fbPDR: In-depth combination of forward and backward analysis in Property Directed Reachability

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*—We describe a thoroughly interweaved forward and backward version of PDR/IC3 called *fbPDR*. Motivated by the complementary strengths of PDR and Reverse PDR and by related work showing the benefits of collaboration between the two, *fbPDR* lifts the combination to a new level. We lay the theoretical foundations for sharing information represented by learned lemmas between PDR and Reverse PDR and demonstrate the effectiveness of our approach on benchmarks from the Hardware Model Checking Competition.**

I. Introduction

Nowadays PDR (or IC3) [1], [2] is considered as one of the most powerful methods in hardware verification. Unlike others it does not unroll a transition relation. It incrementally strengthens a proof until a safe invariant or a counterexample is found. PDR in its usual definition has a ‘fixed direction’: It considers overapproximations of state sets reachable from the *initial states* in k or less steps. The work in this paper is motivated by observations already made in [3], [4] showing that a combination of (forward) PDR and (backward) Reverse PDR is worthwile. Reverse PDR computes overapproximations of state sets which can reach *unsafe states* in k or less steps. In [4] Reverse PDR has been examined thoroughly and communication between PDR and Reverse PDR via proof obligations has been enabled. We now utilize these insights to lift the combination of PDR and Reverse PDR to a next level. Our algorithm really intertwines PDR and Reverse PDR reasoning by strengthening one trace using blocked cubes learnt from the other. Thus both variants profit from all the information gathered by its counterpart and we observe a significant speedup in both finding counterexamples as well as safe inductive invariants.

In general, our objective is to share as much information between PDR and Reverse PDR as possible. The contributions of our paper can be summarized as follows: (1) We show how to learn new lemmas (the negation of blocked cubes) for PDR (resp. Reverse PDR) based on lemmas learned for Reverse PDR (resp. PDR) and show how to make use of those lemmas to strengthen existing PDR / Reverse PDR traces. (2) For extracting blocked cubes from state sets represented in CNF we present and compare several orthogonal methods. (3) We expand traces by additional (initially empty) frames whenever we learn from the other PDR direction that we can exclude counterexamples of shorter lengths. Moreover, we immediately discharge pending proof obligations based on information on the minimal counterexample length provided by the other direction. (4) We provide proofs for our statements and analyze thoroughly how PDR / Reverse PDR with all their implementation details (like delta-encoded traces, e.g.) can be enhanced with information exchange to result in a sound overall implementation. (5) We show that our new insights on information exchange can be combined with the work from [4] in an effective manner. (6) In our experiments, we provide an intensive evaluation and analysis of different variants of our approach compared to related work and state-of-the-art PDR implementations.

Related work: [5] modifies the core functionality of PDR, e.g., by more aggressively pushing so-called may-proof-obligations forward. Moreover, [5] extends the initial states by additional reachable states that are proved to be reachable from the initial states during the attempt to discharge may-proof-obligations. Extending initial states is done in [4] as well, but in this case the initial states are extended due to information transferred from Reverse PDR to PDR. In addition, 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 [2], [6], an intensive analysis as well as optimizations of Reverse PDR have been presented in [4]. The approach of combining PDR and Reverse PDR has been motivated by bidirectional approaches for BDD based model checking [7], [8] and also SAT based model checking using interpolation [9]. In multi-threaded portfolio configurations used by [10], [11] exchanging reachability information via learned lemmas between different *forward* PDR variants has successfully been applied. Also the approach from [12] reuses reachability information (represented by learned lemmas) by initializing PDR with invariants which have been found earlier while proving a different property on multi-property designs.

To the best of our knowledge, a combined bidirectional PDR approach sharing information represented by learnt lemmas 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 present our intertwined approach and prove its soundness. An experimental evaluation is given in Sect. IV and Sect. V 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 represents an FSM $M := (\mathbb{B}^{|\vec{s}|}, \mathbb{B}^{|\vec{i}|}, \mathbb{B}^{|\vec{o}|}, \delta, \lambda, \mathit{init})$ where the transition function $\delta\colon \mathbb{B}^{|\vec{s}|} \times \mathbb{B}^{|\vec{i}|} \to \mathbb{B}^{|\vec{s}'|}$ and the output function $\lambda\colon \mathbb{B}^{|\vec{s}|} \times \mathbb{B}^{|\vec{i}|} \to \mathbb{B}^{|\vec{o}|}$ 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 $\mathit{init}\colon \mathbb{B}^{|\vec{s}|} \to \mathbb{B}$ defines the possible initial states of the FSM. As usual, the transition function can also be represented by a predicate $T\colon \mathbb{B}^{|\vec{s}|} \times \mathbb{B}^{|\vec{i}|} \times \mathbb{B}^{|\vec{s}'|} \to \mathbb{B}$ for the corresponding transition *relation*.

