDR-WLA is able to outperform S-CEGAR in all but one case.

Fig. 3: Comparison of PDR-WLA (*%pdra*) and S-CEGAR (*%abs*). This shows the effectiveness of re-using PDR traces. 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 $\mathcal{S}$). 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 $\mathcal{B}$ (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 ($\mathcal{S}$) 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 $\mathcal{S}$). 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 Ω contains only $R_0 = Init$, 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 $|\mathcal{B}|$) 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 ($\mathcal{S}$) and the set of signals to be abstracted away for each iteration ($\mathcal{B}$) in Algorithm 3. #ReusedClauses is the number of clauses in PDR traces re-used by PDR-WLA. The number is 0 if all refinements occur at $k = 0$. The details of SBR, MFFC, PBR, and PBR-B can be found in Section IV.

ID	#Hard Signals	|S|	CPU Time (seconds)						Iterations					#ReusedClauses				|B| in the last iteration				
			pdr	%abs (S1) SBR +MFFC	%pdra (S2) SBR +MFFC	%pdra (S3) PBR-B +MFFC	%pdra (S4) PBR	%pdra (S5) PBR +MFFC	S1	S2	S3	S4	S5	S2	S3	S4	S5	S1	S2	S3	S4	S5
1	1252	100	479.32	170.50	196.46	369.95	**145.23**	164.67	2	2	3	4	4	11	110	181	181	88	88	98	92	92
2	1437	100	1759.97		3253.29	956.76	931.43	**914.51**	4	4	11	4	4	7438	8129	1493	1493	79	79	81	81	81
3	1437	100	1201.74	653.70	326.80	308.24	**306.83**	335.50	2	2	3	3	3	17	100	155	155	87	87	97	94	94
4	1437	100	1800.60		1529.84	1299.36	**583.27**	597.26	4	5	11	5	5	4981	11061	1732	1732	82	79	82	77	77
5	1437	100	931.73	753.11	401.78	272.46	**169.87**	170.91	2	2	3	3	3	19	84	114	114	87	87	96	90	90
6	1437	100	2531.39		2799.57	1128.25	**672.48**	686.62	4	4	11	6	6	3661	6804	2694	2694	78	78	81	78	78
7	1437	100	1383.61	2521.83	862.04	925.89	**410.96**	415.29	5	4	11	6	6	2080	7241	3317	3317	78	78	85	79	79
8	1252	100	925.41	1213.75	538.48	472.20	**225.96**	227.98	4	4	11	6	6	2518	10122	3889	3889	78	78	83	79	79
9	1437	100	1984.69		949.32	2573.81	387.51	**366.87**	4	4	8	5	5	2304	9868	2198	2198	80	79	82	77	77
10	1437	100	850.77	391.41	302.57	766.21	242.32	**225.09**	2	2	5	5	5	113	625	693	693	90	90	95	94	94
11	1437	100	1151.91	2060.92	896.89	958.62	372.40	**349.45**	4	4	10	5	5	2456	7017	1776	1776	78	78	80	79	79
12	133	101	13.61			675.78		**10.38**	4	4	17	17	10	0	2486	0	0	11	11	21	27	30
13	133	101	15.00			624.42		**8.99**	4	4	16	19	10	0	1713	0	0	15	15	21	26	30
14	94	75			763.06	197.19	295.68	**112.98**	6	6	8	11	6	135	228	551	139	3	3	2	21	3
15	95	75			1685.29	745.23	**259.12**	816.54	7	7	8	11	6	147	115	475	151	3	3	1	21	3
16	82	82		545.37	507.48			**417.23**	3	3	4	12	2	0	0	0	0	12	12	0	0	33
17	72	72	353.69	124.98	128.85	132.70	**77.52**	113.61	9	9	14	18	9	0	0	0	0	16	16	16	14	17
18	58	58	1684.21	1343.36	1237.67	1270.25		**861.53**	3	3	4	9	2	0	0	0	0	13	13	13	13	13
19	2150	103	1731.26	**731.82**	732.24	1544.19		789.06	3	3	18	18	12	0	0	0	0	76	76	77	74	77
20	1132	100	**414.30**	739.13	2138.99	3045.19	2191.30	1296.62	3	3	35	40	33	481	5510	10307	4520	13	13	17	15	9

system ABC and evaluated on industrial benchmarks. PDR-WLA solves more hard problems and offers speedups, compared to PDR and S-CEGAR.

