Sequential Verification Using Reverse PDR

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

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

Abstract. In the last few years IC3 resp. PDR made a great stir 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 observed in BDD based model checking, we consider *Reverse* PDR which starts its analysis with the initial states instead of the unsafe states as in original PDR. We show great benefits of Reverse PDR both by a theoretical and an experimental analysis. Finally, we profit from parallelism offered by modern multi-core processors and use a portfolio approach combining the advantages of Reverse and original PDR.

1. 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 2003 [8], has been considered the strongest amongst these for a long time. In 2011 though PDR resp. IC3 – as called in its first implementation [2] – 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 [5].

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. 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 investigate (both theoretically and experimentally) also the other direction: We consider overapproximations of state sets from which we can reach the unsafe states in k or less steps and the overapproximations are guided by the initial states. We call the method turning around the original direction of PDR '*Reverse PDR*'. In detail we make the following contributions:

- We give a proof showing an exponential complexity gap between PDR and Reverse PDR (and vice versa).
- We analyze the optimization potential for Reverse PDR, in particular we present a new method for generalizing proof obligations in Reverse PDR.
- We make an experimental evaluation comparing prototypes of the original and Reverse PDR empirically.
- The evaluation includes a combination of PDR and Reverse PDR in a portfolio approach.

Related work Since the introduction of PDR, there have been several improvements on efficiency [4, 6]. Besides that, PDR has been lifted to the theory of reals using a SMT-solver [7] and has been transferred to other applications like automated planning [9] and software verification [3]. Interestingly, [9] uses a similar idea of reverting PDR in the context of automated planning with the insight, that reverting the direction generally yields better results in finding plans resp. counterexamples in this special application context.

In Sect. 2 we give some preliminaries needed for this paper. Then we present a theoretical analysis and optimizations of Reverse PDR in Sect. 3. An experimental evaluation is given in Sect. 4 and Sect. 5 summarizes the results.

2. Preliminaries

2.1. 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 [10]. Modern SAT solvers are able to check the satisfiability of boolean formulas in CNF.

2.2. Property Directed Reachability

The method we will further use and adapt is called property directed reachability (PDR) [4] or IC3 as in its original implementation [2]. PDR produces stepwise reachability information in time-frames without unrolling the transition relation. Each time-frame k corresponds to a set of clauses R_k. R_k is the negation of the disjunction over all cubes which have already been proven unreachable from the initial states in up to k steps, i.e., R_k is an overapproximation of the states reachable from the initial states in k steps or less. R_0 plays a special role and always represents the initial states. In round N of the main loop, we have $N + 1$ time frames $R_0, \ldots, R_N$. In each main loop the sets R_k are refined step by step using so-called 'proof obligations'. The proof obligations are produced and discharged by SAT checks: A SAT solver call $SAT?[R_N \wedge unsafe]$ extracts unsafe cubes s and produces proof obligations (s, N). The proof obligation (s, k) means that we have to prove that s can not be reached from *init* in up to k steps. For discharging a proof obligation (s, k) a second type of SAT check $SAT?[R_{k-1} \wedge T \wedge s']$ is used (s' results from s by replacing current state variables by next state variables). If this SAT problem is unsatisfiable, the proof obligation is discharged and the clause $\neg s$ may be added to R_k, otherwise a satisfying assignment gives us a cube $\hat{s}$, leading to a new proof obligation $(\hat{s}, k - 1)$. When all proof obligations have been discharged and no further unsafe cube can be extracted from $R_N \wedge unsafe$, the proof that unsafe states can not be reached within N steps is complete. Then a new time frame R_{N+1}, initialized by the empty clause set, is added and a new iteration of the main loop starts. The algorithm ends when we have to insert a proof obligation whose cube intersects *init* (in this case a trace from *init* to *unsafe* has been found) or when two sets R_{k-1} and R_k become equivalent (in this case R_{k-1} is an inductive invariant and the property holds).