For simplicity, we only consider *invariant* properties P over state variables in this paper. We call the complement of the states represented by P the 'unsafe states', represented by a predicate *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.

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.

B. PDR and Reverse PDR

The method we will further use and adapt is called property directed reachability (PDR) [2] or IC3 [1]. 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 $R_0, \dots, R_N$ 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_k with $k \geq 1$ are initialized by 1 (representing the set of all possible states), and all main loops 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 $R_0, \dots, R_N$. PDR starts main loop N by extracting satisfying cubes from $R_N \wedge$ *unsafe* by calling a SAT solver. It generalizes a satisfying assignment of $R_N \wedge$ *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, 0) is inserted. A proof obligation is identified by its state set (here s), by its frame (here N), and by its depth (here 0) – the depth describes the number of steps required to reach $unsafe$ from an arbitrary state in s. 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 before. If the SAT check is satisfiable, a predecessor minterm m is extracted from the satisfying assignment, m is 'generalized' to a cube $\hat{s}$, and the proof obligation $(\hat{s}, N-1, 1)$ is produced which has to be discharged later on. 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. All newly added proof obligations (s, k, d) are also added as (s, k', d) in all higher time frames $N \geq k' > k$, since paths of length $k'+d$ from *init* to *unsafe* should be excluded as well.

[1]For subsets C of the clause set R_k we write $C \sqsubseteq R_k$, but in the following we often identify predicates R_k with the state sets represented by them and thus we often write $s \subseteq R_k$, if s is a cube representing a subset of states.

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

PDR keeps the invariant that for all $k > 1$ the clause set R_k is even *syntactically* included in R_{k-1} (and thus the state set represented by R_{k-1} is a subset of the state set represented by R_k). The invariant holds in the beginning of the algorithm. If *SAT?*$[R_{k-1} \wedge T \wedge s']$ is unsatisfiable, the same SAT check is also unsatisfiable for k replaced by i with $1 \leq i \leq k-1$ (since the state set R_{i-1} is a subset of R_{k-1}). Therefore s can be excluded (by adding $\neg s$ to the clause set) from R_i with $1 \leq i \leq k-1$ as well. The state set represented by R_0 remains always equal to *init*. It is a subset of R_k ($1 \leq k \leq N$), since no proof obligations (s, k, d) with s intersecting *init* are generated and thus it never happens that an initial state is excluded from R_k.

When all proof obligations are discharged and $R_N \wedge$ *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 there is a pair of state sets R_{k-1} and R_k, where 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 [2] and [6], there is a simple way to arrive at an implementation of Reverse PDR just by exchanging *init* with *unsafe* and interpreting the predicate for T 'the other way round'. In that way, Reverse PDR computes overapproximations $RR_0, \dots, RR_N$ 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?*$[R_N \wedge$ *unsafe*$]$ are replaced by *SAT?*$[RR_N \wedge$ *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.

III. Combining PDR and Reverse PDR

The results from [3], [4] indicate that it is worthwile to combine the complementary strengths of PDR and Reverse PDR into one algorithm and also to enable collaboration between the two. In this paper we extend the collaboration potential considerably.

Note that in the following our analysis considers only the transfer of information from Reverse PDR to PDR. However, all procedures also apply the other way around considering the characteristics of PDR and Reverse PDR from Sec. II-B.

A. Learning new lemmas from Reverse PDR

Basically, PDR gathers information in terms of proof obligations and learnt clauses (lemmas). Proof obligations of Reverse PDR are forward reachable states and can therefore just be added to $init$ in original PDR [4]. Here we look into *learning new clauses (lemmas)* for PDR from lemmas in Reverse PDR which is more subtle though: Consider a Reverse PDR trace $RR_0, RR_1, \dots, RR_N$. For a clause (lemma) $c \in RR_{N-i}$ with $c = \overline{s}$, s contains a proof obligation $\hat{s}$. [4] makes use of the fact that $\hat{s}$ underapproximates the states reachable from $init$ in i steps. $c = \overline{s}$ in turn *overapproximates* the set of states reaching $unsafe$ in $\leq N-i$ steps. However, it is not clear how to make use of this information in PDR where we work with *underapproximations* of states reaching $unsafe$ (i.e., proof obligations) and *overaproximations* of states reachable from *init* (i.e., lemmas). In contrast, by looking at sets RR_{N-i} *as a whole*, we can extract useful information:

