Everything You Always Wanted to Know About Generalization of Proof Obligations in PDR

Tobias Seufert[1], Felix Winterer[1], Christoph Scholl[1], Karsten Scheibler[2], Tobias Paxian[1], Bernd Becker[1]

[1]Institute of Computer Science, Albert-Ludwigs-Universität Freiburg, Germany,

{seufert, winteref, scholl, paxiant, becker}@informatik.uni-freiburg.de

[2]BTC Embedded Systems AG, Germany, scheibler@btc-es.de

Abstract—In this paper we revisit the topic of generalizing proof obligations in bit-level Property Directed Reachability (PDR). We provide a comprehensive study which (1) determines the complexity of the problem, (2) thoroughly analyzes limitations of existing methods, (3) introduces approaches to proof obligation generalization that have never been used in the context of PDR, (4) compares the strengths of different methods from a theoretical point of view, and (5) intensively evaluates the methods on various benchmarks from Hardware Model Checking as well as from AI Planning.

I. Introduction

In 2011, the verification engine PDR resp. IC3 was introduced [1] and is nowadays widely considered as the most powerful algorithm for Hardware Model Checking. Apart from Hardware Model Checking, PDR is in use on lots of different domains, such as Software Model Checking, Hybrid Systems Model Checking, or AI Planning [2]–[9]. It has been lifted to SMT on a wide range of theories. However, in this work we restrict ourselves to bit-level PDR.

The idea of PDR is to avoid the unrolling of the transition relation as in Bounded Model Checking (BMC) [10] and to rather replace small numbers of large and hard SAT problems by many small and easy ones based on a single instance of the transition relation only. PDR repeatedly strengthens a proof by removing unreachable predecessors of unsafe states. Thereby, PDR tries to avoid enumerating single states by putting a lot of effort in generalizing these predecessors to preferably expressive state sets. Firstly, PDR generalizes states which are predecessors of the unsafe states, so called *proof obligations* (POs). POs have already been proven to reach an unsafe state, and should therefore not be reachable from any initial state for the system to be safe. Furthermore, PDR also generalizes states which are proven to be unreachable from the initial states. In this paper we put our focus on the generalization of POs because PDR's efficiency relies heavily on these generalization capabilities [11], [12] and in contrast to the generalization of unreachable states [1], [13], [14], the generalization of proof obligations did not receive that much attention in research lately. Since PDR is in use on such a great variety of applications, these domains pose different challenges to the generalization of POs. Exact methods for the

generalization of POs would amount to pre-image computations requiring quantifier elimination. Common methods approximate quantifier elimination and build on transition functions instead of general transition relations. However, we stress that this is insufficient for a great deal of problem domains.

We discuss exact and approximative generalization techniques for circuits, reverted circuits, circuits with invariant constraints and general transition relations. Our contribution is as follows:

1) We show that generalizing POs in PDR is Π_2^P-complete in general. Thus, a non-approximative solution will always have the same complexity as a 2-QBF problem.

2) We investigate generalization techniques for sequential circuits (i.e., transition *functions*) which have not been used in the context of PDR to the best of our knowledge and we give a thorough analysis of the detailed reasons why known techniques do not work for general transition relations.

3) We discuss which methods are applicable to circuits with invariant constraints and which transformations can be used to enable the correct application of all generalization techniques that are known for circuits.

4) We introduce methods for the general case of transition relations. This includes approximative as well as exact methods which are based on QBF and MaxQBF solving.

5) We provide a thorough analysis which methods need which properties of the transition relation to be correct. In that way we provide users of PO generalization methods with a guide telling which methods can be applied in which context.

6) We provide a thorough comparison of the generalization strengths of the different methods from a theoretical point of view.

7) From a practical point of view, we present an intensive evaluation of the different methods and also combinations of some of them. We consider HWMCC benchmarks with and without invariant constraints [15]–[17], as well as AI Planning benchmarks from the International Planning Competition (IPC). Our results show that the novel methods can improve on well-established existing generalization methods. Due to the complementary strengths of the methods the results can be further improved by portfolio applications of different methods. Even the most expensive methods are able to contribute to overall runtime improvements by providing stronger generalizations.

Moreover, exact methods are used to analyze the potential for improvement of approximate methods.

Related work: Since the introduction of PDR [1], there have been several improvements on the efficiency of the original algorithm. One important insight of [11] was the use of a dynamic generalization technique for POs by using ternary simulation (01X simulation) instead of a static cone-of-influence analysis as performed in [1]. This greatly affected the algorithm's efficiency in terms of runtime and memory consumption. The authors of [18] present a similar generalization technique – to which we refer as *lifting* – which uses a SAT solver call instead of simulation. We consider ternary simulation as well as lifting as the two most used standard techniques for the generalization of proof obligations in the context of digital circuits. We compare them to other in PDR yet unused techniques and even extend the exposition of [18] by a detailed analysis of the limitations of lifting.

In [6], [19], limitations of these techniques are also discussed in the context of spurious POs under abstraction and generalization under invariant constraints. The authors of [20], [21] discuss PO generalization in the context of Reverse PDR. We however give a *general* theoretical analysis of the preconditions for lifting and differentiate between lifting being either incorrect or unable to find generalizations.

For digital circuits, [22] analyzes two techniques for generalizing counterexamples in BMC. The lifting approach which is one of them has been adapted to PDR in [18]. Here we adapt the other one to PDR as well and call it 'Implication Graph Based Generalization' (IGBG) in this paper.

In the more general context of PDR on top of SAT modulo theories (SMT), the authors of [3] briefly mention a cover approach with don't care values for generalizing POs. Technically, this is very much related to the SAT-based cover approach that we discuss for general transition relations. Furthermore, there are various works which discuss finding minimal satisfying assignments for CNF formulae [23]–[26], which relate to the clause cover approaches that we have incorporated for PDR.

In another field of research, namely circuit testing, finding minimal test cubes closely relates to finding minimal POs in PDR. The authors of [27], [28] investigate different techniques including MaxSAT and MaxQBF wrt. their feasibility and generalization capabilities. Furthermore, the CEGAR-based approach of [29] includes a technique similar to [22] in order to perform automatic test pattern generation in the presence of unknown values.

Structure of the paper: In Sect. II we give some preliminaries needed for this In Sect. III we begin with discussing the complexity of the problem of PO generalization in general, we present a wide range of generalization techniques for circuits in Sect. IV as well as for general transition relations in Sect. V. Finally, we analyze the generalization capabilities of the different techniques from a theoretical point of view and introduce further improvements in Sect. VI. An experimental evaluation is given in Sect. VII, and Sect. VIII summarizes the results with directions for future research.

II. PRELIMINARIES

A. Basics and Notations

We discuss reachability analysis in finite state transition systems, which has many applications and can be used, e. g., for the verification of invariant properties or for finding plans in AI Planning tasks.

Finite state transition systems $M = (\{0,1\}^m, \{0,1\}^n, I, T)$ describe transitions between states from $\{0,1\}^m$ under inputs from $\{0,1\}^n$. $I \subseteq \{0,1\}^m$ is the set of initial states, $T \subseteq \{0,1\}^m \times \{0,1\}^n \times \{0,1\}^m$ is the transition relation. There is a transition from state $\vec{\sigma} \in \{0,1\}^m$ to state $\vec{\tau} \in \{0,1\}^m$ under input $\vec{\iota} \in \{0,1\}^n$ iff $(\vec{\sigma}, \vec{\iota}, \vec{\tau}) \in T$. A trace of M is a sequence of states $(\vec{\sigma_0}, \vec{\sigma_1}, \ldots)$ with $\vec{\sigma_0} \in I$, $\vec{\sigma_j} \in \{0,1\}^m$ and $\exists \vec{\iota_j} \in \{0,1\}^n$ with $(\vec{\sigma_j}, \vec{\iota_j}, \vec{\sigma_{j+1}}) \in T$ for all $j \in \mathbb{N}$. The 'reachable states' of M are the states occurring on traces. The goal of reachability analysis is to either compute all reachable states or to decide whether some states from a given set are reachable. For symbolic representations of states, sets and relations we introduce (present) state variables $\vec{s} = (s_1, \ldots, s_m)$, input variables $\vec{i} = (i_1, \ldots, i_n)$, and next state variables $\vec{s'} = (s'_1, \ldots, s'_m)$. States are obtained by assigning Boolean values to variables $\vec{s}$, inputs by assigning Boolean values to variables $\vec{i}$ etc.. The transition relation is then represented by a predicate $T(\vec{s}, \vec{i}, \vec{s'})$, the set of initial states of M is identified with a predicate $I(\vec{s})$. For brevity, we often omit the arguments of the predicates and write them without parenthesis.

Hardware Model Checking: In the context of sequential hardware verification, the transition relation T is derived from a circuit and therefore represents a Boolean *function* from $\{0,1\}^m \times \{0,1\}^n$ to $\{0,1\}^m$. The set of unsafe states (in case of verification of invariant properties) is represented by a predicate $\neg P(\vec{s})$. Reachability analysis checks whether some unsafe state is reachable.

AI Planning: We consider planning problems which implement the *propositional* STRIPS planning formalism. A STRIPS planning task $P = (\vec{s}, I, G, A)$ is defined by a set of state variables $\vec{s}$ with their next state counterparts $\vec{s'}$, a predicate $I(\vec{s})$ which identifies the initial states, a predicate $G(\vec{s})$ which identifies the goal states, as well as a set of actions A. Encoding schemes like from [30] transform planning tasks into reachability problems on finite state transition systems. The resulting transition relation is not necessarily a function but a *general* transition relation. Reachability analysis checks whether some goal state is reachable.