3. Reverse PDR

3.1. Basic Approach

Reverse PDR starts its analysis with the initial states *init* instead of the unsafe states *unsafe*. Instead of pre-image computations as in original PDR, it uses image computations for cubes of states. Reverse PDR computes overapproximations $RR_0, \ldots, RR_N$.[1] By definition, RR_0 always represents the *unsafe* states. RR_k always overapproximates the set of states from which *unsafe* can be reached in up to k steps. This invariant holds, since in the first main loop RR_0 is initialized by *unsafe*, RR_1 is initialized by 1 (representing the set of all possible states), and in later steps we only exclude states s from RR_k, if there is provably no transition from s into RR_{k-1} (which overapproximates – by induction assumption – the set of states from which *unsafe* can be reached in up to $k - 1$ steps).

Let us consider main loop N when Reverse PDR has constructed time frames $RR_0, \ldots, RR_N$. Reverse PDR starts by extracting satisfying cubes from $RR_N \wedge init$ by calling a SAT solver. It generalizes a satisfying assignment of $RR_N \wedge init$ into a cube s of initial states for which it has not yet been proven that *unsafe* can not be reached from them in up to N steps (see Sect. 3.2.2). A new

[1] As sets of clauses, RR_k are predicates. In the following we often identify those predicates with the state sets represented by them.

proof obligation (s, N) is inserted. It has to be proven that it is not possible to reach *unsafe* in up to N steps from the states in s. Discharging this proof obligation works similar to original PDR: We start by asking for *SAT?*$[s \wedge T \wedge RR'_{N-1}]$ (which corresponds to an image computation) where the clauses in RR'_{N-1} are formulated with next state variables instead of current state variables. If this SAT check is unsatisfiable, then the proof obligation has been discharged (as in original PDR) and s is excluded from RR_N by adding $\neg s$ to the clauses in RR_N. If possible, s is generalized before, see Sect. 3.2.1. If the SAT check is satisfiable, a successor minterm m is extracted from the satisfying assignment, m is 'generalized' 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?*$[\hat{s} \wedge T \wedge RR'_{N-2}]$ or may produce new recursive proof obligations at earlier time frames. The idea of generalizing proof obligations is to combine the proof obligation with others that would occur later on anyway; more details will follow in Sect. 3.2.2. Before we add a new proof obligation, we always check whether its cube intersects *unsafe*. If yes, then we stop, 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 all states covered by $\tilde{s}$ can be reached from *init*).

As in original PDR we keep the invariant that for all k the clause set RR_k is even *syntactically* included in RR_{k-1} (and thus the state set represented by RR_{k-1} is a subset of the state set represented by RR_k). The invariant holds in the beginning of the algorithm. If *SAT?*$[s \wedge T \wedge RR'_{k-1}]$ 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 RR_{i-1} is a subset of RR_{k-1}). Therefore we can exclude s (by adding $\neg s$ to the clause set) from RR_i with $1 \leq i \leq k-1$ as well. The state set represented by RR_0 remains always equal to *unsafe*. It is a subset of RR_k $(1 \leq k \leq N)$, since no proof obligations (s, k) with s intersecting *unsafe* are generated and thus it never happens that an unsafe state is excluded from RR_k.

When all proof obligations are discharged and $RR_N \wedge init$ becomes unsatisfiable, a new time frame RR_{N+1} with $RR_{N+1} = 1$ (represented by the empty clause set) is added as in original PDR, and a new main loop is started.