Theorem 1. *Given a Reverse PDR trace of length N and a PDR trace of length N'. Let s be an arbitrary cube $s \subseteq RR_{N-(i+1)}$ with $0 \leq i \leq N-2$. If we strengthen the PDR trace by blocking s in all frames $1 \leq k \leq \min(i, N')$, i.e. by setting $R_k = R_k \wedge \overline{s}$, then in the resulting PDR trace the state sets R_i with $0 \leq i \leq N$ still overapproximate the sets of states reachable from init in $\leq i$ steps. Moreover, the property of syntactical inclusion of CNFs $R_{i+1} \sqsubseteq R_i$ for $1 \leq i \leq N-1$ and semantical inclusion $R_i \subseteq R_{i+1}$ for $0 \leq i \leq N-1$ is preserved by the strengthening.*

Proof. At the end of main loop $N-1$ of Reverse PDR (after discharging all proof obligations up to frame $N-1$) it is guaranteed that there is no path of length i $(0 \leq i \leq N-2)$ from *init* to $RR_{N-(i+1)}$. Since $RR_j \subseteq RR_{j+1}$ for $0 \leq i \leq N-2$, it is also guaranteed that there is no path of length $\leq i$ $(0 \leq i \leq N-2)$ from *init* to $RR_{N-(i+1)}$. This property does not change during the next main loop, since the next main loop may only block additional cubes from $RR_{N-(i+1)}$. Thus, given a Reverse PDR trace $RR_0, RR_1, \ldots, RR_N$ and a cube $s \subseteq RR_{N-(i+1)}$ $(0 \leq i \leq N-2)$, we know that $\overline{s}$ is an overapproximation of the states which can be reached from *init* in $\leq i$ steps. Since a set R_i of PDR is an overapproximation of states reachable from *init* in $\leq i$ steps as well, $R_i \wedge \overline{s}$ overapproximates the states reachable from *init* in $\leq i$ steps.

Since an overapproximation of states reachable in $\leq i$ steps is apparently also an overapproximation of states reachable in $\leq k$ steps with $k \leq i$, $R_k \wedge \overline{s}$ overapproximates the states reachable from *init* in $\leq k$ steps.

Since for $1 \leq k \leq \min(i, N')$ all R_k are intersected with the same clause $\overline{s}$ and all R_k with $k \geq \min(i, N')$ remain unchanged, the syntactical inclusion $R_{i+1} \sqsubseteq R_i$ for $1 \leq i \leq N-1$ after strengthening is trivial, given that the same property holds before strengthening. This immediately implies semantical inclusion $R_i \subseteq R_{i+1}$ for $1 \leq i \leq N-1$. $R_0 \subseteq R_1$ follows from the fact that $R_0 = init$ remains unchanged, R_1 is replaced by $R_1 \wedge \overline{s}$, and $\overline{s}$ contains *init* as an overapproximation of states reachable from *init* in $\leq i$ steps. □

It is important to note that the strengthening of SAT solver queries to SAT?$[\neg s \wedge R_i \wedge T \wedge s']$ for some time frame i and a proof obligation state s as introduced in [1] remains sound also when we strengthen the PDR trace according to Thrm. 1. The proof that adding $\neg s$ to the query is sound [2] basically relies on two facts: s does not intersect the initial states and $R_i \subseteq R_{i+1}$ for $0 \leq i \leq N-1$. The first condition is checked before the query (otherwise we have found a counterexample) and the second condition is preserved as shown by Thrm. 1.

B. Cube extraction from PDR clause sets

To strengthen a PDR trace according to Thrm. 1 we have to extract subcubes from $RR_{N-(i+1)}$ provided by Reverse PDR. $RR_{N-(i+1)}$ is given as a CNF, thus extracting *all* subcubes amounts to a CNF to DNF conversion, and extracting a restricted number of good, i.e. short, subcubes means computing only a part of the corresponding DNF. The naive way of CNF to DNF conversion using the law of distributivity can lead to an exponential growth. Another possibility is to negate the CNF, use Plaisted-Greenbaum-Transformation [13] for translating the DNF into CNF, and negate the result again. However, this method may have disadvantages as well: The number of computed cubes is linear in the size of the CNF, but we may be interested in even more condensed information to be transferred. Moreover, we have to introduce new auxiliary variables which act as additional state space variables.

Here we present two methods we developed for under-approximating a CNF C by picking subcubes. Consider a CNF with n clauses c_i, $i \in \{1, \ldots, n\}$. Now all combinations of literals when selecting one literal from each clause c_i represents one subcube of C. A subset of these cubes is an under-approximation of C and we intend to find such a subset with short (in terms of literals) informative subcubes which is big enough to get a rather accurate under-approximation, but is still computationally manageable in SAT solver calls during PDR reasoning.

