Combining PDR and Reverse PDR for Hardware Model Checking

Tobias Seufert
University of Freiburg, Germany
Email: seufert@informatik.uni-freiburg.de

Christoph Scholl
University of Freiburg, Germany
Email: scholl@informatik.uni-freiburg.de

***Abstract*—In the last few years IC3 resp. PDR attracted a lot of attention as a SAT-based hardware verification approach without needing to unroll the transition relation as in Bounded Model Checking (BMC). Motivated by different strengths of forward and backward traversal already observed in BDD based model checking and by an exponential complexity gap between original PDR and its reverted counterpart '*Reverse* PDR' (which starts its analysis with the initial states instead of the unsafe states as in the original PDR), we take a closer look at Reverse PDR and we present a combined forward/backward version of PDR that inherits the advantages of both original and Reverse PDR. Our experimental results on benchmarks from the Hardware Model Checking Competition demonstrate clear benefits of the combined approach.**

I. Introduction

Sequential circuits still pose a significant challenge in formal hardware verification. With the introduction of bounded model checking (BMC) [1] as an alternative to BDD based methods, SAT based methods have become more and more popular. Interpolation based model checking, introduced in 2003 [2], has been considered the strongest amongst these for a long time. In 2011 though PDR resp. IC3 – as called in its first implementation [3] – caused a stir beating sophisticated multi-engine solvers in hardware model checking competitions. The idea of PDR is to avoid the unrolling of the transition relation and to rather replace small numbers of large and hard SAT problems by many small and easy SAT problems based on a single instance of the transition relation only. A proof is repeatedly strengthened until an inductive invariant or a counterexample is found. Hereby the success of PDR heavily relies on the strength of modern *incremental* SAT solvers such as [4].

The work in this paper is based on observations already made in the context of BDD based symbolic model checking which showed that sometimes forward model checking starting from the initial states and sometimes backward model checking starting from the 'unsafe states' (all states violating a given invariant property) performs better [5], [6]. PDR in its usual definition has however a 'fixed direction': It considers overapproximations of state sets reachable from the *initial states* in k or less steps. (However the overapproximation is guided by the goal to get rid of spurious error paths up to a fixed length to the unsafe states.) For this reason, we take a closer look into *'Reverse PDR'* that considers overapproximations of state sets from which we can reach the unsafe states in k or less steps and the overapproximations are guided by paths starting from the initial states. We analyze which optimizations from PDR can be used in Reverse PDR as well. In particular we give a detailed analysis showing why the methods used in PDR for generalizing proof obligations cannot be transferred to Reverse PDR and we present a new method based on structural arguments. We make an experimental evaluation comparing prototypes of the original and Reverse PDR empirically and thereby show the potential for a combination of the two methods. This is in addition to theoretical results showing an exponential complexity gap between PDR and Reverse PDR (and vice versa). Finally, we present a combined forward/backward version of PDR which inherits the advantages of both original and Reverse PDR. The combined forward/backward version of PDR performs steps of original and Reverse PDR for certain time periods in alternation and exchanges information between both methods. The combined approach is intensively evaluated as well.

Related work: Since the introduction of PDR [3], there have been several improvements on the efficiency of the original algorithm. The authors of [7], e.g., propose the utilization of ternary simulation techniques to have faster and better generalization of newly encountered proof obligations. Tackling the generalization of blocked cubes, [8] introduces the concept of counterexamples to generalization (CTG). These are explanations for failed attempts to generalize blocked cubes. A better generalization exploits the fact that some CTGs may be unreachable though. [9] modifies the core functionality of PDR, e.g., by more aggressively pushing so-called may-proof-obligations forward. Moreover, [9] extends the initial states by additional reachable states which are basically may-proof-obligations that could be proved to be reachable from the initial states. Extending initial states is done in our approach as well, but in our case the initial states are extended due to information transferred from Reverse PDR to PDR. In addition, in our approach the Reverse PDR component extends the unsafe states due to information learnt from PDR.

Whereas a basic version of Reverse PDR has already been proposed in [7] and [8], a really intensive analysis as well as optimizations of Reverse PDR have been missing in our view. [10] uses a similar idea of reverting PDR in the context of automated planning with the interesting insight that reverting the direction generally yields better results in finding plans resp. counterexamples in this special application context.

Other approaches combine ideas of PDR and interpolation based model checking into a unified approach [11] or extend the strength of PDR by combining it with abstraction refinement [12]–[14].

Using bidirectional approaches for BDD based model checking [5], [6] has motivated our approach of combining the original PDR and Reverse PDR. In [15] a bidirectional SAT based verification technique is presented which is based on interpolation instead of PDR. The forward-backward algorithm calculates sequences of intersections between the overapproximations of forward resp. backward reachable states and tries to show that these do not represent a counterexample by strengthening with interpolants.

To the best of our knowledge, a combined bidirectional PDR

978-3-9819263-0-9/DATE18/©2018 EDAA

approach has not been presented and analyzed before.