Immediately before starting a new main loop, a propagation phase takes place: Starting with $k = 2$ and ending with $k = N + 1$, for each clause $\neg s$ in RR_{k-1} we check by *SAT?*$[s \wedge T \wedge RR'_{k-1}]$ whether the cube s can also be blocked in RR_k, i.e., whether $\neg s$ can also be added to RR_k. After this, by construction, RR_{k-1} is logically equivalent to RR_k iff they have become syntactically equal. If RR_{k-1} and RR_k are equal, then the proof that there is no trace from *init* to *unsafe* is complete. This follows from the following facts: (1) The preimage of RR_{i-1} is always included in RR_i, since the only possibility for a cube s to be excluded from RR_i is unsatisfiability of $s \wedge T \wedge RR'_{i-1}$. Since $RR_{k-1} = RR_k$, the preimage of RR_{k-1} is included in RR_{k-1}. (2) Since *unsafe* is included in the state set RR_{k-1}, there is no backward trace starting from *unsafe* to a state outside RR_{k-1}. (3) Since after a main loop the last time frame does not intersect with *init* any more and since in the course of the algorithm only cubes are *removed* from the sets RR_i, RR_{k-1} does not contain initial states. Thus, there is no trace from *init* to *unsafe*.

We will have a closer look at some specific optimizations for Reverse PDR in the next section. Other technical details are directly inherited from original PDR, e.g.: (a) The clause sets for RR_k can be optimized by removing subsumed clauses. (b) When inserting the proof obligation (s, k), we know that all states in the cube s can be reached from *init*. Therefore we can insert also proof obligations (s, l) with $k < l \leq N$, since traces from s to *unsafe* of lengths larger than k should also be excluded, if the property holds. Thus, instead of recursive calls for proof obligations we handle a queue of proof obligations and dequeue proof obligations in smaller time frames first.

3.2. Optimizations

3.2.1. Generalizing Cubes in Unsatisfiable Cases

If a SAT problem $SAT?[s \wedge T \wedge RR'_{k-1}]$ is unsatisfiable, then we can try to derive a *sub*cube $\hat{s}$ of s that is sufficient to make the SAT problem unsatisfiable. If we can find such a subcube $\hat{s}$ of s, we can block the larger subcube $\hat{s}$ in RR_k (i.e., add $\neg\hat{s}$ to RR_k). The generalization works based on unsatisfiable cores or on final conflict clauses in SAT solvers where s is specified by assumptions [5]. Furthermore, s can be minimized by a series of SAT checks [4]. The generalization method for unsatisfiable cases is basically the same as in original PDR.

3.2.2. Generalizing Cubes in Satisfiable Cases

$SAT?[RR_N \wedge init]$: If $SAT?[RR_N \wedge init]$ is satisfiable, the SAT solver returns a satisfying minterm m of $RR_N \wedge init$. Using ternary simulation on a circuit representation of $RR_N \wedge init$, m can be reduced to a subcube $\hat{s}$ containing satisfying assignments only, leading to a generalized proof obligation as in original PDR [4].

$SAT?[s \wedge T \wedge RR'_{k-1}]$: 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 can be generalized to a satisfying subcube $\hat{s}$ by ternary simulation as well [4]. The ternary simulation relies 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. If some state variable s_i in m is replaced by the unknown value X and ternary simulation does not propagate the X-value to outputs with literals occurring in s', then s_i resp. $\neg s_i$ can be removed from m. Unfortunately, this method can not be generalized to Reverse PDR, since the circuit specifying T has a 'fixed direction' and cannot be simply 'reverted'. On the other hand, generalizing satisfiable cases has been proven to be heavily effective in [4].

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 m' can be generalized to a subcube $\hat{s}'$ amounts to the question whether $\hat{s}'$ is completely included in the image of s and in RR'_{k-1}. An *exact* solution to this question can be obtained by solving a QBF problem: Assume that the vector of primary inputs is $\vec{i} = (i_1, \ldots, i_m)$, the vector of current state variables is $\vec{s} = (s_1, \ldots, s_n)$ and the next state variables *not* occurring in $\hat{s}'$ are $s'_{i_1}, \ldots, s'_{i_l}$. Then m' can be generalized to $\hat{s}'$ iff the QBF

$$\forall s'_{i_1} \ldots \forall s'_{i_l} \exists \vec{s} \exists \vec{i} : s \wedge T \wedge RR'_{k-1} \wedge \hat{s}' \tag{1}$$

is satisfiable. However, 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 $\hat{s}'$ seem to cause too much overhead. For that reason we use a rough approximation which nevertheless helps in practice as experimental results in Sect. 4 show.