A *literal* represents a Boolean variable or its negation. *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. A cube $c = s_{i_1}^{\sigma_1} \wedge \ldots \wedge s_{i_k}^{\sigma_k}$ of literals over state variables with $i_j \in \{1, \ldots, m\}$, $\sigma_j \in \{0,1\}$, $s_{i_j}^0 = \neg s_{i_j}$ and $s_{i_j}^1 = s_{i_j}$ represents the set of all states where s_{i_j} is assigned to σ_j for all $j = 1, \ldots, k$. Usually we use letters c or $\hat{c}$ to denote cubes of literals over present state variables, d' or $\hat{d}'$ to denote cubes of literals over next state variables, and i to denote cubes of literals over input variables. Sometimes

```
1  function Pdr(I, T, P)
2      if BaseCases() = 'Unsafe' then return 'Unsafe'
3      while true do
4          if Strengthen() = 'Unsafe' then return 'Unsafe'
5          N ← N + 1, add new R_N ← P  /* New time frame. */
6          if Propagate() = 'Safe' then return 'Safe'
```

Algorithm 1: PDR: main loop.

```
1  function Strengthen()
2      while SAT?[R_N ∧ T ∧ ¬P'] do  /* SAT: error pred.    */
3          m ← satisfying present state assignment
4          c ← SatGeneralization(m)
5          if ResolveRecursively(c, N) = 'Unsafe' then return
               'Unsafe'
6      return 'strengthened'  /* successfully strengthened */
```

Algorithm 2: PDR: strengthen the trace.

```
1  function ResolveRecursively(d, k)
2      if k = 0 then  /* Proof obligation in frame 0.        */
3          return 'Unsafe'
4      while SAT?[¬d ∧ R_{k-1} ∧ T ∧ d'] do  /* pred. in R_{k-1}?  */
5          m̂ ← satisfying present state assignment
6          ĉ ← SatGeneralization(m̂)
7          if ResolveRecursively(ĉ, k - 1) = 'Unsafe' then return
               'Unsafe'
8      d̂ ← UnsatGeneralization(d)
9      R_1 ← R_1 ∧ ¬d̂, ..., R_k ← R_k ∧ ¬d̂
10     return 'resolved'
```

Algorithm 3: PDR: recursively resolve PO (d, k).

we write $c(\vec{s})$, $d'(\vec{s'})$ etc. to emphasize on which variables the corresponding cubes depend. By *minterms* (often named m) we denote cubes containing literals for *all* state variables. Minterms represent single states.

We assume that the transition relation T of a finite state transition system has been translated into CNF by standard methods like [31]. Modern SAT solvers [32] are able to check the satisfiability of Boolean formulas in CNF. We denote a satisfiability check performed by a SAT solver for some formula F by SAT?$[F]$. If the SAT solver terminates, it reports either 'satisfiable' or 'unsatisfiable'. We use the same terminology for satisfiablity checks of QBF formulas.

Reachability analysis (e. g., by PDR) often makes use of special properties of the transition relation T. For instance when T results from a circuit, then it represents a *function*, i. e., it is *right-unique* and *left-total*. A relation $T(\vec{s}, \vec{i}, \vec{s'})$ is right-unique iff for all assignments $\vec{\sigma}$ to $\vec{s}$ and $\vec{\iota}$ to $\vec{i}$ there is *at most* one assignment $\vec{\tau}$ to $\vec{s'}$ such that $(\vec{\sigma}, \vec{\iota}, \vec{\tau}) \in T$. $T(\vec{s}, \vec{i}, \vec{s'})$ is left-total iff for all assignments $\vec{\sigma}$ to $\vec{s}$ and $\vec{\iota}$ to $\vec{i}$ there is *at least* one assignment $\vec{\tau}$ to $\vec{s'}$ such that $(\vec{\sigma}, \vec{\iota}, \vec{\tau}) \in T$. Similarly, $T(\vec{s}, \vec{i}, \vec{s'})$ is *left-unique* (*right-total*) iff for all assignments $\vec{\tau}$ to $\vec{s'}$ and $\vec{\iota}$ to $\vec{i}$ there is *at most* (*at least*) one assignment $\vec{\sigma}$ to $\vec{s}$ such that $(\vec{\sigma}, \vec{\iota}, \vec{\tau}) \in T$.

B. An Overview of PDR

In this paper we consider Property Directed Reachability (PDR) [11] (also called IC3 [1]).

Without unrolling the transition relation as in Bounded Model Checking (BMC) [10], PDR produces sets of clauses for each time step individually with the ultimate goal of finding an inductive strengthening of the safety property P (proof of safety). We call these sets *time frames* and each time frame k corresponds to a predicate R_k represented by a set of clauses[1]. Hereby, for each time frame $k \geq 1$, PDR proceeds with a new time frame $k+1$ if the clauses created in R_k are sufficient such that $R_k \wedge T \Rightarrow P'$. Additionally, PDR maintains the invariant that all clauses $\neg c$ from R_{k+1} are *inductive relative* to R_k, i.e. $(\neg c \wedge R_k \wedge T) \Rightarrow \neg c'$ for $\neg c \in R_{k+1}$ which is exactly the case if $\neg c \wedge R_k \wedge T \wedge c'$ is unsatisfiable. As a result, R_k over-approximates the set of states which can be reached from I in up to k steps and thus the state sets represented by the R_i are monotonically increasing in i (for $i \geq 1$). R_0 is always equal to I.

We present the main loop of PDR in Alg. 1. In iteration N, PDR basically tries to construct error paths of length $N+1$ and starts with checking whether $R_N \wedge T \Rightarrow P'$ via a SAT solver call with SAT?$[R_N \wedge T \wedge \neg P(\vec{s'})]$ in Strengthen() (Alg. 2).

---

[1] In the following we often identify predicates R_k with the state sets represented by them. We further identify the predicate T with the transition relation it represents.

If the SAT solver reports 'satisfiable', a predecessor minterm m is extracted from the satisfying assignment, m is 'generalized' to a cube c, and thus c represents only predecessor states of the unsafe states. It has to be proven that there is no path from the initial states to c. To do so, the 'proof obligation' (PO) c on level N (also called *Counterexample To Induction* (CTI)) has to be recursively resolved. For POs d on level k in general, ResolveRecursively (Alg. 3), checks whether the clause $\neg d$ is inductive relative to R_{k-1}, i.e. $\neg d \wedge R_{k-1} \wedge T \Rightarrow \neg d'$, leading to new SAT calls SAT?$[\neg d \wedge R_{k-1} \wedge T \wedge d']$ (Alg. 3, l. 4).

If this SAT query is unsatisfiable, then d has no predecessor in R_{k-1} and therefore $\neg d \wedge R_{k-1} \wedge T \Rightarrow \neg d'$ holds. After a possible generalization into $\hat{d}$ it can be blocked in R_i with $i \in \{1, \ldots, k\}$ by $R_i = R_i \wedge \neg \hat{d}$ (Alg. 3, l. 9). If the SAT query is satisfiable, a new predecessor minterm $\hat{m}$ has been found and it is generalized using SatGeneralization() (Alg. 3, l. 6) into a PO $\hat{c}$ at level $k - 1$.

If the strengthening of R_N is sufficient and therefore the SAT call from l. 2 in Alg. 2 is unsatisfiable, we conclude that $R_N \wedge T \Rightarrow P'$, increase N by 1, and continue with the next iteration of PDR in Alg. 1 (l. 5) after trying to propagate all blocked cubes forward (l. 6).

The procedure stops, if an error path is found (PO on level 0) or if during propagation (Alg. 4) R_{k-1} and R_k become equivalent (l. 6), i. e., an inductive invariant R_k has been found.

The efficiency of the method strongly depends on the success of the mentioned generalizations in SatGeneralization() (l. 4 of Alg. 2 and l. 6 of Alg. 3) and UnsatGeneralization() (l. 8 of Alg. 3).

In this paper we provide a detailed analysis of the generalization of POs in SatGeneralization().

Some of the known methods for that purpose assume special properties of the transition relation T such as the *function* property, since it results from a digital circuit. Those properties do not hold in all application contexts, e. g., they do not hold for transition relations occurring in AI Planning, for transition relations resulting from circuits with additional invariant constraints, or for transition relations in *Reverse*

```
1  function Propagate()
2  │   for k ∈ {1, . . . , N − 1}, c blocked in Rₖ do
3  │   │   if ¬ SAT?[Rₖ ∧ T ∧ c'] then
4  │   │   │   Rₖ₊₁ ← Rₖ₊₁ ∧ ¬c   /* UNSAT: push forward */
5  │   │   if Rₖ ≡ Rₖ₊₁ then
6  │   │   │   return 'Safe'            /* Proof of safety. */
7  │   return 'propagated'
```

Algorithm 4: PDR: propagate blocked cubes forward.

PDR [20], [21].

Reverse PDR computes overapproximations RR_k of the sets of states from which $\neg P(\vec{s})$ can be reached in up to k steps. As already observed in [11] and [13], there is a simple way to arrive at an implementation of Reverse PDR based on the fact that there is a path from $I(\vec{s})$ to $\neg P(\vec{s})$ using a transition relation T iff there is a path from $\neg P(\vec{s})$ to $I(\vec{s})$ using the 'reverted transition relation'. Thus, a basic version of Reverse PDR is obtained just by exchanging $I(\vec{s})$ with $\neg P(\vec{s})$ and interpreting the predicate for T 'the other way around'.