Structure of the paper: In Sect. II we give some preliminaries needed for this paper. In Sect. III we give a detailed analysis of optimizations that cannot be transferred from PDR to Reverse PDR and propose an alternative optimization. In Sect. IV we present a combined approach after motivating it by an exponential gap between PDR and Reverse PDR. An experimental evaluation is given in Sect. V and Sect. VI summarizes the results with directions for future research.

II. Preliminaries

A. Basic Notions

We discuss the verification of sequential boolean circuits. A sequential boolean circuit consists of a vector of current state variables $\vec{s}$ corresponding to memory elements (flip-flops), a vector of input variables $\vec{i}$, and a vector of output variables $\vec{o}$. A sequential boolean circuit represents an FSM $M := (\mathbb{B}^{|\vec{s}|}, \mathbb{B}^{|\vec{i}|}, \mathbb{B}^{|\vec{o}|}, \delta, \lambda, \mathit{init})$ where the transition function $\delta : \mathbb{B}^{|\vec{s}|} \times \mathbb{B}^{|\vec{i}|} \to \mathbb{B}^{|\vec{s}'|}$ and the output function $\lambda : \mathbb{B}^{|\vec{s}|} \times \mathbb{B}^{|\vec{i}|} \to \mathbb{B}^{|\vec{o}|}$ are defined by a combinational circuit with inputs $\vec{s}$ and $\vec{i}$ and outputs $\vec{s}\,'$ and $\vec{o}$. Here the variables $\vec{s}\,'$ are called next state variables ($|\vec{s}| = |\vec{s}'|$). The predicate $\mathit{init} : \mathbb{B}^{|\vec{s}|} \to \mathbb{B}$ defines the possible initial states of the FSM. As usual, the transition function can also be represented by a predicate $T : \mathbb{B}^{|\vec{s}|} \times \mathbb{B}^{|\vec{i}|} \times \mathbb{B}^{|\vec{s}'|} \to \mathbb{B}$ for the corresponding transition *relation*.

For simplicity, we only consider *invariant* properties over state variables in this paper. Invariants are predicates over state variables and an invariant P holds for an FSM M, if all states occurring on traces starting from some initial state satisfy P. We call the complement of the states represented by P the 'unsafe states', represented by a predicate $\mathit{unsafe} = \neg P$. Thus, our verification goal is to prove (or disprove) that unsafe states cannot be reached from initial states by following transitions of the FSM.

When v is a boolean variable, then v and $\neg v$ are called *literals*. *Cubes* are conjunctions of literals, *clauses* are disjunctions of literals. The negation of a cube is a clause and vice versa. A boolean formula in *Conjunctive Normal Form* (CNF) is a conjunction of clauses. As usual, we often represent a clause as a set of literals and a CNF as a set of clauses. In the following, we abbreviate cubes over current (next) state variables by letters s (s'). By *minterms* we denote cubes containing literals for all state variables. We assume that the transition relation T of an FSM M has been translated into CNF by standard methods like [16]. Modern SAT solvers are able to check the satisfiability of boolean formulas in CNF.

B. PDR and Reverse PDR

The method we will further use and adapt is called property directed reachability (PDR) [7] or IC3 as in its original implementation [3]. PDR produces stepwise reachability information in time-frames without unrolling the transition relation. Each time-frame k corresponds to a predicate R_k represented as a set of clauses, leading to predicates $R_0, \ldots, R_N$ in main loop N of PDR.[1] R_k always overapproximates the set of states which can be reached from *init* in up to k steps. This invariant holds, since in the first main loop R_0 is initialized by *init* (and always remains unchanged), R_1 is initialized by 1 (representing the set of all possible states), and later steps only exclude states s from R_k, if there is provably no transition from R_{k-1} (which overapproximates – by induction assumption – the set of states which can be reached from *init* in up to $k-1$ steps) to s.

Let us consider main loop N when PDR has constructed time frames $R_0, \ldots, R_N$. PDR starts main loop N by extracting satisfying cubes from $R_N \wedge \mathit{unsafe}$ by calling a SAT solver. It generalizes ($\star_1$) a satisfying assignment of $R_N \wedge \mathit{unsafe}$ into a cube s, thus s represents unsafe states for which it has not yet been proven that they can not be reached from *init* in up to N steps. A new proof obligation (s, N) is inserted. It has to be proven that it is not possible to reach the states in s from *init* in up to N steps (otherwise an unsafe state would be reachable from *init*). Discharging this proof obligation is done by asking for *SAT?*$[R_{N-1} \wedge T \wedge s']$ (which corresponds to an pre-image computation) where the cube s' is identical to s, but uses next state variables instead of current state variables.[2] If this SAT check is unsatisfiable, then the proof obligation has been discharged and s is excluded from R_N by adding $\neg s$ to the clauses in R_N. If possible, s is generalized ($\star_2$) before. If the SAT check is satisfiable, a predecessor minterm m is extracted from the satisfying assignment, m is 'generalized' ($\star_3$) to a cube $\hat{s}$, and the proof-obligation $(\hat{s}, N-1)$ is produced. Again, $(\hat{s}, N-1)$ may be proved by an unsatisfiable call *SAT?*$[R_{N-2} \wedge T \wedge \hat{s}']$ or may produce new recursive proof obligations at earlier time frames. Before a new proof obligation is added, PDR always checks whether its cube intersects *init*. If yes, then PDR stops, since there is a path from *init* to *unsafe*, i.e., a counterexample (the reason for inserting a proof obligation $(\tilde{s}, k)$ is exactly the fact that *unsafe* can be reached from all states covered by $\tilde{s}$).