The subcube $\hat{s}'$ has to fulfill the following conditions: (1) In the part of the image of s where the next state variables in $\{s'_1, \ldots, s'_n\} \backslash \{s'_{i_1} \ldots s'_{i_l}\}$ have exactly the same valuations as in m', all possible combinations from $\mathbb{B}^l$ occur for the values of $s'_{i_1} \ldots s'_{i_l}$. (2) The subcube $\hat{s}'$ is completely contained in RR'_{k-1}. A sufficient condition (1') for fulfilling condition (1) is: The transition functions δ_{i_j} computing s'_{i_j} all have (structural) support sets that are disjoint from the (structural) support sets

of all other transition functions and their support sets do not contain any variable occurring in the cube s. Moreover, these transition functions δ_{i_j} are different from the constant functions 0 and 1. The idea of condition (1') 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 (1') for the support sets we can change the variables in the support sets of δ_{i_j} computing s'_{i_j} arbitrarily without changing $\hat{s}'$. Since the support sets of those transition functions δ_{i_j} are in addition disjoint, we can produce arbitrary value combinations at their outputs without changing $\hat{s}'$.

Our implementation for finding subcubes of m' fulfilling condition (1') works as follows:

1. In a preprocessing step, find the set D of all non-constant transition functions δ_i whose structural support set is disjoint from the structural support set of all other transition functions.

2. If $SAT?[s \wedge T \wedge RR'_{k-1}]$ is satisfiable, then find the transition functions $\delta_i \in D$ whose structural support sets do not contain variables occurring in the cube s. Let $C = \{s'_{j_1}, \ldots, s'_{j_k}\}$ be the set of next states variables computed by those transition functions. The variables from C are candidates for being removed from the satisfying minterm m' of $SAT?[s \wedge T \wedge RR'_{k-1}]$.

Whether a variable from C can really be removed from m' in order to obtain a shorter generalized subcube $\hat{s}'$, depends on condition (2). To account for condition (2) we use ternary simulation as in the case '$SAT?[RR_N \wedge init]$ satisfiable': We consider RR'_{k-1} as a circuit and try to replace in m' variables from C by the unknown value X. As long as the X-value does not propagate to the output of RR'_{k-1}, we can remove this variable from m'. Finally, we arrive at a generalized subcube $\hat{s}'$ of m'. Note that omitting condition (2) neither leads to wrong results nor impedes termination of Reverse PDR: Without condition (2), $\hat{s}'$ may contain states which have already been blocked in time frame $k-1$, but at least one new state to be blocked (namely m'). Therefore generalizing m' to $\hat{s}'$ can not produce a larger number of proof obligations in the following steps.

3.2.3. Adding $\neg s'$ to the query

For the original PDR, Bradley [2] introduced strengthened queries $SAT?[R_{k-1} \wedge \wedge \neg s \wedge T \wedge s']$ instead of $SAT?[R_{k-1} \wedge T \wedge s']$ making the queries more likely to be unsatisfiable which improves the chances to discharge proof obligations. Here we show that for Reverse PDR a corresponding strengthening, turning $SAT?[s \wedge T \wedge RR'_{k-1}]$ into $SAT?[s \wedge T \wedge RR'_{k-1} \wedge \neg s']$, is sound as well. We argue that whenever $s \wedge T \wedge RR'_{k-1} \wedge \neg s'$ is unsatisfiable, it is not possible to reach *unsafe* from s in up to k steps, i.e., s can be removed from RR_k. Assume $s \wedge T \wedge RR'_{k-1} \wedge \neg s'$ is unsatisfiable (and therefore also $s \wedge T \wedge RR'_{i-1} \wedge \neg s'$ with $1 \leq i < k$). We differentiate between two cases:

- Case 1: $s \wedge T \wedge RR'_{k-1}$ is unsatisfiable as well. Then we have nothing to prove.
- Case 2: $s \wedge T \wedge RR'_{k-1}$ is satisfiable. Then the only states in RR_{k-1} which can reached from s lie inside s.