III. GENERALIZATION OF POs AND ITS COMPLEXITY

Generalization plays a crucial role for the efficiency of PDR [11], [12]. As explained above, generalization in PDR takes place, when clauses are learnt as well as when new POs are created. In this paper we restrict our attention to the latter type of generalization.

Assume that we try to resolve a PO d by a call $SAT?[\neg d \wedge R_{i-1} \wedge T \wedge d']$, but the SAT solver returns a (full) satisfying assignment with a minterm m representing a single current state. m is then a new PO, but before trying to resolve this PO we try to generalize it into a shorter cube c. The question, whether a given subcube c of m is still a PO with successors in d, can be formulated as the following problem:

Definition 1: (PO Generalization Problem (POGP)) Given a transition relation $T(\vec{s}, \vec{i}, \vec{s'})$, a cube $c = s_1^{\sigma_1} \wedge \ldots \wedge s_k^{\sigma_k}$ over present state variables, and a cube $d' = (s_1')^{\tau_1} \wedge \ldots \wedge (s_l')^{\tau_l}$ over next state variables[2], decide whether *for all* $(\sigma_{k+1}, \ldots, \sigma_m) \in \{0,1\}^{m-k}$ *there is* $(\tau_{l+1}, \ldots, \tau_m) \in \{0,1\}^{m-l}$ and an input $\vec{\imath} \in \{0,1\}^n$, such that $T(\sigma_1, \ldots, \sigma_m, \vec{\imath}, \tau_1, \ldots, \tau_m) = 1$, i.e., such that there is a transition from $(\sigma_1, \ldots, \sigma_m)$ to $(\tau_1, \ldots, \tau_m)$ under input $\vec{\imath}$.

The problem formulation contains a quantifier alternation which is already an indicator for the hardness of POGP.

Theorem 1: POGP is Π_2^P-complete.

Proof: We show that POGP is Π_2^P-hard by reducing 2-QBF to POGP. We consider a 2-QBF formula $\phi = \forall \vec{x} \exists \vec{y} : \Phi(\vec{x}, \vec{y})$ with $\vec{x} = (x_1, \ldots, x_p)$, $\vec{y} = (y_1, \ldots, y_n)$. Now define $T(\vec{s}, \vec{i}, \vec{s'}) := s_1 \wedge \Phi(\vec{x}, \vec{y}) \wedge s_1' \wedge \ldots \wedge s_{p+1}'$ with $\vec{s} := (s_1, x_1, \ldots, x_p)$, $\vec{i} = \vec{y}$, $\vec{s'} = (s_1', \ldots, s_{p+1}')$. Further define $c = s_1$ and $d' = s_1' \wedge \ldots \wedge s_{p+1}'$. The defined instance of POGP asks whether for all $\sigma_2, \ldots, \sigma_{p+1} \in \{0,1\}^p$ there is $\vec{\imath} \in \{0,1\}^n$ such that $T(1, \sigma_2, \ldots, \sigma_{p+1}, \vec{\imath}, 1, \ldots, 1) = \Phi(\sigma_2, \ldots, \sigma_{p+1}, \vec{\imath}) = 1$. The answer is yes iff $\forall \vec{x} \exists \vec{y} : \Phi(\vec{x}, \vec{y})$ is satisfiable.

POGP is *in* Π_2^P, since its answer is yes iff the 2-QBF $\forall s_{k+1} \ldots \forall s_m \exists i_1 \ldots \exists i_n \exists s_{l+1}' \ldots \exists s_m' : c(\vec{s}) \wedge T(\vec{s}, \vec{i}, \vec{s'}) \wedge d'(\vec{s'})$ is satisfiable. ∎

The proof of Theorem 1 shows that POGP can basically be viewed as a 2-QBF problem. From a different point of view, POGP asks whether the cube c is an implicant of the Boolean function $\Phi(\vec{s}) := \exists \vec{i} \exists \vec{s'} : T(\vec{s}, \vec{i}, \vec{s'}) \wedge d'(\vec{s'})$. This point of view does not change the complexity of the problem and to take advantage of this view algorithmically, we would have to perform symbolic elimination of the quantifiers $\exists \vec{i}$ and $\exists \vec{s'}$ before considering implicants (or *prime* implicants to make the cube c as short as possible).

Due to the high complexity of the problem we first look into approximate solutions in the next two sections. We start in Sect. IV with the special case of sequential circuits and continue with the general case in Sect. V. For the general case we consider an exact method as well. In Sect. VI we compare the strengths of the different methods, analyze the effectiveness of approximate solutions for the special case of left-unique transition relations (motivated by Reverse PDR), and finally discuss further improvements.

IV. APPROXIMATIVE PO GENERALIZATION FOR CIRCUITS

For the special case of digital circuits, where the transition relation represents a *function*, different approximations of POGP can be used.

Firstly, we give a short overview of the commonly used techniques for circuits in Sect. IV-A. These are 01X-simulation as proposed in [11] as well as the lifting approach proposed in [18] which is based on a technique of lifting BMC counterexamples from [22]. Besides 01X-simulation and lifting, we also consider the justification technique which is implemented as an optional PO generalization technique in ABC's [33] PDR implementation a known 'standard method'.

While it is rather obvious that 01X-simulation of circuits does not apply for general transition relations, it is more subtle in the case of the lifting approach. Therefore we thoroughly discuss the limitations of lifting and additionally discuss extensions which may improve its generalization capabilities in Sect. IV-B. We also present two techniques for circuits which have not been used in the context of PDR yet (to the best of our knowledge): (1) It is feasible to find a state with the maximum amount of X-valued state-bits by using a 01X-encoding of the circuit and a MaxSAT solver (similar to [27], see Sect. IV-D). Note that [23] and [24] discuss an approximate version of this method which uses a SAT solver with an appropriate decision heuristics (see Sect. IV-E). (2) Additionally, for BMC, [22] presents an alternative to the mentioned lifting technique which is based on a reverse traversal of the implication graph of a SAT solver. This method – we call it IGBG – can be adapted to the PDR case, too (see Sect. IV-C).

Finally, at the end of this section, we consider the case of circuits with invariant constraints which lead to transition relations not representing functions (see Sect. IV-F).

A. Standard Methods

1) 01X-Simulation: This approach uses a three-valued logic with a don't care value X (the two-valued semantic can be

---

[2]To simplify notations we assume here *w.l.o.g.* that the variables not occurring in c and d' are at the end of the vector of state variables.

extended by $(X \wedge 0 = 0), (X \wedge 1 = X), (X \wedge X = X), (\neg X = X)$). We start with a (full) satisfying assignment to $\neg d \wedge R_{k-1} \wedge T(\vec{s}, \vec{i}, \vec{s'}) \wedge d'(\vec{s'})$ leading to a PO state m. Now, present state bits from m are iteratively assigned to X followed by a simulation of the circuit. If an X propagates to an output which is asserted by $d'(\vec{s'})$, the state bit is necessary in m, otherwise it is redundant and can be removed from m. The process is greedily iterated until no more redundant state bits are found. Apart from the greedy search for redundant state bits and from the fact that only predecessors of d' under a fixed input assignment i are considered, ternary simulation has an additional source of non-optimality: As an example consider an AND-gate with output b where both inputs are just the negation of each other: $b \leftrightarrow (a \wedge \neg a)$. If a is assigned to X, the X will propagate to b using the rules of three-valued logic even though b is constant–0. Since this method uses ternary simulation of circuits, it is inherently restricted to transition relations resulting from circuits (which are transition *functions*).

2) Justification Based Generalization: A technique strongly related to 01X-simulation is to apply justification to the circuit. Given a full assignment m to all present state variables, i to all primary inputs, and d' to a subset of the next state variables, we look for a *partial* assignment to the present state variables which is still able to justify resp. imply the assignment d'.

In principle, we traverse the circuit and heuristically determine the variables of m which are (together with all variables from i) sufficient to imply d'.

First of all, the circuit is simulated with the assignment m and i. Second, the literals of all present state variables which are not included in the syntactical support set of the next state variables contained in d' are removed from m. Then, priorities are assigned to all primary input variables (∞) and to all variables of the remaining literals in m (arbitrary natural numbers). We prefer to keep variables with a higher priority in the assignment. All input literals receive priority ∞, because i remains untouched and therefore, if a next state assignment can be justified by an input or a state variable, we will always prefer to use the input and ignore the state variable. Now an iterative procedure is started. In a first iteration, the priorities are forward propagated from the present state variables and primary inputs towards the next state variables (outputs of the circuit). The priority of a (circuit input or gate output) variable v is denoted by $prio(v)$ in the following. By this propagation, the method implicitly constructs justification paths from the circuit inputs to the next state variables in d'. Consider a gate with output z and inputs x, y. If the value at z is only justified by x and not by y, then $prio(x)$ is propagated to z, since there is no choice for justification. If the value at z can be justified by x *or* by y, then the higher priority is propagated to z, since we prefer justification paths starting from variables with high priority. If both values of x *and* y are needed to justify the value at z, then the lower priority is propagated to z in order to remember overall the input with the lowest priority which is connected to z by a justification path. For an AND-gate $z \leftrightarrow x \wedge y$, e.g., this leads to the following rules:

1) If $z = 0$ and $x \oplus y = 1$, then $prio(z) = prio(min(x,y))$.
2) If $z = 0$ and both $x = 0$ and $y = 0$, then $prio(z) = max(prio(x), prio(y))$.