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.

VIII. Acknowledgements

This work was supported in part by SRC contract 2710.001 "SAT-based methods for scalable synthesis and verification" and NSF/NSA grant "Enhanced equivalence checking in crypto-analytic applications".

References

[1] Z. S. Andraus, M. H. Liffiton, and K. A. Sakallah. Reveal: A formal verification tool for verilog designs. In *Proc. of LPAR '08*.

[2] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without bdds. In *Proc. of TACAS'99*.

[3] A. R. Bradley. Sat-based model checking without unrolling. In *Proc. of VMCAI'11*.

[4] B. A. Brady, R. E. Bryant, and S. A. Seshia. Learning conditional abstractions. In *Proc. of FMCAD'11*.

[5] B. A. Brady, R. E. Bryant, S. A. Seshia, and J. W. O'Leary. ATLAS: automatic term-level abstraction of RTL designs. In *Proc. of MEMOCODE'10*.

[6] R. Brayton and A. Mishchenko. ABC: An academic industrial-strength verification tool. In *Proc. of CAV'10*.

[7] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. of CAV'00*.

[8] N. Eén, A. Mishchenko, and R. Brayton. Efficient implementation of property directed reachability. In *Proc. of FMCAD'11*.

[9] N. Eén and N. Sörensson. An extensible sat-solver. In *Proc. of SAT'03*.

[10] K. Fan, M.-J. Yang, and C.-Y. Huang. Automatic abstraction refinement of TR for PDR. In *Proc. of ASP-DAC'16*.

[11] Y.-S. Ho, P. Chauhan, P. Roy, A. Mishchenko, and R. Brayton. Efficient uninterpreted function abstraction and refinement for word-level model checking. In *Proc. of FMCAD'16*.

[12] H. Jain, D. Kroening, N. Sharygina, and E. Clarke. Word level predicate abstraction and refinement for verifying rtl verilog. In *Proc. of DAC'05*.

[13] D. Kroening and M. Purandare. Ebmc: The enhanced bounded model checker. `www.cprover.org/ebmc`.

[14] S. Lee and K. A. Sakallah. Unbounded scalable verification based on approximate property-directed reachability and datapath abstraction. In *Proc. of CAV'14*.

[15] A. Mishchenko, N. Eén, and R. Brayton. A toolbox for counter-example analysis and optimization. In *Proc. of IWLS'13*.

[16] A. Mishchenko, N. Eén, R. K. Brayton, J. Baumgartner, H. Mony, and P. K. Nalla. GLA: Gate-level abstraction revisited. In *Proc. of DATE'13*.

[17] M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a sat-solver. In *Proc. of FMCAD'00*.

[18] Y. Vizel, O. Grumberg, and S. Shoham. Lazy abstraction and sat-based reachability in hardware model checking. In *Proc. of FMCAD'12*.

[19] D. Wang, P.-H. Jiang, J. Kukula, Y. Zhu, T. Ma, and R. Damiano. Formal property verification by abstraction refinement with formal, simulation and hybrid engines. In *Prof. of DAC'01*.

[20] T. Welp and A. Kuehlmann. Property directed reachability for qf_bv with mixed type atomic reasoning units. In *Proc. of ASP-DAC'14*.

[21] T. Welp and A. Kuehlmann. QF BV model checking with property directed reachability. In *Proc. of DATE'13*.