1) SAT-based approach: Here we obtain the subcubes by using a SAT solver: A satisfying assignment A of a SAT solver call for C yields exactly one cube. This cube usually can be minimized. This is done by scanning the clauses in C and greedily choosing the literals which cover most clauses. This is done until all clauses are covered. The technique is similar to the one presented in [14] and is sketched in Alg. 1. In

```
1  resultDNF := $\emptyset$; ;
2  $\forall 1 \leq i \leq n$ satSolver.$addClause(c_i)$;
3  A := satSolver.$solve()$;
4  while cube.size() < climit && resultDNF.size() < nlimit && $A \neq \emptyset$ do
5      uncovered := $\{c_1, \ldots, c_n\}$;
6      cube := $\emptyset$;
7      while uncovered $\neq \emptyset$ do
8          mostFrequLit := $\emptyset$;
9          for clause in uncovered do
10             for literal in clause do
11                 if A.contains(literal) then
12                     literal.occurences.$add$(clause);
13                     if literal.occurences.size() >
                       mostFrequLit.occurences.size() then
14                         mostFrequLit := literal;
15         cube.$add$(mostFrequLit);
16         uncovered := uncovered \ mostFrequLit.occurences;
17     resultDNF.$add$(cube);
18     satSolver.$addClause$($\neg$ cube);
19     A := satSolver.$solve()$;
```

Algorithm 1: *findCube*: SAT-based cube extraction.

line 14 of Alg. 1 we determine the literal from our satisfying assignment A which occurs most frequently in C, i.e. which covers most clauses. We add the most frequently occurring literal to our current cube (see line 15) and repeat this for the remaining uncovered clauses (see line 16) until the set of uncovered clauses is empty. After we have completed one cube we add its complement as a blocking clause (see line 18), find the next satisfying assignment and repeat the whole procedure until *climit* or *nlimit* are reached or there do not remain any new cubes to be extracted.

2) Randomized selection: The other variant uses a *simulated annealing*-like [15] approach[3] which basically also computes short cubes by greedily choosing the most frequently occurring literals. However it does not use a SAT solver and blocking clauses, but avoids always producing the same

[3]In fact we use an inverse approach here with 'heating' instead of annealing.

cube by utilizing a probability distribution. If there are N occurences of literals in clauses, the probability to choose literal l amounts to $(\frac{occurences(l)}{N})^t$ with a constant t (in our implementation t is initialized by 2). During *heating* (in contrast to annealing) steps, 'chaos is increased' by reducing t, which means that more frequently occurring literals are chosen with relatively smaller probability than before. This leads to a diversification of the chosen cubes. Due to lack of space, we omit more details here. In Sec. IV we provide a detailed analysis of how the two methods perform in practice.

C. Discharging proof obligations

After main loop N, Reverse PDR arrives at a *safe* trace of length N which implies that there are no counterexamples of length less or equal to N. This alone can be precious information when we consider a combination of PDR and Reverse PDR.

Theorem 2. *We consider a safe Reverse PDR trace of length N and a PDR trace of length N'. Then we can consider all proof obligations (s, k, d) with $k + d \leq N$ in the PDR trace to be discharged without affecting the result of PDR.*

Proof. A proof obligation (s, k, d) says that s reaches $unsafe$ within d steps and that s is present in frame k. If we could prove that s is reachable from *init* in frame k, i.e., there would be a trace of length $\leq k$ from *init* to s, then there would be a path of length $\leq k+d$ from *init* to *unsafe*. This contradicts the fact that there are no counterexamples of length less or equal to $N \geq k+d$ due to the safe Reverse PDR trace. Thus, we can consider the proof obligation (s, k, d) to be discharged. □

D. Delta-encoded trace and termination

There is one subtle, but important implementation detail of usual PDR implementations which is affected by strengthening a PDR trace according to Sect. III-A. If not taken into account properly, this detail could lead to erroneous termination detections when using so–called delta-encoded traces [2]. To be able to explain this problem, we need to have a closer look at termination checking in PDR. Termination checking is done during a propagation phase which takes place immediately before starting a new main loop $N + 1$ (and after adding a new empty frame $R_{N+1} = \emptyset$). Starting with $k = 2$ and ending with $k = N + 1$, for each clause $\neg s$ in R_{k-1} PDR checks by *SAT?*$[R_{k-1} \wedge T \wedge s']$ whether the cube s can also be blocked in R_k, i.e., whether $\neg s$ can also be added to R_k. After this, by construction, for $k > 1$ R_{k-1} is logically equivalent to R_k iff they have become syntactically equal. If there is a pair of state sets R_{k-1} and R_k, where R_{k-1} and R_k are equal, then R_k is an inductive invariant and the proof that there is no trace from *init* to *unsafe* is complete.