Fig. 1: Not right-unique

3) If z is 1, then both $x = 1$ and $y = 1$ and therefore $prio(z) = min(prio(x), prio(y))$.

After propagating, we pick the lowest priority, say $prio(v_0)$, which arrived at some next state variable from d'. The propagation of $prio(v_0)$ to a next state variable from d' means that we could not avoid to include v_0 into the implicitly constructed system of justification paths, although we prefer variables with high priority. Thus we add the according literal of v_0 from m to our (initially empty) generalized proof obligation cube $\hat{c}$. We now set $prio(v_0) = \infty$, because we already consider this variable in our generalized cube, and start with the next forward propagation iteration. We terminate, once we only observe priorities ∞ at the next state variables after propagating the current priority assignment. Then we include all corresponding literals into the partial assignment.

A partial assignment achieved by this method is 01X-simulatable.

3) Lifting: The authors of [18] propose an approach which uses an unsatisfiable SAT solver query that reveals a generalization of the PO state. Assume a circuit defining a transition function T. In the original PDR approach a satisfiable query $SAT?[\neg d \wedge R_{k-1} \wedge T \wedge d']$ provides a satisfying minterm m and some complete assignment i to the primary inputs $\vec{i}$. Since $m \wedge i$ is a complete assignment to all inputs of the circuit defining the transition function, it implies a fixed next state in the cube d'. Thus, the 'lifting query' $SAT?[m \wedge i \wedge T \wedge \neg d']$ is unsatisfiable by construction. The final conflict clause of this query yields a generalization of m, because we are now able to remove all literals from m which are unnecessary for the unsatisfiability proof. Again results are not necessarily optimal, since they depend on the order in which the literals of m propagate during the SAT solving (and since only a fixed input assignment i is considered). To further increase the number of removed literals in lifting, [22] proposes to iteratively omit literals from the unsatisfiable core (revealed by a final conflict clause) and query the solver again with the corresponding unsatisfiable lifting call. This procedure is called *literal dropping* and trades runtime against more general POs. For our experiments in Sect. VII-A1 we consider both variants.

B. Limitations and Extensions of Lifting

Here we discuss the preconditions we require for a sound application of lifting in PDR as well as possible extensions in order to improve its efficiency. For the lifting approach to be correct, T has to represent a *function* (i.e., T is *left-total* and *right-unique*, see Sect. II-A). We now consider those two properties separately.

Firstly, we assume that T is *not* right-unique (see Fig. 1). This means that the assignment $m \wedge i$ does not necessarily imply *one unique* successor state. This property could render

Fig. 2: Not left-total

our lifting query $SAT?[m \wedge i \wedge T \wedge \neg d']$ satisfiable, since there could indeed be another transition from $m \wedge i$ to a state outside d'. Thus, the approach would say that the PO m cannot be generalized, although this could actually be possible. Existentially quantifying the input vector $\vec{i}$ instead of setting it to one fixed assignment i would not improve the situation (but rather make it worse), because this would increase the probability of having transitions to states outside d'.

Secondly, we assume that T is *not* left-total (see Fig. 2). This means that there are present state / input combinations which do not lead to any successor state at all. We consider the state $\hat{m}$ which results from removing literal l from m, i.e., $\hat{m} = m \setminus \{l\}$. Thus in the beginning $\hat{m} \wedge l \wedge i \wedge T \wedge d'$ is satisfiable.

We further assume that $\hat{m} \wedge \neg l$ has no successor in T at all (T is not left-total), i.e., $\hat{m} \wedge \neg l \wedge T$ is already unsatisfiable. Now $\hat{m} \wedge i \wedge T \wedge \neg d'$ is unsatisfiable, such that the lifting query would remain unsatisfiable when literal l is dropped from m. However, $\hat{m}$ is not necessarily a correct PO, since it is not possible to reach d' from each point (state) in $\hat{m}$.

To summarize, while missing right-uniqueness can only lead to (unnecessarily) failing lifting attempts, it is most crucial to ensure left-totality, since otherwise lifting could lead to wrong results in terms of spurious counterexamples.

Finally, we discuss two variants which can potentially speed up the lifting approach and/or improve its results. We apply the approximate SAT approach from [12] to lifting in order to investigate its isolated effect on PO generalization. We further introduce literal rotation from [34] to bit-level SAT-based PDR.

1) Approximate SAT: If we decide to apply iterative literal dropping, we can make use of the observation made in [11] that – in the context of lifting – satisfiable calls of the SAT solver are much more costly in terms of runtime than unsatisfiable calls and that the SAT solver usually reports unsatisfiability after only deciding few variables. The authors of [12] therefore propose a technique which they call 'approximate SAT' and which considers any SAT call as satisfiable once a certain number of decisions is made by the SAT solver (in [12] a constant number of 100 is proposed). Hence, we can avoid unnecessary computation time in satisfiable SAT solver calls which would in the end only conclude that we have to keep a certain literal anyway. On the other hand we could prematurely conclude that a call is satisfiable and keep a literal even though it would not have been necessary. Thus there is a trade-off between runtime and accuracy. In our experimental section we will analyze whether this technique is worthwhile.

2) Literal Rotation: The authors of [34] propose to additionally 'rotate' literals in order to replace or complement standard literal dropping. Technically, we provide the cube $m = l_1 \wedge \ldots \wedge l_k \wedge \ldots \wedge l_n$ as assumptions to an incremental SAT solver (in the order $l_1, \ldots, l_n$). Once a literal l_k is

conflicting, the SAT solver will traverse the implication graph and collect the previously decided assumptions $l_i, \ldots, l_j$ with $i, j \in \{1, \ldots, k-1\}$ which are necessary for the conflict (unsatisfiable core). lifting without any literal dropping would conclude that the literals $l_i, \ldots, l_j, l_k$ are necessary and subsequently, that we may generalize m to $\hat{m} = l_i \wedge \ldots \wedge l_j \wedge l_k$. Literal *rotation* however invokes the SAT solver again on the reduced set of assumptions $\hat{m}$ and 'rotates' the order of the assumptions to $(l_k, l_i, \ldots, l_j)$. As a result a newly found unsatisfiable core will at most contain all literals of $\hat{m}$ and will possibly reveal a more general unsatisfiable core. We remark that since $l_i, \ldots, l_j$ obviously implies $\neg l_k$ the call will remain unsatisfiable. These solver queries are rather inexpensive [34] and therefore, we can repeat rotating until some literal would re-appear as the first one in the order. We remark, that even if we have performed all possible rotations on an initial conflict, there might still be literals in the resulting cube which can be removed by further literal dropping [34]. Again, we will discuss the efficiency of this method in our experimental section.

C. Implication Graph Based Generalization (IGBG)

We can also adapt another method to PO Generalization in PDR which is inspired from [22] as the aforementioned lifting method. Having a circuit, applying a full assignment i (for primary inputs) and m (for state variables) to T, i.e., querying the SAT solver with $SAT?[m \wedge i \wedge T]$, the SAT solver will only require Boolean Constraint Propagation (BCP) to deduce a satisfying assignment of the formula. Hence it is possible to just traverse the implication graph in a backward direction and collect the literals from m which are responsible for implying the next state valuation d'. Obviously, the method makes use of the right-uniqueness property of transition functions (since otherwise BCP would not be sufficient).

Interestingly, a reduced cube $\hat{c}$ resulting from this method is exactly 01X-simulatable, i.e., if we apply it as a simulation pattern with all don't care literals (which are not contained in $\hat{c}$) set to X, then no X-value will propagate to the next-state variables included in d' [22].

D. MaxSAT 01X-encoding

In order to avoid the iterative greedy approach of 01X-simulation for removing redundant state bits, we introduce a partial MaxSAT [35]–[37] encoding to find a better approximate solution to POGP. Partial MaxSAT problems consist of hard clauses and soft clauses. A MaxSAT solution satisfies all hard clauses and a maximal number of soft clauses.

For the 01X-encoding of the Boolean circuit for the transition function we introduce *two* variables $v^{(0)}$ and $v^{(1)}$ for each Boolean variable v which represents either an input, an output or an internal signal, while $((v^{(0)} = 0 \wedge v^{(1)} = 0) \leftrightarrow v = X)$ as well as $((v^{(0)} = 1 \wedge v^{(1)} = 0) \leftrightarrow v = 0)$ and $((v^{(0)} = 0 \wedge v^{(1)} = 1) \leftrightarrow v = 1)$; we explicitly forbid $(v^{(0)} = 1 \wedge v^{(1)} = 1)$. All gates are replaced by a two-rail encoding according to [38]. The 01X-encoded circuit simulates information propagation using 01X-logic. For each state variable s_i we introduce a new variable t_i and a unit soft clause $sc_i = \{t_i\}$ accompanied by the hard clauses representing $t_i \leftrightarrow ((s_i^{(0)} = 0) \wedge (s_i^{(1)} = 0))$. Starting with a satisfying solution to $SAT?[\neg d \wedge R_{k-1} \wedge T \wedge d']$

with $d' = ((d'_{i_1})^{\tau_1} \wedge \ldots \wedge (d'_{i_k})^{\tau_k})$ that provides full assignments $m = s_1^{\sigma_1} \wedge \ldots \wedge s_m^{\sigma_m}$ and $i = i_1^{\iota_1} \wedge \ldots \wedge i_n^{\iota_n}$, we introduce hard clauses fixing state bits s_i to X or σ_i, input bits i_j to ι_j, and next state bits d'_{i_j} to τ_j. The other hard clauses of the considered MaxSAT problem correspond to the 01X-encoding of T. Maximizing the number of satisfied soft clauses means maximizing the number of present state bits which are assigned to X and are thus not included in the resulting subcube c of m from which all transitions under i lead into d'. We call the resulting MaxSAT problem *MS01X*.