When all proof obligations are discharged and $R_N \wedge \mathit{unsafe}$ becomes unsatisfiable, it has been proven that there is no path of length $\leq N$ from *init* to *unsafe*. Then a new time frame R_{N+1} with $R_{N+1} = 1$ (represented by the empty clause set) is added, and a new main loop is started. If R_{k-1} and R_k are equivalent, then R_k is an inductive invariant and the proof that there is no trace from *init* to *unsafe* is complete.

As already observed in [7] and [8], there is a simple way to arrive at an implementation of Reverse PDR based on the observation that there is a path from *init* to *unsafe* using a transition relation T iff there is a path from *unsafe* to *init* using the 'reverted transition relation'. Thus, a basic version of Reverse PDR is obtained just by exchanging *init* with *unsafe* and interpreting the predicate for T 'the other way round'. In that way, Reverse PDR starts its analysis with the initial states *init* instead of the unsafe states *unsafe* and computes overapproximations $RR_0, \ldots, RR_N$ with RR_k overapproximating the set of states from which *unsafe* can be reached in up to k steps. SAT solver calls of type *SAT?*$[R_N \wedge \mathit{unsafe}]$ are replaced by *SAT?*$[RR_N \wedge \mathit{init}]$ and SAT solver calls of type *SAT?*$[R_{k-1} \wedge T \wedge s']$ are replaced by *SAT?*$[s \wedge T \wedge RR'_{k-1}]$ (here the clauses from RR'_{k-1} are formulated with next state variables instead of current state variables, the cube s with current state variables instead of next state variables), i.e., pre-image computations are replaced by image computations.

[1] In the following we often identify predicates R_k with the state sets represented by them.

[2] Here we review only the basic idea of PDR and omit detailed improvements like strengthening the query by conjunction with $\neg s$.

Many optimizations proposed in the literature including generalizations ($\star_1$) and ($\star_2$) (see [3], [7], [8], e.g.) contribute to the efficiency of PDR and can be easily transferred to Reverse PDR. Unfortunately, the generalization ($\star_3$) mentioned above cannot be directly transferred from PDR to Reverse PDR, since it makes use of the fact that in the verification of sequential circuits the transition relation T results from a circuit, i.e., from transition *functions*, which is not the case for the reverted transition relation in Reverse PDR. Since this generalization of minterms into larger cubes is essential for the efficiency, we will have a closer look into this issue in the next section.

III. Optimization of Reverse PDR

We start with considering the *original* PDR approach in order to make clear why and where a simple transfer of the methods from original to Reverse PDR fails in case ($\star_3$) from the previous section. In the original PDR approach a satisfiable query *SAT?*$[R_{k-1} \wedge T \wedge s']$ provides a satisfying minterm m (expressed with current state variables) which may be generalizable to a satisfying subcube $\hat{s}$ (forming a new proof obligation). The question whether m can be generalized to a satisfying subcube $\hat{s}$ can – in principle – be reduced to a certain QBF problem. In the original PDR approach this QBF problem is approximately (and more efficiently) solved either by an appropriate SAT check [17] or by ternary simulation [7]. Both methods rely on the fact that the transition relation T results from a combinational circuit with current state variables and primary inputs as inputs and next state variables as outputs. We will show that the trick of approximate solving such a QBF problem does not work for Reverse PDR, neither with SAT checking nor with ternary simulation. Therefore for Reverse PDR we resort to a structural method.

Suppose that in the original PDR the query *SAT?*$[R_{k-1} \wedge T \wedge s']$ is satisfiable by an assignment m to the present state variables and $\hat{i}$ to the primary inputs. We would like to check whether we can generalize the proof obligation $(m, k-1)$ to a proof obligation $(\hat{s}, k-1)$ by removing literals from m. This amounts to answering the question whether for each state in the cube $\hat{s}$ there is an assignment to the primary inputs $\vec{i}$ that leads to a next state in the cube s'. Let $\vec{r}$ be the vector of present state variables *not* occurring in the cube $\hat{s}$ and $\vec{t'}$ be the vector of next state variables *not* occurring in the cube s', see Fig. 1 for the notations. Thus, we have to check whether the QBF problem $\forall \vec{r} \exists \vec{i} \exists \vec{t'} : \hat{s} \wedge T \wedge s'$ (1) is satisfiable. We approximate this QBF step by step and in the end we get rid of the quantifier alternation. In a first step we restrict the choice for the primary inputs to the fixed assignment $\hat{i}$ from the original satisfying assignment above. Thus we arrive at an approximate QBF $\forall \vec{r} \exists \vec{t'} : \hat{i} \wedge \hat{s} \wedge T \wedge s'$ (2). Now we consider a fixed assignment $\hat{r}$ to the variables in $\vec{r}$. Since T results from a *circuit* and $\hat{i} \wedge \hat{r} \wedge \hat{s}$ assigns values to all inputs of the circuit, $\hat{i} \wedge \hat{r} \wedge \hat{s} \wedge T$ has *exactly one* satisfying assignment. Thus, $\exists \vec{t'} : \hat{i} \wedge \hat{r} \wedge \hat{s} \wedge T \wedge s'$ is equivalent to $\forall \vec{t'} : (\hat{i} \wedge \hat{r} \wedge \hat{s} \wedge T) \Rightarrow s'$ and Eqn. (2) is equivalent to $\forall \vec{r} \forall \vec{t'} : (\hat{i} \wedge \hat{s} \wedge T) \Rightarrow s'$ (3). Eqn. (3) is satisfiable iff its negation $\exists \vec{r} \exists \vec{t'} : \hat{i} \wedge \hat{s} \wedge T \wedge \neg s'$ (4) is unsatisfiable which corresponds exactly to the SAT formula used in [17]. Unsatisfiability of Eqn. (4) implies satisfiability of QBF formula from Eqn. (1). Minimizing the length of the subcube $\hat{s}$ is reduced to unsatisfiable core techniques.