A delta-encoded trace [2] $R_0, \Delta R_1, \ldots, \Delta R_N$ stores each learned clause (corresponding to the negation of a blocked cube) only in the highest time frame in which it holds, i.e., we obtain the original trace from the delta encoded one by $R_0, \cup_{j=1}^{N} \Delta R_j, \ldots, \cup_{j=N-1}^{N} \Delta R_j, \Delta R_N$. This reduces the effort of termination checking: Suppose that there is a time frame ΔR_{k-1} with an empty cube set after the propagation phase, then this exactly means that $R_k = R_{k-1}$. Now we consider two types of clauses after the propagation phase: (1) For all clauses $\overline{s}$ moved from ΔR_{k-1} to ΔR_k during the propagation phase we have unsatisfiability of *SAT?*$[R_{k-1} \wedge T \wedge s']$. (2) For all clauses $\overline{\hat{s}}$ from ΔR_i with $i \geq k$ we have unsatisfiability of *SAT?*$[R_{i-1} \wedge T \wedge \hat{s}']$, since otherwise the cube $\hat{s}$ would not be blocked in frame i, and therefore we also have unsatisfiability of *SAT?*$[R_{k-1} \wedge T \wedge \hat{s}']$ due to $R_{k-1} \subseteq R_{i-1}$ for $i \geq k$. Altogether, for all clauses $\overline{s} \in R_k$ after the propagation phase we have unsatisfiability of *SAT?*$[R_{k-1} \wedge T \wedge s']$. This means that the image of R_{k-1} is included in R_k and because of $R_{k-1} = R_k$, *init* $\subseteq R_{k-1}$, and $R_{k-1} \cap$ *unsafe* $= \emptyset$, R_{k-1} is a safe inductive invariant.

This approach may go wrong, if the delta-encoded trace contains clauses added by information transfer from Reverse PDR. The argument for clauses in R_k of type (1) is the same, but if a clause $\overline{\hat{s}}$ from ΔR_i with $i \geq k$ has been obtained by information transfer from Reverse PDR, then *SAT?*$[R_{i-1} \wedge T \wedge \hat{s}']$ is not necessarily unsatisfiable. By Thrms. 1 and 2 we know that inserting $\hat{s}$ as a 'proof obligation' would discharge it after a series of recursive steps, but this is not done in our approach, since we want to learn from Reverve PDR without proving learnt information once again. However, for this reason, it is possible that the image of R_{k-1} is *not* included in R_k, although $\Delta R_{k-1} = \emptyset$ after the propagation phase. Thus, we cannot conclude termination in this situation. To be safe in the termination check, we either have to check for *all* clauses $\overline{s}$ in R_{k-1}, whether they can be propagated via SAT?$[R_{k-1} \wedge T \wedge s]$ or we have to check termination only in frames higher than the highest frame which has been strengthened by Reverse PDR information. In our implementation we choose the second option.

E. Combined Algorithm

In Alg. 2 we present our algorithm combining the original (or forward) PDR with the Reverse (or backward) PDR. Alg. 2 extends an algorithm from [4] by the collaboration methods presented in this paper. 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 10 of Alg. 2), the combined algorithm finishes with this result.

```
1  $initPDR(init, unsafe)$; $initRPDR(init, unsafe)$;
2  po$^{fw}$ := $\emptyset$; po$^{bw}$ := $\emptyset$;
3  **while** $true$ **do**
4      (res, newPo$^{fw}$) := $resumePDR$(po$^{bw}$, tlimit$^{fw}$);
5      **if** $res \neq unknown$ **then return** $res$ po$^{fw}$ := po$^{fw}\cup$ newPo$^{fw}$;
       $compress$(po$^{fw}$, slimit$^{fw}$);
6      $expandRPDR()$;
7      $strengthenRPDR()$;
8      $propagate()$;
9      (res, newPo$^{bw}$) := $resumeRPDR$(po$^{fw}$, tlimit$^{bw}$);
10     **if** $res \neq unknown$ **then return** $res$ po$^{bw}$ := po$^{bw}\cup$ newPo$^{bw}$;
       $compress$(po$^{bw}$, slimit$^{bw}$);
11     $expandPDR()$;
12     $strengthenPDR()$;
13     $propagate()$;
```

Algorithm 2: *fbPDR*.

One part of the information exchange between the two directions of PDR takes place on the basis of proof obligations po^{fw} and po^{bw} which are added to $unsafe$ and $init$, resp., as a "target enlargement" [4], see lines 5, 9, 10, 4.