E. SAT 01X-encoding

It is also possible to approximate *MS01X* by using a simple SAT solver [24]. We compute a 01X-encoding of the circuit like we do for *MS01X*, but omit the MaxSAT-specific clauses. Here the notion of *sign-minimality* [23] is exploited, which describes the fact that if a SAT solvers' decision heuristics only decides Boolean variables with one polarity (say 0), then the resulting model has a (locally) maximal number of variables assigned to this polarity. If we employ the 01X-encoding scheme from above (which encodes X with (00)) and the SAT solver makes only decisions to 0, then the resulting model has a (locally) *maximal* (but not necessarily globally *maximum*) number of state bits assigned to X. In the following, we call this technique *S01X*.

F. PO Generalization with Invariant Constraints

It is important to note that even in the context of transition relations defined by circuits, the transition relation is not necessarily a function. A common reason for non-left-total transition relations are invariant constraints (e. g., restricting the inputs). The AIGER 1.9 standard [39] is a popular example for this. Here a circuit with transition relation T is restricted by an invariant constraint C. If some present state / input combination does not satisfy C, then there is no transition from this state under this input assignment, i. e., the resulting transition relation is not left-total. This immediately implies that the lifting approach to PO generalization may produce erroneous results (see above) and cannot be used. There are several options to avoid this problem:

1) One can use PO generalization techniques for general transition relations that will be discussed in Sect. V. Our experimental results show however that this leads to sub-optimal generalizations.
2) One can use *IGBG* which requires right-uniqueness to be applicable, but not left-totality.
3) One can use 01X-simulation with the additional requirement that it does not produce an X at the output of C.
4) It is possible to transform the transition relation into a right-unique and left-total transition function.
   a) We can maintain the same set of reachable states by introducing self-loops for each non-admissible transition. For this, we simply introduce for each state variable a multiplexer which feeds back the old state value in case that the invariant constraint is violated.
   b) We can introduce a new dead-end state and direct all non-admissible transitions into this state. To do so, the tool *aigunconstraint* from the AIGER-suite [39] introduces an additional latch for implementing the dead-end state (doubling the state space and thus changing the set of reachable states).
5) The original lifting call from Sect. IV-A3 can be changed to consider the invariant constraints. If we are able to separate C from T having $T = \hat{T} \wedge C$, we can change $SAT?[m \wedge i \wedge T \wedge \neg d']$ into $SAT?[m \wedge i \wedge \hat{T} \wedge (\neg C \vee \neg d')]$. By construction, the minterm m satisfies the invariant constraint and the transition from m under i leads into d'. Therefore the changed SAT query is unsatisfiable as well. If the SAT query remains unsatisfiable for some subcube c of m, then it is of course unsatisfiable for each state m' in c, thus each such m' satisfies C and the transition from m' under i leads into d', i. e., c is a PO.

V. PO GENERALIZATION FOR GENERAL TRANSITION RELATIONS

Here, we look into methods for PO generalizations that work without specific assumptions on the transition relation. We start with approximate techniques and finally consider exact solutions. First, we describe a new technique called GeNTR. We have already used a corresponding technique in the context of SMT-based PDR [34]. Secondly, we adapt well-known covering approaches and use them in the context of bit-level PDR. Lastly, we introduce techniques which solve the underlying 2-QBF problem and remove literals greedily as well as optimally using MaxQBF. While both QBF approaches are completely new in the context of PDR, a similar MaxQBF approach has been used in [27] in the context of test cube generation.

A. Generalization with Negated Transition Relation (GeNTR)

If T does not represent a function, it is possible to lift m with a similar formula as in the lifting approach from Sect. IV-A3. Assume a satisfiable query $SAT?[\neg d \wedge R_{k-1} \wedge T \wedge d']$. A SAT solver provides complete assignments m, t', and i to state variables $\vec{s}$, next state variables $\vec{s}'$, and additional variables $\vec{i}$ of T. The cube $m \wedge i \wedge t'$ represents a satisfying assignment of the predicate T. Hence, this cube renders $\neg T$ unsatisfiable. Therefore, the query $SAT?[m \wedge i \wedge \neg T \wedge t']$ is unsatisfiable and its final conflict clause can be used to obtain a generalization of m.

B. Covering Approach

Another technique for general transition relations is the extraction of a minimal satisfying assignment given a complete satisfying assignment from a SAT solver query. Extracting a partial assignment which still satisfies all clauses is equivalent to the *Hitting Set* problem, a special case of *Set Cover* [40]. We give the intuition: Given a set of clauses Γ and a full satisfying assignment A, pick a subset of the elements (literals) of A which 'hit' all clauses in Γ. For brevity, we focus on the most commonly referenced methods in this context – a greedy algorithm and an ILP-encoding.

a) Greedy Algorithm: We start with a full satisfying assignment A to $\neg d \wedge R_{k-1} \wedge T \wedge d'$ and Γ contains the clauses representing T. Initially, our partial assignment P consists of all literals in A which are not present state literals. P is removed from A, and all clauses which are covered by literals from P are removed from Γ. Then we (1) scan the clauses of Γ for the

most frequently occurring literal l of A, (2) add l to the partial assignment P, remove l from A, remove the clauses covered by l from Γ, and start over with (1) until Γ is the empty set. Obviously, the greedy algorithm has a polynomial runtime in the input size. However, the solution is not necessarily optimal w. r. t. the size of the partial assignment.

b) ILP-encoding: The ILP encoding has a binary ILP-variable v_l for each present state variable s_i with literal l occurring in A. It is formulated in a way that in the solution v_l equals 1 iff the corresponding literal l (from A) occurs in the covering. The optimization goal is to minimize the sum over all ILP-variables v_l.

The two (unate) covering approaches mentioned above start with complete assignments $\vec{\sigma}$, $\vec{\tau}$, and $\vec{\iota}$ to state variables $\vec{s}$, next state variables $\vec{s}'$, and additional variables $\vec{i}$ of T, keep the assignments $\vec{\iota}$ and $\vec{\tau}$ fixed and minimize the remaining assignments in $\vec{\sigma}$ while still satisfying T. To give the cover approach a higher degree of freedom, we can also allow to vary the assignments $\vec{\iota}$ to $\vec{i}$ and $\vec{\tau}$ to $\vec{s}'$ as long as $\vec{\tau}$ remains in the next state cube d'. This additional degree of freedom could be easily integrated into the ILP formulation. However, with this additional degree of freedom we rather consider an *approximate* approach based on a SAT solver.

c) SAT-based Cover: In the spirit of the *S01X* approach for circuits from Sect. IV-E we introduce for each present state variable s_i two new variables $s_i^{(0)}$ and $s_i^{(1)}$. An assignment of $(1,0)$, $(0,1)$, or $(0,0)$ to $(s_i^{(0)}, s_i^{(1)})$ means that s_i is 0, 1 or unassigned (X), resp.. We replace in the CNF for T all occurrences of s_i / $\neg s_i$ by $s_i^{(1)}$ / $s_i^{(0)}$ as well as we add a clause $\{\neg s_i^{(0)}, \neg s_i^{(1)}\}$ to rule out the illegal value $(1,1)$ [23], [24]. To ensure that s_i can only be unassigned or equal to the value σ_i fixed by $m = s_1^{\sigma_1} \wedge \ldots \wedge s_m^{\sigma_m}$ we assign $s_i^{(1-\sigma_i)}$ to 0. Moreover, we assign next state variables to enforce d'. As already discussed in Sect. IV-E, a SAT solver which only decides variables to 0 then computes a solution with a locally maximal number of unassigned present state variables.

C. Solving the QBF Problem

As stated in Sect. III the problem of PO generalization is inherently a 2-QBF problem, thus for achieving a minimal-sized PO we have to solve a QBF problem. However, the QBF formulation is just a *decision problem* which checks whether each state in a *given* cube c has a transition into a cube d'. Reducing a given minterm m over state variables to a minimal cube c is an *optimization problem* for which we consider two options:

a) Greedily applying a QBF solver: With this approach we iteratively probe single state literals for don't cares. Instead of flipping a variable to X and simulating the circuit, this generalized approach probes a state variable by universally quantifying it. Assume that we start with a minterm m which is a full assignment to the state variables. If we want to check whether the variables from $\vec{r}$ can be removed from m, leading to $\hat{m}$, we give the query $SAT?[\forall \vec{r} \exists \vec{k} \exists \vec{i} \exists \vec{s}' : \hat{m} \wedge T \wedge d']$ to the QBF solver. Here $\vec{k}$ are the variables from $\vec{s}$ remaining in $\hat{m}$.