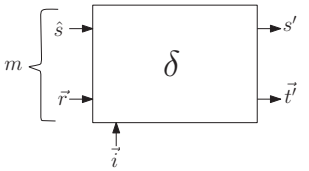

Fig. 1: Notions for PDR

An alternative approximation of Eqn. (1) can be obtained by ternary simulation [7]: We assign $\hat{i}$ to the primary inputs and $\hat{s}$ to a subset of the state inputs. The inputs in $\vec{r}$ are replaced by the unknown value X. If ternary simulation does not propagate the X-values to outputs with literals occurring in s', then the original minterm m can be replaced by $\hat{s}$.

Unfortunately, both methods can not be transferred to Reverse PDR.

Ternary simulation can not be transferred, since we would need to introduce unknown values X at the circuit *outputs* which cannot simply be 'simulated', since the circuit specifying T has a 'fixed direction' and cannot simply be 'reverted'.

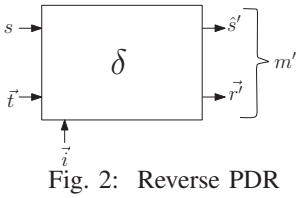

Fig. 2: Reverse PDR

In Reverse PDR, a satisfying valuation of *SAT?*$[s \wedge T \wedge RR'_{k-1}]$ yields a minterm m' in the image of the cube s. The question whether the proof obligation $(m', k-1)$ can be generalized to $(\hat{s}', k-1)$ with a subcube $\hat{s}'$ amounts to the question whether $\hat{s}'$ is completely included in the image of s. Again, this can be formulated as a QBF problem: Let $\vec{r'}$ be the vector of next state variables *not* occurring in the cube $\hat{s}'$ and $\vec{t}$ be the vector of present state variables *not* occurring in the cube s, see Fig. 2 for the notations. We have to check whether the QBF problem $\forall \vec{r'} \exists \vec{i} \exists \vec{t} : s \wedge T \wedge \hat{s}'$ (5) is satisfiable. In contrast to the original PDR, a transformation analogous to the transition from Eqn. (2) to Eqn. (3) is not an equivalence transformation anymore, since we can not use the circuit-based argument as before. Nevertheless, a SAT check that is analogous to Eqn. (4) could be an *approximation* to the QBF from Eqn. (5). The analogous SAT formula $\exists \vec{r'} \exists \vec{t} : \hat{i} \wedge \neg s \wedge T \wedge \hat{s}'$ (6) is *not* an approximation due to a subtle reason: Unsatisfiability of Eqn. (6) guarantees the following: *If* for a valuation $\hat{r}'$ to the removed variables $\vec{r'}$ there is a predecessor of $(\hat{s}', \hat{r}')$ under primary input assignment $\hat{i}$, then it is included in s. However, since it is *not* guaranteed that there is any predecessor of $(\hat{s}', \hat{r}')$ under input $\hat{i}$ *at all*, it is not guaranteed that $(\hat{s}', \hat{r}')$ occurs in the image of s. This means that it is *not sound* to use unsatisfiablity of Eqn. (6) for reducing a minterm m' to a subcube $\hat{s}'$.

Using the QBF from Eqn. (5) for checking whether minterms in proof obligations can be shortened to subcubes does not seem to be advisable however, since QBF solving is much harder than SAT solving in practice and repeated QBF queries not only for a fixed $\hat{s}'$, but for minimizing the size of the subcube $\hat{s}'$ causes too much overhead. On the other hand, generalizing satisfiable cases has been proven to be heavily effective in [7]. For that reason we use a structural method as a rough approximation which nevertheless is successful in practice as experimental results in Sect. V show. A related and slightly simpler structural check has been used in the context of SAT based image computation [18].[3]

[3]For SAT based image computation the sizes of cubes in the image have to be maximized in order to make the enumeration of image cubes more effective.