Our focus though will be on the lines 6, 7 and 11, 12. For PDR traces of length N and Reverse PDR traces of length N' expanding the traces in lines 6 and 11 opens new empty time frames until the length $\max(N, N')$ has been reached.

Here we make use of the (exchanged) information that no counterexample of length $< \max(N, N')$ exists. Moreover, during expansion, proof obligations are discharged according to Sect. III-C and added as blocked cubes. After expanding, all frames including the new frames are strengthened by new learnt lemmas according to Sects. III-A and III-B. After adding new frames and strengthening, we call the standard PDR cube propagation procedure $propagate()$ (lines 8 and 13), but without integrated termination checking, see Sect. III-D. Afterwards, Alg. 2 calls standard (time-limited) implementations of PDR (line 4) resp. Reverse PDR (line 9) (including the usual cube propagation and termination check).

IV. Experimental Results

Our implementation of PDR/Reverse PDR (called *fbPDR*) is derived from [2] and also uses features of Bradley's original algorithm and newer developments, for instance, the better generalization scheme from [6]. Our Reverse PDR implementation uses the proof-obligation generalization scheme presented in [4]. The transition relation is represented as a Tseitin-transformed CNF [16], preprocessed with variable elimination. We use one MiniSat v2.2.0 [17] instance per time frame. All experiments have been run with the same resource constraints (7 GB, 3600 s) and the complete benchmark set of HWMCC'15 (excluding the access restricted Intel benchmarks) and '17.[4] We have used one core of an Intel Xeon CPU E5-2650v2 with 2.6 GHz. As proposed in [6] 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 the combined approach resp. 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, ABC's PDR[6] as well as the combined PDR/Reverse PDR implementation from [4].

a) Configuration of fbPDR: The configuration of *fbPDR* we have chosen for our experiments uses limits $nlimit$ = #CNFclauses/2 and $climit$ = #latches/20 if #latches $<$ 500 and $climit$ = #latches/100 otherwise for Alg. 1. #CNFclauses denotes the number of clauses which have to be translated into DNF. Further we have imposed a timeout for CNF to DNF conversion for each strengthened time frame which amounts to $5s$ divided by the total amount of frames to be strengthened. As suggested in [4] we use for Alg. 2 time limits $tlimit^{fw} = tlimit^{bw} = 30\ s$ for continuous executions of PDR / Reverse PDR and upper bounds for exchanged proof obligations $slimit^{bw} = 500$ and $slimit^{fw} = 500$.

b) fbPDR vs. portfolio without collaboration: To evaluate the effect of the intertwining of PDR and Reverse PDR using the methods from Sect. III we compare our combined approach with a portfolio-like approach, which also executes PDR and Reverse PDR in an alternating manner but does not share any information between the two at all. In Fig. 1 we present the results. The points above the diagonal are those in favor of *fbPDR*. Apparently sharing information represented by learnt lemmas and exchanging proof obligations has great effects on the efficiency of a combined PDR approach, there is a significant amount of benchmarks which could not be solved by the portfolio-like algorithm but by *fbPDR*. On the other hand there are only very few benchmarks on which the portfolio-like approach is the only one to solve it with an execution time very close to the timeout of 3600 s.

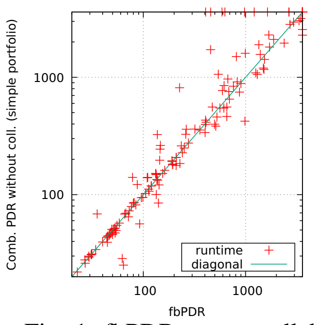


Fig. 1: fbPDR vs. no collab.

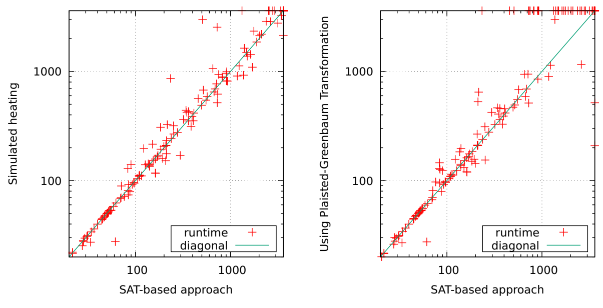
When looking into benchmark *6s407rb296*, e.g., (solved by the original PDR part), we observe, that *fbPDR* terminates after $455.8s$ with 913 learnt clauses and 16 main iterations (time frames). The portfolio-like approach does not terminate within the time limit reaching 21 main iterations in its original PDR part (with 4996 learnt clauses) and only 9 main iterations in its Reverse PDR part (with 9 learnt clauses). By omitting enlarging $init$ / $unsafe$ as in [4] we observe that the result of *fbPDR* has been obtained by a combination of techniques from [4] and from this work. Without enlarging $init$ / $unsafe$ the run times are higher, but the instance is still solved within the time limit. Then, *fbPDR* terminates after $2516.7s$ with 3670 learnt clauses and 21 main iterations. The original PDR part profits from Reverse PDR by strengthening nearly all of its frames with small sets of 2 to 15 (in later stages) short clauses of length 1 to 3.