Fig. 3: Quality relations between generalization methods

b) Applying MaxQBF: To achieve an optimum we even have to go one step further than pure QBF solving. The notion of (partial) MaxQBF [28], [41] allows us to add soft clauses to our problem to find the maximum number of removable literals. Similar to our MaxSAT approach from above, we introduce a soft clause per state variable which is a candidate for removal. The encoding is more complicated though, since we have to maximize the number of universally quantified state variables. We adapt a technique from [28] which uses a multiplexer for each state variable selecting between either the assignment from m or a universally quantified variable:

- For each state variable s_i we introduce a variable $s_i^{\forall}$ as well as a variable u_i.
- We add clauses for $C_i^{\forall} = u_i \rightarrow (s_i \leftrightarrow s_i^{\forall})$ and $C_i^{\varepsilon} = \neg u_i \rightarrow (s_i \leftrightarrow \varepsilon)$ where ε is the original assignment to s_i in m.
- Furthermore we introduce a unit soft clause $\{u_i\}$ for all s_i.
- We solve the MaxQBF problem $\exists \vec{u} \forall \vec{s}^{\forall} \exists \vec{s} \exists \vec{i} \exists \vec{s}' : T(\vec{s}, \vec{i}, \vec{s}') \wedge d' \wedge \bigwedge_{i=0}^{n} C_i^{\forall} \wedge \bigwedge_{i=0}^{n} C_i^{\varepsilon} \wedge \bigwedge_{i=0}^{n} u_i$.

If in the solution to the MaxQBF problem the soft clause $\{u_i\}$ corresponding to state variable s_i is satisfied, then s_i may be universally quantified and therefore removed from the PO. The result provides a minimal subcube c of m such that there are transitions from all states in c to d'.

VI. ANALYSIS AND IMPROVEMENTS

In Sect. VI-A, we show how to categorize the different approaches we discussed in the preceding sections. Sect. VI-B analyzes the effects of methods from Sect. V on the special case of left-unique transition relations. Furthermore, we discuss the combination of certain techniques in order to achieve stronger generalization results in Sect. VI-C. Finally, in Sect. VI-D, we analyze the additional degree of freedom we might achieve by neglecting the restriction to an initial proof obligation consisting of a minterm (full satisfying assignment of the state variables).

A. Categorizing Generalization Capabilities

Fig. 3 gives an overview of the methods presented in Sects. IV and V. The methods in the first line (written in bold) are suitable for general transition relations, the other methods need transition *functions* or at least a property implied by the function property (such as left-totality for lifting and right-uniqueness for *IGBG*). In the following, we classify the different methods w. r. t. their generalization strength. A solid arrow from method B to method A means that method B is weaker than A in the sense that B always computes weaker (or equivalent) generalizations.

Definition 2 (Weaker Generalization): Consider arbitrary POGPs consisting of present state minterms (PO states) m,

transition relations T, and next state cubes d'. Assume that method A generalizes m to cube c^A, whereas method B generalizes m to cube c^B. We consider B *weaker* than A, if c^A contains always *less or equally many* literals than c^B, i.e., c^A is more or equally general than c^B.

In many cases the relation between two methods A and B is not that easy to categorize. Methods for PO generalization often use heuristics (for removing literals in certain orders, e.g.) and depending on heuristical choices the results are better or worse - so the methods are often incomparable. Nevertheless we introduce here the notion of 'potentially weaker' (represented by dashed arrows in Fig. 3 from method B to method A when B is 'potentially weaker' than A). This notion intuitively means that method B has a conceptual weakness compared to method A (like 01X-simulation which suffers from the imprecision of 01X-logic, in contrast to lifting). This weakness may or may not be hidden by different heuristical decisions. For a formal definition of 'potentially weaker' we make use of the fact that the methods for PO generalization do not need to be started with minterms, but they can also be started with cubes which then either cannot be confirmed to be POs or can be confirmed (and possibly even improved). When B is 'potentially weaker' than A, then B *may* produce more general (better) results than A (due to different heuristical decisions), but if we apply A to a result of B, then A will always at least confirm the result or even improve it. (E.g. lifting started with result cubes of 01X-simulation can always guarantee that those result cubes are valid POs, but 01X-simulation cannot always give such a guarantee for cubes computed by lifting due to the imprecision of 01X-logic.)

Definition 3 (Confirmation of a Generalization Result): Consider an arbitrary POGP consisting of a PO state m, a transition relation T, and a next state cube d'. Assume that m is generalized to a cube c^B by method B. Then method A is able to *confirm the generalization result* of method B, if started with c^B, T, and d', it is able to decide that c^B is a valid PO or if it is able to generalize c^B even further.

Definition 4 (Potentially Weaker Generalization): If method A is able to confirm the generalization result of method B for all POGPs, then B is *potentially weaker* than A.

We first look into the general methods. Beforehand, we prove that the *GeNTR* approach implicitly computes a cover of the CNF for T.

Theorem 2: *GeNTR* (implicitly) computes a cover of the CNF for T.

Proof: *GeNTR* starts with complete assignments m, t', and i to state variables $\vec{s}$, next state variables $\vec{s}'$, and additional variables $\vec{i}$ of T. The query $SAT?[m \wedge i \wedge \neg T \wedge t']$ is unsatisfiable. It computes a subcube c of m such that $SAT?[c \wedge i \wedge \neg T \wedge t']$ is still unsatisfiable, i. e., each extension of $c \wedge i \wedge t'$ to a full assignment evaluates $\neg T$ to 0 and thus T to 1. This means, that each such extension satisfies all clauses in T. This implies that $c \wedge i \wedge t'$ has to cover all clauses in T. ∎

Thus, *GeNTR* is equivalent to the greedy cover approach. Of course, the results may differ, since different heuristics are used. The ILP cover approach solves the covering problem in an optimal way and is thus stronger than greedy covering

Fig. 4: Example 1.

and *GeNTR*. Nevertheless, it is 'potentially weaker' than *SAT-based cover*, since *SAT-based cover* has the freedom to choose different values for $\vec{i}$ variables and $\vec{s}'$ variables which are not fixed by d'. As already discussed in Sect. V-C QBF is exact as a decision problem, but it is not an optimization method which is able to *compute* the reduced cube. Greedy application of QBF depends on the chosen order and therefore it is only potentially stronger than all other methods except for MaxQBF. MaxQBF provides an optimal solution and dominates all other methods.

Now we look at methods suitable for transition functions. *IGBG*, *justification*, 01X-simulation, and *S01X* have basically the same strength, since the results of *IGBG* and *justification* are 01X-simulatable [22] and the result of *S01X* is only locally optimal. The four methods may produce different X-values on present state variables, however. 01X-simulation, *IGBG*, *justification*, and *S01X* are weaker than *MS01X*, since MaxSAT computes an *optimal* selection of X-values for state bits. All 01X-based methods are 'potentially weaker' than lifting. If X-values on present state variables do not propagate to the next state bits included in the next state cube d' for a fixed assignment to the remaining present state variables and the primary inputs, then the values assigned to the d'-variables are implied by this assignment. In other words, this assignment, the transition relation T and $\neg d'$ are contradictory, and thus the present state bits with X-values can be potentially removed by the lifting method. On the other hand lifting does not suffer from the known imprecision of ternary logic as discussed in Sect. IV-A1.

The weaker, covering based methods for general transition relations (greedy cover, *GeNTR*, ILP cover) should not be preferred over circuit-based methods (if they are applicable), since they have a fundamental weakness as illustrated by the following example:

Example 1: Consider a transition relation T specified by the circuit in Fig. 4. The next state cube d' is given by s_1'. We assume that a SAT solver produces the satisfying assignment $i_1 = s_1' = 1$, $s_1 = s_2 = s_2' = 0$, i. e., we start generalization with the minterms $m = \neg s_1 \wedge \neg s_2$, $i = i_1$. It is easy to see that 01X-simulation can assign both s_1 and s_2 to X, leading to an X at output h of the AND-gate, but keeping the 1 at the output s_1' of the OR-gate. The lifting approach can remove the assignments to both s_1 and s_2 as well. However, covering the clauses $(\neg h \vee s_1)$, $(\neg h \vee s_2)$, and $(h \vee \neg s_1 \vee \neg s_2)$ of the AND-gate can only remove *one* of the input assignments ($\neg s_1$ or $\neg s_2$). In general, clause covering means to find assignments to the inputs of all gates that justify the assignments at their outputs. This property largely restricts the potential to remove input assignments to gates by clause covering approaches.

Of course, apart from theoretical comparisons, an experimental evaluation (see Sect. VII) is crucial, since the overall effect

on the PDR algorithm is also influenced by the effectiveness of heuristics as well as by the runtimes needed to compute generalizations.

B. The Special Case of Left-unique Transition Relations

All methods described in Sect. V are sound for arbitrary transition relations and thus also for left-unique transition relations. Our investigations for the special case of left-unique transition relations are motivated by Reverse PDR where the predicate for T is interpreted 'the other way around'. If the original transition relation results from a circuit, i. e., if it is (left-total and) right-unique, then the reverted transition relation is left-unique. In fact, we can prove that the cover methods from Sect. V (including *GeNTR*) do not result in any generalization of POs for left-unique transition relations. Only the QBF- and MaxQBF-based methods are able to generalize POs in this case. Our covering approaches minimize the number of assigned present state variables and still cover all clauses from T. Therefore we can assume w.l.o.g. that all variables from $\vec{i}$ and $\vec{s}'$ are assigned. Since T is left-unique, a full assignment to the variables from $\vec{s}$ is then implied, i. e., removing any assignment of a variable from $\vec{s}$ would render T to be unsatisfied and we would lose the covering property of the assignment.