The subcube $\hat{s}'$ has to fulfill the following condition: In the part of the image of s where the next state variables which are *not* removed have the valuation $\hat{s}'$ (i.e. exactly the same valuation as in m'), all possible combinations from $\mathbb{B}^{|\vec{r'}|}$ occur for the values of $\vec{r'}$.

A sufficient condition (*) implying this condition is:

> Let $\delta_i|_s(\vec{t},\vec{i})$ be the cofactors of transition functions δ_i with state variables fixed to s, $supp(\delta_i|_s)$ their (structural) support sets. For all δ_{i_j} computing r'_j we have $supp(\delta_{i_j}|_s) \cap supp(\delta_i|_s) = \emptyset \; \forall i \neq i_j$ and $\delta_{i_j}|_s$ is different from the constant functions 0 and 1.

The idea of condition (*) is as follows: Assume that we fix the primary inputs and the current state variables such that s occurs at the inputs and $\hat{s}'$ occurs at the outputs of the transition function $\vec{\delta}$. Due to the conditions for the support sets in (*) we can change the variables in the support sets of $\delta_{i_j}|_s$ computing r'_j arbitrarily without changing $\hat{s}'$. Since the support sets of those transition functions $\delta_{i_j}|_s$ are in addition disjoint from each other and they are not constant, we can produce arbitrary value combinations at their outputs without changing $\hat{s}'$.

If *SAT?*$[s \wedge T \wedge RR'_{k-1}]$ is satisfiable with the satisfying minterm m', we consider a bipartite graph $G = (V, E)$ for checking which literals can be removed from m': V is partitioned into two disjoint node sets V_1 and V_2. The nodes in V_1 correspond to variables from $(\vec{t},\vec{i})$, the nodes in V_2 correspond to the cofactors $\delta_i|_s$ of the transition functions, $\{v_1, v_2\} \in E$ with $v_1 \in V_1$, $v_2 \in V_2$ iff $v_1 \in supp(v_2)$. According to condition (*) we can remove a literal from m', if it corresponds to a non-constant function $\delta_i|_s$ whose node $v_2 \in V_2$ is only connected to nodes $v_1 \in V_1$ with degree 1 (**).

Fixing additional variables from $(\vec{t},\vec{i})$ to the constants from the satisfying assignment increases the chance to obtain disjoint support sets (of course at the risk of turning $\delta_i|_s$ into constant functions). We use the following heuristics: In step 1 we set all $v_1 \in V_1$ with $degree(v_1) > k_1$ to constants (k_1 is a constant and chosen as $0.5 \times |V_2|$ in our implementation). Those nodes v_1 and their outgoing edges are removed from G. The intuition is that a node v_1 with a high degree prevents many literals from being removed from m'. In step 2 we consider for each node $v_2 \in V_2$ whether condition (**) 'almost holds'. Let $D_1 = \{v_1 \in V_1 \,|\, (v_1, v_2) \in E, degree(v_1) = 1\}$ and $D_2 = \{v_1 \in V_1 \,|\, (v_1, v_2) \in E, degree(v_1) > 1\}$. If $\frac{|D_1|}{|D_1|+|D_2|} > k_2$, then the variables from D_2 are set to constants and they are removed from G. (In our implementation $k_2 = 0.95$.) After steps 1 and 2 we obtain the literals to be removed from m' by checking condition (**). In our implementation non-constant functions are determined approximately by simulating 20 randomly chosen input vectors.

IV. Combining PDR and Reverse PDR

A. Relation between PDR and Reverse PDR

Our experiments in Sect. V show that it is worthwhile to consider both original and Reverse PDR, since they outperform each other on different benchmark instances. There may be even an exponential gap between PDR and Reverse PDR:

Theorem 1: There are sequential circuits for which PDR causes exponentially more SAT queries than Reverse PDR and vice versa.

The proof is omitted due to lack of space and can be found in [19]. These observations motivate a combination of PDR and Reverse PDR into a single algorithm that inherits the advantages of both directions.

B. Combined Algorithm

In Alg. 1 we present our algorithm combining the original (or forward) PDR with the Reverse (or backward) PDR. Basically, it runs the original PDR for some time limit $tlimit^{fw}$, then Reverse PDR for some time limit $tlimit^{bw}$, afterwards it resumes the original PDR again for time $tlimit^{fw}$ etc.. Whenever one of the two directions finishes with the result "$safe$" or "$unsafe$" (lines 5 or 9 of Alg. 1), the combined algorithm finishes with this result.

```
1  init_PDR(init, unsafe); init_RPDR(init, unsafe);
2  po^fw := ∅; po^bw := ∅;
3  while true do
4      (res, new_po^fw) := resume_PDR(po^bw, tlimit^fw);
5      if res ≠ unknown then return res;
6      po^fw := po^fw ∪ new_po^fw;
7      compress(po^fw, slimit^fw);
8      (res, new_po^bw) := resume_RPDR(po^fw, tlimit^bw);
9      if res ≠ unknown then return res;
10     po^bw := po^bw ∪ new_po^bw;
11     compress(po^bw, slimit^bw);
```

Algorithm 1: Combined forward and backward PDR.