Another interesting problem instance is *beemskbn2f1*. The implementation using only the techniques introduced in this work requires $1399.8s$ (with 1347 learnt clauses and 85 main iterations). The combination with enlarging $init$ / $unsafe$ pays off here as well, leading to a total execution time of $628.1s$ (with 2848 learnt clauses and 46 main iterations). At the first sight, it seems peculiar that the latter requires more learnt clauses but less main iterations, but this could be due to the fact that enlarging $init$ / $unsafe$ may increase the size (in terms of literals) of learnt clauses, since it prevents PDR from generalizing blocked cubes into reachable states (see [4] for details). Hence one requires more clauses to prove a time frame to be safe, but these clauses are of better 'quality'-meaning that it is more likely that they are part of a safe inductive invariant. However, the portfolio-like approach does not terminate within the time limit (reaching 4397 / 19579 learnt clauses and 48 / 26 main iterations in its original PDR / Reverse PDR part).

Even though the effects are not as significant as for the original PDR part, also the Reverse PDR part profits from being strengthened by original PDR. When considering the benchmark instance *bobtuttt*, the Reverse PDR part of *fbPDR* solves it in $212.4s$, with 837 learnt clauses and 11 main iterations. The Reverse PDR part is expanded from frame 3 to 9 and strengthened by 20 to 40 (rather long) clauses of length 60 to 65. The portfolio-like algorithm requires $1061.33s$, with 7 main iterations and 4669 learnt clauses.

c) Comparison of different variants of fbPDR: In Fig. 2 we compare different variants of extracting cubes for strengthening traces as presented in Sect. III-B: the SAT–based (partial) CNF-to-DNF conversion used in *fbPDR*, a complete CNF-

[4]http://fmv.jku.at/hwmcc15/ resp. http://fmv.jku.at/hwmcc17/
[5]https://github.com/arbrad/IC3ref, downloaded in Sep. 2016.
[6]https://bitbucket.org/alanmi/abc, downl. on 9/10/2017, standard config.



(a) vs. 'simulated heating' (b) vs. Plaisted-Greenb. Trans.

Fig. 2: Comparison of CNF to DNF conversions.

to-DNF conversion using Plaisted-Greenbaum transformation, and a (partial) CNF-to-DNF conversion using randomized literal selection. In Fig. 2a the overall run times using the SAT-based subcube extraction technique appear to be slightly better than the ones when using randomized literal selection (for both variants we used identical constraints on numbers and sizes of cubes as given in this section). On the other hand, Fig. 2b clearly shows that the complete CNF-to-DNF conversion using Plaisted-Greenbaum transformation leads to much inferior results. The reason lies in more expensive SAT solver calls after strengthening with large numbers of lemmas resulting from a complete CNF-to-DNF conversion as well as in the fact that the auxiliary variables newly introduced by Plaisted-Greenbaum transformation are only locally known in single time frames and thus impede cube propagation and CNF simplifications with syntactic subsumption checks. Altogether we can conclude that *fbPDR* performs best using a SAT-based partial subcube extraction technique while imposing rather tight bounds on length and amount of blocked cubes which are shared between PDR and Reverse PDR. In the next experiment we evaluated a variant of *fbPDR* which uses trace expansions and discharging of proof obligation based on information learned from PDR / Reverse PDR, but omits strengthening of traces according to Sect. III-A. The results shown in Fig. 3 clearly underline the importance of strengthening PDR (resp. Reverse PDR) traces with information learnt from Reverse PDR (resp. PDR).