We continue our argumentation with $k-1$ instead of k etc.. Case 1 applies at latest when we arrive at $k = 1$, since $RR_0 = unsafe$, $unsafe \wedge s$ is unsatisfiable (otherwise we would not have added a proof obligation with s, but we would have stopped with a counterexample) and thus $s \wedge T \wedge RR'_0 \wedge \neg s' = s \wedge T \wedge RR'_0$. Altogether there is no trace of length up to k from s to *unsafe* and we can thus remove s from RR_k by adding the clause $\neg s$.

3.3. PDR and Reverse PDR

Our experiments in Sect. 4 show that it is worthwhile to consider both original and Reverse PDR, since they outperform each other on different benchmark instances. Here we show that 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.*

Proof: Consider a sequential circuit consisting of two modulo-2^n counters in parallel with state bits $s = (s_0, ..., s_{n-1})$ and $t = (t_0, ..., t_{n-1})$, respectively. Both counters read the same primary input i_1. If $i_1 = 0$, the counter values remain the same, if $i_1 = 1$, the counters increment their values (modulo 2^n). Assume that the initial states *init* are given by $s \neq t$ and the unsafe states are given by $s = t$. It is clear that the design fulfills the safety property, since the counters always count in parallel, thus from different counter values we can never reach identical counter values.

We first consider original PDR: In its first time frame, PDR starts with $R_1 = 1$ and picks satisfying assignments of $R_1 \wedge$ *unsafe* until the formula becomes unsatisfiable. Each satisfying assignment consists of a new pair of identical counter values s and t, all 2^n possible value combinations with $s = t$ have to be enumerated and blocked until $R_1 \wedge$ *unsafe* becomes unsatisfiable. No satisfying assignment can be generalized, since omitting any literal would lead to a cube with a non-empty intersection with *init* $= (s \neq t)$. Thus, already in the first time frame we have an exponential number of SAT queries.

Now we look at Reverse PDR: Reverse PDR starts in its first time frame by enumerating satisfying assignments of $RR_1 \wedge$ *init*, *init* $= (s \neq t)$. Using generalization by ternary simulation, it produces $2 \cdot n$ cubes as proof obligations in frame 1: $s_0 \wedge \neg t_0, \neg s_0 \wedge t_0, \ldots, s_{n-1} \wedge \neg t_{n-1}, \neg s_{n-1} \wedge t_{n-1}$. All $2 \cdot n$ generalized proof obligations $(c, 1)$ are immediately discharged by unsatisfiable SAT checks of type *SAT?*$[c \wedge T \wedge RR'_0]$ with $RR'_0 =$ *unsafe* $= (s = t)$, since we can not reach identical counter values from different ones. At the end, we have $RR_1 = \bigwedge_{i=0}^{n-1}((\neg s_i \vee t_i) \wedge (s_i \vee \neg t_i))$ (which is logically equivalent to *unsafe*) and no satisfying assignment remains. Then, before starting a second main loop, the frame RR_2 is initialized by 1 and Reverse PDR tries to propagate all blocked cubes (resp. all clauses) from RR_1 to RR_2. This amounts to $2 \cdot n$ SAT checks of type *SAT?*$[c \wedge T \wedge RR'_1]$. Since RR_1 is logically equivalent to *unsafe* as RR_0, all blocked cubes can be propagated, RR_2 is syntactically equal to RR_1 and Reverse PDR has completed the safety proof. Altogether, Reverse PDR needs $6 \cdot n$ SAT checks for completing the proof.[2] □

4. Experimental Results

Our implementation of original PDR is derived from [4] and augmented to support *Reverse PDR*, too. The transition relation is represented as a Tseitin-transformed CNF [10], preprocessed with variable elimination. We use one MiniSat v2.2.0 [5] instance per time frame. Note that our implementation is a prototype, not yet being able to fully compete with the IC3 implementation [2]. Currently our implementation of PDR is still by factor 3.01 slower than the IC3 reference implementation [2].