Example 2: Assume a very simple left-unique transition relation $s_1 = i_1 \wedge s_1'$ with CNF $T = (\neg s_1 \vee i_1) \wedge (\neg s_1 \vee s_1') \wedge (s_1 \vee \neg i_1 \vee \neg s_1')$. It is easy to see that in each covering of T s_1 is assigned. Assume that the next state cube is s_1'. For each valuation of s_1 there is a valuation of i_1 such that $T \wedge s_1'$ is satisfied. Hence, only the QBF approach is able to remove s_1 from an initial cube s_1 or $\neg s_1$.

Since QBF- and MaxQBF-based methods, which are the only promising approaches for left-unique transition relations considered so far, are rather expensive, [20] introduced a *structural* approach to PO generalization for Reverse PDR on circuits. This structural approach basically removes state variables corresponding to the outputs of non-constant circuit outputs with disjoint support sets. Since it is only applicable for Reverse PDR on circuits, we omit a more detailed exposition.

C. Combining State Lifting Methods

Some methods are a good fit for collaboration, e. g., *S01X*, 01X-simulation, *justification*, the implication graph based method *IGBG*, and the MaxSAT 01X-encoding *MS01X*. *S01X*, 01X-simulation, *justification*, and *IGBG* (see Sect. IV-C) yield 01X-simulatable generalization results. Thus the result of one of these methods can be a starting point to *MS01X* which may be able to improve the generalization even further. *MS01X* introduces a soft clause per state variable which is a candidate for removal. If we already have found a set of don't cares while using another method, we only have to introduce soft clauses for the remaining state variables. For all other variables we can introduce the don't care value as a hard clause or assumption, heavily decreasing the search space of the MaxSAT solver. In our experiments we use a heuristics for dynamically combining *MS01X* with *IGBG*. If *MS01X* is successful and not much slower than *IGBG*, we only use *MS01X* in the future. If *MS01X* is much slower and not significantly more successful than *IGBG*,

we use *IGBG* only. In all other cases we use the mentioned combination where *MS01X* improves on a precomputed result of *IGBG*. A similar method could be used with the MaxQBF approach, only with the difference, that MaxQBF is able to improve or at least meet *every* previous result of our different methods.

D. More Degrees of Freedom?

The PO generalization methods discussed so far look for a minimal subcube c of a minterm m (resulting from a satisfied SAT solver call $SAT?[\neg d \wedge R_{k-1} \wedge T \wedge d']$) with the property that $\tilde{m} \wedge T \wedge d'$ is still satisfiable for all minterms $\tilde{m}$ covered by c. Thus those methods are restricted by the initial choice of m and starting from another minterm $\hat{m}$ could lead to a much better solution. Some of the optimization based methods like *S01X*, *MS01X*, SAT-based cover, ILP-based cover, greedy QBF, and *MaxQBF* offer themselves to let the choice of the present state variables open in the optimization. One small additional detail should be considered in this context: In order to minimize the cube c mentioned above as much as possible, the methods based on a fixed initial minterm m allow c to contain states which have already been excluded from R_{k-1} before, but c contains at least one new state not yet excluded from R_{k-1} (at least m). (This is the approach used in the literature on PDR, see e. g., [1], [11]). However, if we let the choice of the present state variables open and only require that $\tilde{m} \wedge T \wedge d'$ is satisfiable for all minterms $\tilde{m}$ covered by c, then it could happen that c contains *only* states which have already been excluded from R_{k-1}, so it could be useless. Therefore we now require that $\tilde{m} \wedge R_{k-1} \wedge T \wedge d'$ is satisfiable for all minterms $\tilde{m}$ covered by c, i. e., we replace T by $R_{k-1} \wedge T$ in those methods.

We call the resulting methods $S01X^{\text{free}}$, $MS01X^{\text{free}}$ etc. to differentiate them from the methods considered so far which are now called $S01X^{\text{fix}}$, $MS01X^{\text{fix}}$ etc. Here we give more details on how to realize the approaches mentioned in Sect. VI-D where the choice of the present state variables is open.

1) MS01X^{free} and S01X^{free}: Compared to $MS01X^{\text{fix}}$ (see Sect. IV-D) the MaxSAT encoding has to be modified in a straightforward manner: The soft clauses $sc_i = \{t_i\}$ and the corresponding hard clauses representing $t_i \leftrightarrow ((s_i^{(0)} = 0) \wedge (s_i^{(1)} = 0))$ remain unchanged. We omit the hard clauses fixing present state bits s_j to X or σ_j. The hard clauses for primary inputs are now for all $1 \leq j \leq n$ $((i_j^{(0)} \vee i_j^{(1)})$ (only disallowing X for i_j). For all $1 \leq j \leq k$ the hard clauses for $((d_{i_j}'^{(0)} = (1 - \tau_j)) \wedge (d_{i_j}'^{(1)} = \tau_j))$ (fixing d_{i_j}' to τ_j) remain unchanged. The hard clauses representing an 01X-encoding of T remain unchanged as well. Finally, we replace in the CNF R_{k-1} each literal s_i by $s_i^{(1)}$ and each literal $\neg s_i$ by $s_i^{(0)}$. We add the resulting clauses to the MaxSAT problem. In that way we enforce that the chosen assignments to the present states are included in of R_{k-1}.

As for $S01X^{\text{fix}}$, $S01X^{\text{free}}$ simply results from $MS01X^{\text{free}}$ by omitting the soft clauses $sc_i = \{t_i\}$ (and their corresponding hard clauses $t_i \leftrightarrow ((s_i^{(0)} = 0) \wedge (s_i^{(1)} = 0)))$.

2) SATCover^{free}: Compared to $SATCover^{\text{fix}}$ (see Sect. V-Bc) we remove the restriction that present state bits s_i can only be

unassigned or equal to the value σ_i fixed by $m = s_1^{\sigma_1} \wedge \ldots \wedge s_m^{\sigma_m}$, i.e., we just have to remove the assignment of $s_i^{(1-\sigma_i)}$ to 0.

3) ILP^{free}: Compared to *ILP*$^{\text{fix}}$ (see Sect. V-Bb) it is also possible to arrive at *ILP*$^{\text{free}}$ if we do not confine ourselves to the assignment A but much rather allow the ILP solver to choose between both polarities of variables.

For ILP$^{\text{free}}$ we introduce two binary ILP-variables v_l and $v_{\bar{l}}$ for each variable v in $\neg d \wedge R_{k-1} \wedge T \wedge d'$, one ILP variable for each literal. Γ contains the clauses representing $\neg d \wedge R_{k-1} \wedge T \wedge d'$. We introduce a linear constraint $\sum_{i=1}^{n} v_{l_i} \geq 1$ for each clause $c = \{l_1, \ldots, l_n\}$ of Γ. Now we have to assert that only one polarity for each variable is used by introducing the linear constraints $v_l + v_{\bar{l}} \leq 1$. As an optimization goal we minimize the sum over all ILP-variables corresponding to present state variables. If in the solution to ILP$^{\text{free}}$ both variables corresponding to literals of a present state variable s_i are assigned to 0, then the computed generalized cube c does not contain a literal for s_i. The *binate covering* ILP-problem now has to solve the original SAT-problem again, but with an additional optimization component, and is therefore more complex. Nevertheless it may find a smaller partial assignment.

4) GreedyQBF^{free}: As *GreedyQBF*$^{\text{fix}}$ (see Sect. V-Ca), *GreedyQBF*$^{\text{free}}$ partitions the present state variable into a set of variables $\vec{r}$ which are removed from the cube representing the PO and a set of variables $\vec{k}$ remaining in the PO. Whereas in *GreedyQBF*$^{\text{fix}}$ the QBF formula is $\forall \vec{r} \exists \vec{k} \exists \vec{i} \exists \vec{s}' : \hat{m} \wedge T \wedge d'$, we remove in *GreedyQBF*$^{\text{free}}$ the fixing of the $\vec{k}$ variables to $\hat{m}$. Moreover, we use the formula $\neg d \wedge R_{k-1} \wedge T \wedge d'$ as in the original SAT problem leading to $\exists \vec{k} \forall \vec{r} \exists \vec{i} \exists \vec{s}' : \neg d \wedge R_{k-1} \wedge T \wedge d'$.

5) MaxQBF^{free}: Compared to *MaxQBF*$^{\text{fix}}$ (see Sect. V-Cb), in *MaxQBF*$^{\text{free}}$ we introduce a new *existentially* quantified variable $s_i^{\exists}$ for one input of the multiplexer, instead of using the original assignment ε from m. In such a modified MaxQBF problem *MAXQBF*$^{\text{free}}$ we conjoin with $\bigwedge_{i=0}^{n} C_i^{\exists}$ instead of $\bigwedge_{i=0}^{n} C_i^{\varepsilon}$ whereas $C_i^{\exists} = \neg u_i \rightarrow (s_i \leftrightarrow s_i^{\exists})$.

For *MAXQBF*$^{\text{free}}$ we use the formula $\neg d \wedge R_{k-1} \wedge T \wedge d'$ as in the original SAT problem, i.e., the MaxQBF query is $\exists \vec{u} \exists \vec{s}^{\exists} \forall \vec{s}^{\forall} \exists \vec{s} \exists \vec{i} \exists \vec{s}' : \neg d \wedge R_{k-1} \wedge T(\vec{s}, \vec{i}, \vec{s}') \wedge d' \wedge \bigwedge_{i=0}^{n} C_i^{\forall} \wedge \bigwedge_{i=0}^{n} C_i^{\exists} \wedge \bigwedge_{i=0}^{n} u_i$. This query yields a PO with a minimal number of state variables (which still satisfies $\neg d \wedge R_{k-1}$ and) which is not constrained to m as a 'starting point'.