Information exchange between the two directions of PDR takes place on the basis of proof obligations. Sets of proof obligations po^{fw} of the original PDR represent *underapproximations* of the set of states from which the unsafe states can be reached. Therefore, in Reverse PDR the set of unsafe states can be extended by the proof obligations po^{fw} as a "target enlargement". Extending $unsafe$ in Reverse PDR by po^{fw} has two effects:

1) During Reverse PDR, the intersection of a cube s that can be reached from $init$ (i.e., of a proof obligation in Reverse PDR) with the extended $unsafe$ may now be non-empty, because of a non-empty intersection of s with a proof obligation from forward PDR. Due to this non-empty intersection, a counterexample, i.e., a trace from the initial states to $unsafe$, has been found where the first part of the trace (reaching s from $init$) has been constructed by Reverse PDR and the second part (reaching $unsafe$ from s) has been constructed by forward PDR.

2) Sometimes in (Reverse) PDR the generalization of cubes s into $\hat{s}_1$ for unsatisfiable cases (see Sect. II) may be unnecessarily large such that it prevents early convergence of the procedure. In the combined algorithm unnecessary large generalizations are restricted by the stronger requirement that a generalized cube $\hat{s}_2$ does not intersect the *extended* unsafe states, not only the unsafe states as in the original Reverse PDR. If a larger $\hat{s}_1$ that includes states from which *unsafe* can be reached (as has been proved by forward PDR) is removed from RR_k, then the procedure cannot converge with $RR_k \equiv RR_{k+1}$, since the sets RR_i overapproximate the set of states from which *unsafe* can be reached in i steps and thus some of the excluded states have to be added later on, at higher time frames.

Of course, a dual argumentation is possible for transferring information from Reverse PDR to PDR: Sets of proof obligations po^{bw} of Reverse PDR represent *underapproximations* of the set of states which can be reached from the initial states.

So in the original PDR the set of initial states can be extended by the proof obligations po^{bw}.

Since the number of proof obligations new_po^{fw} that are computed by forward PDR in line 4 and can be transferred to backward PDR in line 8 may become huge, adding *all* such proof obligations to $unsafe$ in backward (Reverse) PDR may slow down the checks for non-empty intersections of proof obligations with the extended $unsafe$. Therefore we reduce the set of transferred proof obligations as soon as their number exceeds an upper bound $slimit^{fw}$ (see lines 6 and 7). In the procedure $compress$ (line 7) proof obligations with lower quality are removed first. We consider a proof obligation (t, m) of forward PDR to have higher quality, if forward PDR had to invest more effort for computing (t, m), i.e., if the distance of t from $unsafe$ is higher. As a secondary quality measure we use the *size* of the cube t which (heuristically) increases the chance of finding error paths by intersections of t with proof obligations in Reverse PDR.

The reduction of proof obligations transferred from Reverse PDR to the original (forward) PDR (see lines 10 and 11) is performed in an analogous manner.

V. Experimental Results

Our implementation of original PDR is derived from [7] and augmented to support Reverse PDR too. Besides the presented features in [7] we also use features of Bradley's original algorithm and newer developments. We have extended our implementation by the better generalization scheme from [8] for instance. The transition relation is represented as a Tseitin-transformed CNF [16], preprocessed with variable elimination. We use one MiniSat v2.2.0 [4] instance per time frame. The Reverse PDR implementation uses the same code base as our original PDR implementation. All experiments have been run with constrained resources – 7 GB for memory and 3600 s for execution time as in HWMCC'15.[4] We have used one core of an Intel Xeon CPU E5-2650v2 with 2.6 GHz. Further, all experiments have been run on the complete benchmark set of HWMCC'15 excluding the access restricted Intel benchmarks. As proposed in [8] we use the functional versions of the reverse encoded *beem* benchmarks in order to avoid any bias in favor of Reverse PDR. To obtain a clear and fair evaluation of PDR resp. Reverse PDR only, we refrain from using any other complete or incomplete proof engine (like BMC, SMC, interpolation) or preprocessing. As reference we used the latest IC3 implementation[5] published by Aaron Bradley as well as the PDR implementation of ABC[6]. Both use variable elimination too.

a) Comparison of Reverse PDR with and without structural generalization of proof obligations: To evaluate the effect of the generalization of proof obligations by the structural method from Sect. III we compare versions of Reverse PDR with and without the structural generalization. In Fig. 3 we present the results. Apparently, the overall performance of Reverse PDR greatly improves. The method has been applicable[7] in 81.1% of the solved benchmarks and- when applied- cubes have been shortened by an average of 21.7% of their original size. A detailed analysis showed that these cube reductions are directly related to reduced numbers of proof obligations which have to be discharged. On average, generalization of proof obligations led to 26.4% less processed proof obligations in all solved benchmarks.