[2] If we exchange *init* and *unsafe*, the same example can prove exponentially more SAT checks for Reverse PDR than for original PDR.

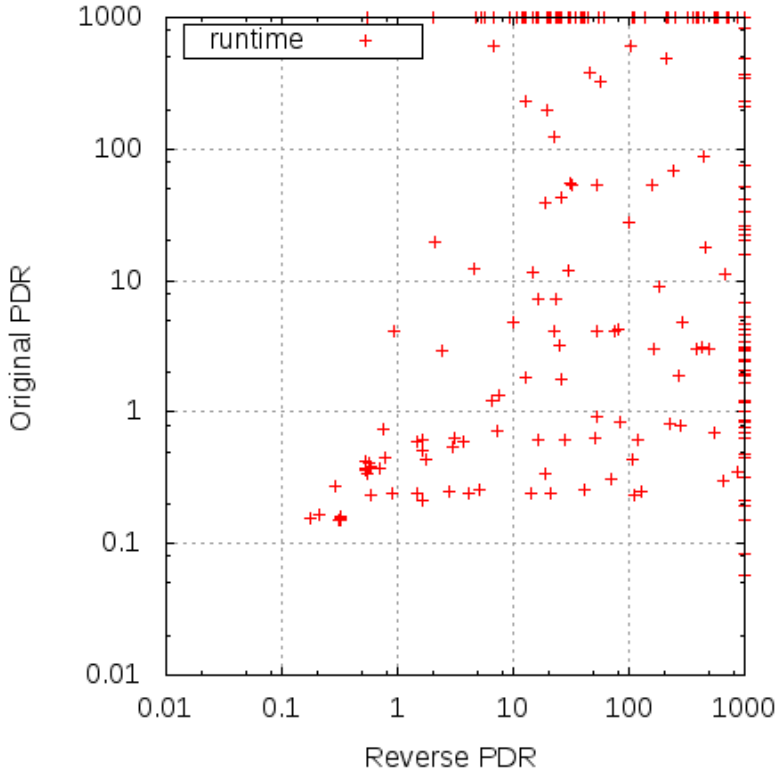


Figure 1: Comparison of Reverse PDR and original PDR on HWMCC'11 benchmarks. The x-axis shows Reverse PDR execution times, the y-axis shows original PDR execution times. The scale is logarithmic.

The Reverse PDR implementation uses the same code base as our original PDR implementation. All experiments have been run with constrained resources – 7 GB on memory and 900 s on execution time. We have used an Intel Xeon CPU E5-2643 with 3.3 GHz. All experiments have been run on HWMCC'11 benchmarks excluding the access-restricted Intel benchmarks.

4.1. Comparison of original PDR and Reverse PDR

The scatter plot in Fig. 1 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 outruns the original version by magnitude and the other way around. Although usual hardware verification approaches are confined to the original PDR, this results justifies a further consideration of both versions of PDR. Fig. 2 shows slight advantages for original PDR in overall solved benchmarks with emphasis on the earliest few seconds.

4.1.1. Using a portfolio approach

To complete the analysis of Reverse PDR's benefit we implemented a portfolio approach, meaning that we let original and Reverse PDR run concurrently and stop when the first procedure finished the benchmark. Fig. 2 displays a comparison between the portfolio approach and original PDR showing a major increase in solved benchmarks for the portfolio implementation. The mean over-