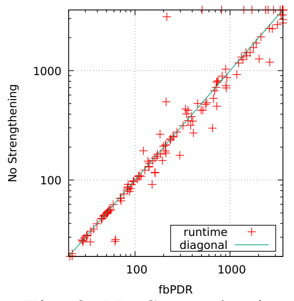


Fig. 3: No Strengthening

d) fbPDR *versus other implementations:* In Fig. 4 we compare *fbPDR* against ABC's PDR, Aaron Bradley's IC3 reference implementation (ic3ref), and the implementation *Comb. PDR* using combined PDR / Reverse PDR provided by [4]. Moreover, we add a variant *w/o enl. init/unsafe* only implementing the collaboration techniques discussed in this work, without enlarging *init* / *unsafe* by shared proof-obligations [4].[7] Apparently *fbPDR* greatly outperforms the implementation from [4] as well as state-of-the-art implementations of pure PDR. This is already true for a version which does not use the ideas from [4] (see *w/o enl. init/unsafe*). The results for *fbPDR* show in addition that the methods presented in this

[7]We provide result tables and binaries under https://www.dropbox.com/s/ckbzq6kd10aebod/fbPDR.zip?dl=0.

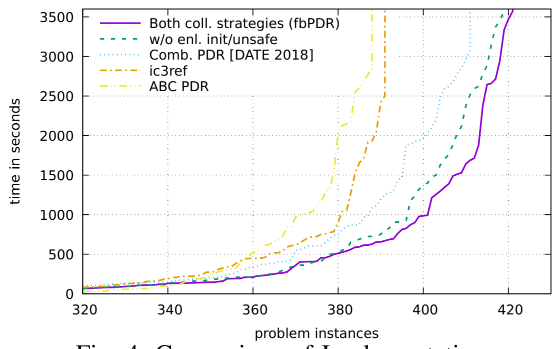


Fig. 4: Comparison of Implementations

paper and sharing proof-obligations are compatible and lead to even better results when used in combination.

V. Conclusions and Future Work

We looked into the combination of PDR and Reverse PDR and presented a sound method that profits from information represented by lemmas of one PDR direction for the work of the other direction. In doing so, we efficiently extract underapproximating subcubes from clause sets of one PDR direction for strengthening the other one. The method is successfully combined with exchanging of proof obligations which has been presented before. Altogether we achieved a very well competing and strongly interweaved combination of a PDR and a Reverse PDR engine, sharing all of their gathered informations. For the future we would like to look into a more adaptive way to distribute the run time between PDR and Reverse PDR. Furthermore, we consider it to be a promising field of research to examine even more advanced cooperation schemes between original and Reverse PDR.

References

[1] A. R. Bradley, "Sat-based model checking without unrolling," in *VMCAI*, 2011, pp. 70–87.
[2] N. Eén, A. Mishchenko, and R. K. Brayton, "Efficient implementation of property directed reachability," in *FMCAD*, 2011, pp. 125–134.
[3] T. Seufert and C. Scholl, "Sequential verification using Reverse PDR," in *MBMV*, 2017, pp. 79–89.
[4] ——, "Combining PDR and reverse PDR for hardware model checking," in *DATE*, 2018, pp. 49–54.
[5] A. Ivrii and A. Gurfinkel, "Pushing to the top," in *FMCAD*, 2015, pp. 65–72.
[6] Z. Hassan, A. R. Bradley, and F. Somenzi, "Better generalization in IC3," in *FMCAD*, 2013, pp. 157–164.
[7] G. Cabodi, S. Nocco, and S. Quer, "Mixing forward and backward traversals in guided-prioritized BDD-based verification," in *CAV*, 2002, pp. 471–484.
[8] C. Stangier and T. Sidle, "Invariant checking combining forward and backward traversal," in *FMCAD*, 2004, pp. 414–429.
[9] Y. Vizel, O. Grumberg, and S. Shoham, "Intertwined forward-backward reachability analysis using interpolants," in *TACAS*, 2013, pp. 308–323.
[10] S. Chaki and D. Karimi, "Model checking with multi-threaded ic3 portfolios," in *VMCAI*, 2016, pp. 517–535.
[11] M. Marescotti, A. Gurfinkel, A. E. J. Hyvärinen, and N. Sharygina, "Designing parallel pdr," in *FMCAD*, 2017, pp. 156–163.
[12] E. Goldberg, M. Güdemann, D. Kroening, and R. Mukherjee, "Efficient verification of multi-property designs (the benefit of wrong assumptions)," in *DATE*, 2018, pp. 43–48.
[13] D. A. Plaisted and S. Greenbaum, "A structure-preserving clause form translation," in *Journal of Symbolic Computation*, 1986, pp. 293–304.
[14] K. Ravi and F. Somenzi, "Minimal assignments for bounded model checking," in *TACAS*, 2004, pp. 31–45.
[15] S. Kirkpatrick, D. G. Jr., and M. P. Vecchi, "Optimization by simmulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
[16] G. Tseitin, "On the complexity of derivations in propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logics*, 1968.
[17] N. Eén and N. Sörensson, "An extensible SAT-solver," in *SAT*, 2003, pp. 502–518.