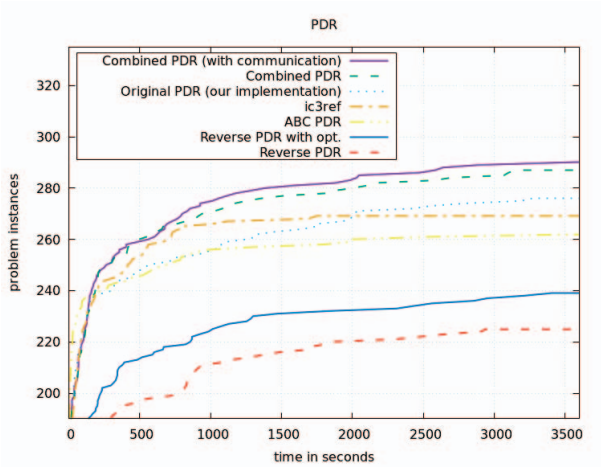

Fig. 3: Comparison of Implementations

b) Comparison of original PDR and Reverse PDR: The scatter plot in Fig. 4 displays the required execution time of all benchmarks for original and Reverse PDR. The wide spreading suggests that there are lots of benchmarks on which Reverse PDR outperforms the original version by magnitude and the other way around. Although Fig. 3 shows an overall advantage of the original PDR over Reverse PDR, the complementary strengths of both approaches demonstrated by Fig. 4 provide a motivation to combine the advantages of the approaches.

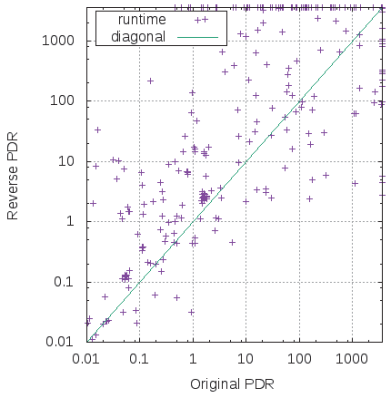

Fig. 4: Reverse and original PDR

c) The combined PDR approach: The configuration of the combined algorithm from Sect. IV we have chosen for our experiments uses time limits $tlimit^{fw} = tlimit^{bw} = 30\ s$ for continuous executions of PDR / Reverse PDR and upper bounds for proof obligations $slimit^{bw} = 500$ and $slimit^{fw} = 500$. We had our best results by using fixed upper bounds for proof obligations and replacing 'shallow' and less general proof obligations with 'deeper' and shorter ones using our metric from Sect. IV. We also tried an activity-based scheme as suggested in [9] for reachable may-proof-obligations, but found it inferior to our approach.[8] Fig. 3 displays the results showing a considerable increase in solved benchmarks for the combined approach. The gain in overall performance using communication via proof obligations is also clearly visible. A detailed analysis of execution times is shown in Fig. 5 for

[4]http://fmv.jku.at/hwmcc15/

[5]https://github.com/arbrad/IC3ref, downloaded in Sep. 2016.

[6]https://bitbucket.org/alanmi/abc, from 9/10/2017, default settings.

[7]This means there has been at least one succesful run, i.e. at least one next state variable has been pruned.

[8]Possibly because we are able to use deep and also generalized proof obligations.

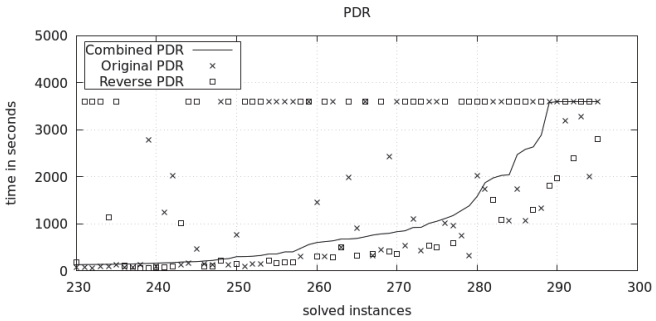

Fig. 5: Detailed analysis of solved benchmarks.

PDR	benchm.	time	learnt cl.	opened fr.	cex len.
comb.	6s406rb111	562.3s	890	15	-
orig.	6s406rb111	timeout	8892	15	-
rev.	6s406rb111	timeout	68	11	-
comb.	beemskbn2f1	1578.5s	3589	49	116
orig.	beemskbn2f1	2025.6s	4116	54	126
rev.	beemskbn2f1	timeout	3966	21	-
comb.	6s284rb1	1978.3s	5065	31	125
orig.	6s284rb1	timeout	24471	22	-
rev.	6s284rb1	1514.9s	7381	31	108

TABLE I: Benchmarks: 6s406rb111 (unsat), beemskbn2f1 (sat), 6s284rb1 (sat).

all solved benchmarks[9]. The combined algorithm successfully combines the advantages of PDR and Reverse PDR in the sense that its run times are usually in the order of the doubled minimum of the run times for PDR and Reverse PDR, or below that. The communication between original and Reverse PDR via proof obligations enables the combined algorithm to run even faster (sometimes significantly faster) than the doubled minimum of the run times for PDR and Reverse PDR. In a few cases even *both* standalone approaches run into a timeout whereas the combined approach solves the benchmark. We take a closer look at some of the benchmarks in Tab. I. For *6s406rb111*, both standalone methods have a timeout whereas the combined algorithm terminates with an inductive invariant during the original PDR part after 562.3 seconds with a total of 890 learnt clauses (blocked cubes) in contrast to 8892 blocked cubes just before timeout when using standalone original PDR. Looking at benchmark *beemskbn2f1* it turns out that the combined algorithm finishes its analysis with a counterexample of length 116 during execution of the original PDR part after 1578.5 seconds and 49 main iterations (time frames) while original PDR requires 54 main iterations and 2025.6 seconds for a counterexample of length 126. Reverse PDR does not terminate after 3600 seconds. *6s284rb1* is solved by the Reverse PDR part of the combined approach whereas the original PDR has a timeout. Note that the combined PDR approach is able to produce counterexamples much longer than the trace by pushing proof obligations into later time frames (as in the original PDR) *and* by using information learnt from the opposite direction. We also tested how generalization of proof obligations (see Sect. III) within *Reverse* PDR affects original PDR run times in the combined approach. We found out that there are indeed positive effects. Again we present two exemplary benchmarks in Tab. II; they are solved by the original PDR part of the combined approach.