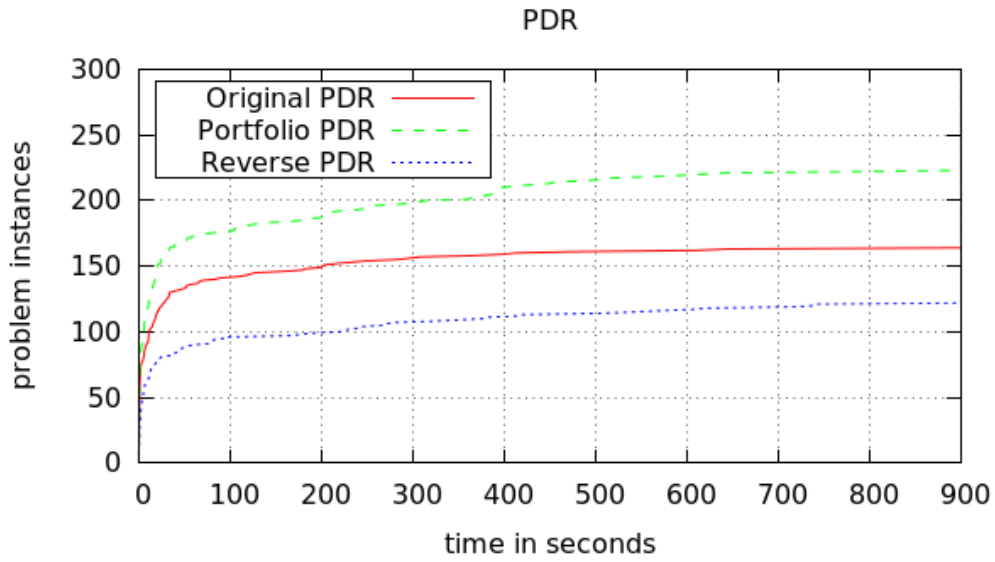


Figure 2: Comparison of Reverse PDR, original PDR, and the portfolio approach on HWMCC'11 benchmarks. The x-axis shows execution times, the y-axis displays the number of solved benchmarks.

head of the portfolio implementation – i.e. the runtime effort monitoring the concurrent execution and terminating the slower procedure – amounts to less than 0.01 %.

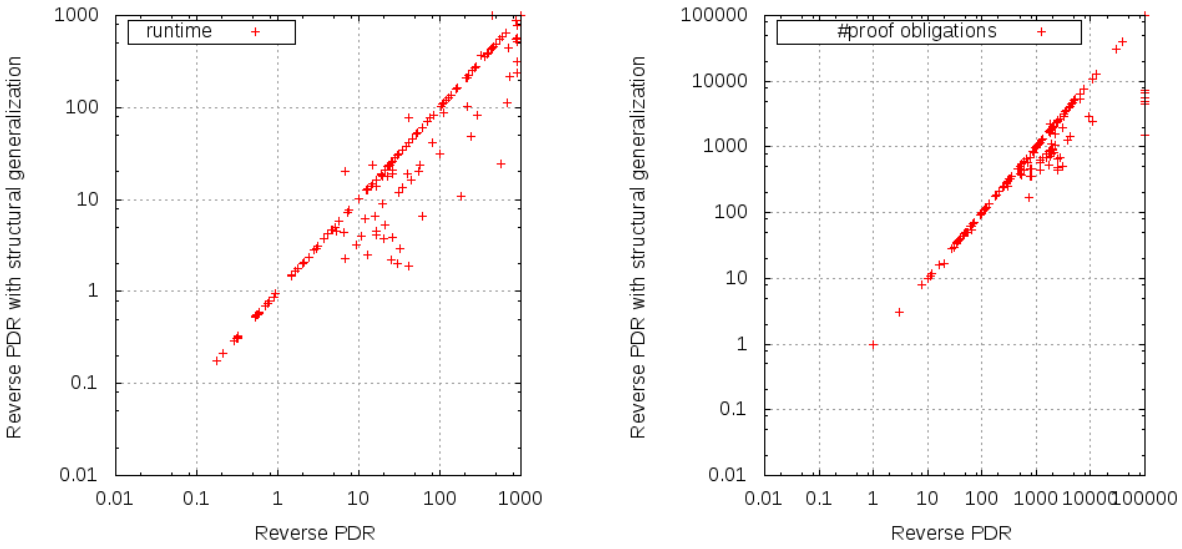
4.2. Comparison of Reverse PDR with and without structural generalization of proof obligations

In the previous experiments we used the generalization of proof obligations by the structural method according to condition (1') from Sect. 3.2.2.[3] To evaluate the effect of this structural generalization procedure we compare versions of Reverse PDR with and without the structural generalization. In Fig. 3a we present the results. Apparently, when applicable, the heuristics speeds up Reverse PDR almost every time. In very few cases though it seems that the overhead of the procedure outweighs its benefit - producing some results above the diagonal. The mean overhead of the structural optimization amounts to 0.6 % of Reverse PDR execution time. The mean overall execution time improvement using structural optimization amounts to 5.7 %.
The reason for execution time improvements lies in the ability of structural optimization to reduce the number of proof-obligations which have to be discharged. Fig. 3b compares the number of processed proof-obligations of Reverse PDR with and without additional structure-based generalization of proof obligations leaving us similar results to comparing the execution times.

5. Conclusions and Future Work

We proved a theorem showing an exponential complexity gap between original PDR and its reverted variant – Reverse PDR. We also observed this complexity gap practically in the execution of a HWMCC benchmark suite.

[3] In our implementation we omit checking condition (2), see Sect. 3.2.2.



(a) The x-axis shows Reverse PDR execution times, the y-axis shows execution times of Reverse PDR with structural generalization. The scale is logarithmic.

(b) The x-axis shows Reverse PDR processed proof-obligations, the y-axis shows processed proof-obligations of Reverse PDR with structural generalization. The scale is logarithmic.

Figure 3: Comparison of Reverse PDR with and without the structural generalization.

Overall it seems that original PDR is more powerful in the sheer number of solved benchmarks than Reverse PDR. However, considering the many benchmarks on which original PDR is vastly outperformed by Reverse PDR, the reverted variant still brings significant benefit – especially when used in a portfolio approach in combination with original PDR. We showed that there is optimization potential for Reverse PDR in the generalization of proof obligations even though ternary simulation does not apply in Reverse PDR.

Apparently there is still work to do in analyzing and optimizing Reverse PDR, leaving this a promising field of research.

References

[1] Biere, Armin, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu: *Symbolic model checking without bdds*. In *International conference on tools and algorithms for the construction and analysis of systems*, pages 193–207. Springer Berlin Heidelberg, 1999.

[2] Bradley, Aaron R.: *Sat-based model checking without unrolling*. In *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, volume 6538 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2011.

[3] Cimatti, Alessandro and Alberto Griggio: *Software model checking via IC3*. In Madhusudan, P. and Sanjit A. Seshia (editors): *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 277–293. Springer, 2012, ISBN 978-3-642-31423-0. `http://dx.doi.org/10.1007/978-3-642-31424-7_23`.

[4] Eén, Niklas, Alan Mishchenko, and Robert K. Brayton: *Efficient implementation of property directed reachability*. In *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*, pages 125–134. FMCAD Inc., 2011.

[5] Eén, Niklas and Niklas Sörensson: *An extensible sat-solver*. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, pages 502–518, 2003.

[6] Hassan, Zyad, Aaron R. Bradley, and Fabio Somenzi: *Better generalization in IC3*. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 157–164, 2013.

[7] Hoder, Krystof and Nikolaj Bjørner: *Generalized property directed reachability*. In Cimatti, Alessandro and Roberto Sebastiani (editors): *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2012, ISBN 978-3-642-31611-1. http://dx.doi.org/10.1007/978-3-642-31612-8_13.

[8] McMillan, Kenneth L.: *Interpolation and sat-based model checking*. In *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, pages 1–13, 2003.

[9] Suda, Martin: *Property directed reachability for automated planning.* J. Artif. Intell. Res.(JAIR), 50:265–319, 2014.

[10] Tseitin, G.: *On the complexity of derivations in propositional calculus*. In *Studies in Constructive Mathematics and Mathematical Logics*. 1968.