[9]We provide result tables and binaries under http://bit.ly/2wXZNFd

PDR	benchm.	time	learnt cl.	opened fr.	cex len.
with gen.	oski1rub03i	692.2s	210	10	-
without	oski1rub03i	1665.53	1978	17	-
with gen.	6s320rb0	1282.6s	8316	9	9
without	6s320rb0	1756.7s	10531	9	9

TABLE II: Benchmarks: oski1rub03i (unsat), 6s320rb0 (sat).

VI. Conclusions and Future Work

We showed that there is not only a theoretical complexity gap between original PDR and Reverse PDR, but also a practical gap in the execution of the latest HWMCC benchmark suite. We put this fact to use by implementing a combined algorithm which is able to solve a lot more benchmarks than each standalone PDR variant. By exchanging proof obligations we developed a way in which original and Reverse PDR can profit from each other while being executed alternately.

We showed that there is optimization potential for Reverse PDR in the generalization of proof obligations even though known generalization approaches from original PDR do not apply.

Apparently there is still work to do in analyzing and optimizing Reverse PDR. Furthermore, communication and cooperation between original and Reverse PDR may have not been utilized to its full potential yet, leaving this a promising field of research.

References

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *TACAS*, 1999, pp. 193–207.
[2] K. L. McMillan, "Interpolation and SAT-based model checking," in *CAV*, 2003, pp. 1–13.
[3] A. R. Bradley, "Sat-based model checking without unrolling," in *VMCAI*, ser. LNCS, vol. 6538. Springer, 2011, pp. 70–87.
[4] N. Eén and N. Sörensson, "An extensible SAT-solver," in *SAT*, 2003, pp. 502–518.
[5] G. Cabodi, S. Nocco, and S. Quer, "Mixing forward and backward traversals in guided-prioritized BDD-based verification," in *CAV*, ser. LNCS, vol. 2404. Springer, 2002, pp. 471–484.
[6] C. Stangier and T. Sidle, "Invariant checking combining forward and backward traversal," in *FMCAD*, 2004, pp. 414–429.
[7] N. Eén, A. Mishchenko, and R. K. Brayton, "Efficient implementation of property directed reachability," in *FMCAD*, 2011, pp. 125–134.
[8] Z. Hassan, A. R. Bradley, and F. Somenzi, "Better generalization in IC3," in *FMCAD*, 2013, pp. 157–164.
[9] A. Ivrii and A. Gurfinkel, "Pushing to the top," in *FMCAD*, 2015, pp. 65–72.
[10] M. Suda, "Property directed reachability for automated planning." *J. Artif. Intell. Res.(JAIR)*, vol. 50, pp. 265–319, 2014.
[11] Y. Vizel and A. Gurfinkel, "Interpolating property directed reachability," in *CAV*, ser. LNCS, vol. 8559. Springer, 2014, pp. 260–276.
[12] Y. Vizel, O. Grumberg, and S. Shoham, "Lazy abstraction and SAT-based reachability in hardware model checking," in *FMCAD*. IEEE, 2012, pp. 173–181.
[13] J. Baumgartner, A. Ivrii, A. Matsliah, and H. Mony, "IC3-guided abstraction," in *FMCAD*, 2012, pp. 182–185.
[14] J. Birgmeier, A. R. Bradley, and G. Weissenbacher, "Counterexample to induction guided abstraction refinement (CTIGAR)," in *CAV*, ser. LNCS, vol. 8559. Springer, 2014, pp. 831–848.
[15] Y. Vizel, O. Grumberg, and S. Shoham, "Intertwined forward-backward reachability analysis using interpolants," in *TACAS*, ser. LNCS, vol. 7795. Springer, 2013, pp. 308–323.
[16] G. Tseitin, "On the complexity of derivations in propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logics*, 1968.
[17] H. Chockler, A. Ivrii, A. Matsliah, S. Moran, and Z. Nevo, "Incremental formal verification of hardware," in *FMCAD*, 2011, pp. 135–143.
[18] P. Chahan, E. Clarke, and D. Kroening, "Using SAT based image computation for reachability analysis," Carnegie Mellon University, Tech. Rep. CMU-CS-03-151, September 2003.
[19] T. Seufert and C. Scholl, "Sequential verification using Reverse PDR," in *MBMV*, 2017, pp. 79–89.