VII. EXPERIMENTAL RESULTS

We divide the experimental results into *two sections*. In the first one, Sect. VII-A, we discuss the results on Hardware Model Checking Competition (HWMCC) benchmarks, in the second one, Sect. VII-B, the results on AI Planning benchmarks of the International Planning Competition (IPC). For each benchmark, we limited the execution time to 3,600 s and set a memory limit of 7 GB. We used one core of an Intel Xeon CPU E5-2650v2 with 2.6 GHz. We provide our binaries and results under [42].[3]

---

[3]If the paper will be accepted, we will also share the sources of our implementation with the scientific community to provide a broad basis for experiments with different PO generalization techniques.

A. Hardware Model Checking

Our PDR implementation is based on *ic3ref* [43] and augmented to support Reverse PDR too. Unless the order by which literals are removed from POs is given by the definition of the algorithm (as in greedy covering for instance), we always consider the variable order as implemented in ic3ref, which uses an activity-based order (literals which could be removed more often are preferred). Furthermore, we leave the *clause* generalization part of ic3ref which uses *ctgDown* from [13] completely untouched. All experiments have been performed on the complete benchmark set of HWMCC'15 [15] and '17 [16] excluding the access restricted Intel benchmarks (730 instances) and on the subset of HWMCC'19 [17] bit-vector benchmarks containing invariant constraints (231 instances). If a generalization technique requires a SAT solver, we stick to MINISAT v2.2.0 [44] as used in ic3ref. As MaxSAT solver we use *Pacose* [45] with Glucose 4 [46] and DGPW encoding [47], as QBF solver *DepQBF* [48] with incremental solving. The used MaxQBF solver is *quantom*, an extension of *antom* [49] which is a rather experimental implementation on top of a search-based QBF solver. Finally, *gurobi9.02* [50] is used for ILP-solving.

Structure of the Section: We structure our experiments on Hardware Model Checking benchmarks as follows. In a first set of experiments we considered the original (forward) PDR without invariant constraints applied to HWMCC benchmarks (see Sect. VII-A1).

- In the first experiment, we compared the *generalization capabilities* of all techniques. To enable a fair evaluation of all methods by comparing them on exactly the same problems, we extracted *single* PO generalization problems (POGPs) from PDR runs on HWMCC benchmarks (see Sect. VII-A1a).
- Next, we analyze the different methods within full PDR runs (see Sect. VII-A1b and Sect. VII-A1c).

In Sect VII-A2, we consider the special case of left-unique transition relations (see Sect. VI-B) with the example of Reverse PDR. The experiments refer to full PDR runs.

Finally, in Sect. VII-A3, we examine full forward (standard) PDR runs on AIGER 1.9 [39] Benchmarks with invariant constraints that invalidate left-totality.

1) Original PDR:

a) Single PO generalization problems: For comparison, we considered only POGPs which (a) allow for PO generalization and (b) can be solved by all methods (including MaxQBF). The problems were extracted during a MaxQBF run, i.e., the cubes of next state POs d' in $SAT?[\neg d \wedge R_{k-1} \wedge T \wedge d']$ are minimal. We randomly picked 258 such POGPs. In Table I we present both the average reduction ratio and the average quality of the methods. The reduction ratio for a POGP is the number of removed state bits divided by the total number of state bits. The quality of a method for a single POGP relates its reduction capability to that of MaxQBF which produces an optimal result, i.e., the quality of a method on a POGP is just the quotient of the number of removed state bits for this method and the number of removed state bits for the optimal MaxQBF (which achieves the optimal quality of 100%

		GeNTR	Greedy Cover	ILP Cover	SAT Cover	01X sim.	S01X	MS01X	IGBG	Justification	Lifting	Lifting + lit.drop.	Greedy QBF	MaxQBF
fix	Reduction	1.07	1.91	4.36	4.48	10.76	10.54	10.84	9.43	10.76	9.41	11.58	14.73	14.85
	Quality	7.7	9.6	39.3	37.2	59.0	55.3	59.2	55.7	59.0	55.5	65.8	99.8	100
free	Reduction	-	-	6.09	4.03	-	8.68	9.96	-	-	-	-	12.35	12.68
	Quality	-	-	54.0	42.9	-	56.3	67.6	-	-	-	-	97.3	100

TABLE I: Reduction ratio (in %) and quality (in %) for *fix* and *free* variants.

by definition). Table I shows average numbers for all 258 random POGPs. In Table I we differentiate between the variants starting with a fixed minterm m from a satisfying assignment (called *fix* variants) and the variants which are not restricted by the initial choice of m (called *free* variants, see Sect. VI-D). Interestingly, the *free* variants mostly achieve worse results than their corresponding *fix* variant. The effect of the restriction that the resulting PO has to be completely included in R_{k-1} in the *free* variants (see Sect. VI-D) apparently outweighs the independence from the initial choice of m. For the *fix* variants we observe e.g. that the covering approaches are not very suitable for HWMCC benchmarks (especially greedy cover and GeNTR). 01X-simulation has a slightly better quality than lifting (in spite of its theoretical inferiority due to the imprecision of ternary logic). This is however compensated, if lifting is extended by literal dropping. The quality of 01X-based methods, IGBG, and lifting based methods lies between 55 and 66% of the optimal quality of MaxQBF which shows the potential of more exact methods for PO generalization. Interestingly, greedy QBF achieves already 99.8% of the MaxQBF quality.

b) Full PDR runs, overall performance: Note that within full PDR runs the single POGPs become different due to different generalization results and runtimes of the generalization methods become relevant as well. In Fig. 5 we restrict the presentation to the methods with the most promising performance on the set of HWMCC'15/17 Benchmarks (in particular, we do not show any *free* variants).

We start with comparing methods from each category to determine their respective strongest methods. As categories, we consider *lifting* with its different configurations, *01X-based methods* (IGBG, 01X-simulation, S01X, MS01X, and justification), and *cover approaches* (ILP, SAT cover, greedy cover, GeNTR).

In the lifting category, we compare standard lifting from Sec. IV-A3 against variants using additional literal dropping as well as literal rotation. Since exhaustive literal dropping is very costly, we additionally try two heuristics to trade reduction ratio against runtime. Firstly, we use approximate SAT with a decision limit of 100 – as proposed in [12]. Secondly, we limit the number of overall attempts to drop a literal to 32 and the number of failed attempts to 2. After each successful attempt, we reset the current count of failed attempts to 0 and shuffle the remaining literals randomly. We copied this strategy from TIP[4] [51] and therefore denote it by 'TIP-like'. Regarding literal rotation, we also implemented a version which acts 'TIP-like'. We consider one attempt of literal rotation as failed, if it has not been able to remove any literal from the UNSAT core. The first plot of Fig. 5 (starting at the top) displays all results from the lifting category. Apparently, literal dropping as well as literal rotation does not pay off when performed exhaustively. Literal rotation is performing better than literal dropping, most likely because of the relatively cheap unsatisfiable SAT solver queries (see Sect. IV-B2). Forcing MINISAT to report 'satisfiable' after 100 decisions (approximate SAT), is able to increase the performance of literal dropping but is still not able to outperform literal rotation. Imposing hard bounds on the number of literal dropping attempts (and fails) however, yields significantly better results. We make the same observation for literal rotation. The two 'TIP-like' configurations of literal rotation and literal dropping are able to improve the overall performance of the 'standard technique' of lifting.

The second plot of Fig. 5 displays all results of the 01X-based methods. Interestingly, *S01X*, the *justification* approach from ABC, as well as *IGBG* and the heuristics based on *MS01X / IGBG* significantly outperform the 'standard technique' of greedy 01X-simulation. *IGBG* and its combination with *MS01X* performs best. Regarding *S01X*, we also tried to alter the decision heuristics of MINISAT to decide state variables first. However, we did not observe any relevant difference with respect to our results.

The third plot of Fig. 5 displays all results of the cover approaches. For *ILP cover* and *greedy cover* the relation between quality and computational cost of PO generalization is apparently not beneficial enough as they perform worse than doing no PO generalization at all – at least on Hardware Model Checking problems. A possible reason could be its cost inefficiency which is discussed in more detail in Sect. VII-A1c. In the category of cover approaches, PO generalization with *SAT cover* solves most instances, followed by *GeNTR*.

For the results displayed by the plot at the bottom of Fig. 5 we picked the best performing technique from each category. The heuristics based on *MS01X / IGBG* outperforms the best approach based on *lifting*.

In particular, it is also interesting to observe (by comparing the different plots) that MS01X / IGBG as well as IGBG outperform the 'standard techniques', i.e., lifting and greedy 01X-simulation. Additionally, our results indicate that we should always prefer 01X-based or lifting-based methods to cover approaches – whenever the transition relation's properties allow for it.

The different techniques show a great variety of uniquely solved benchmarks as displayed by the overall 'virtual best' at the bottom of Fig. 5 as well as the 'virtual best' of each category. (The 'virtual best' approach corresponds to running all methods in parallel and counting a benchmark as solved as soon as at least one method solved it.)

We further refer to Table II were we split the results of the best performing techniques of each category into counterex-

---

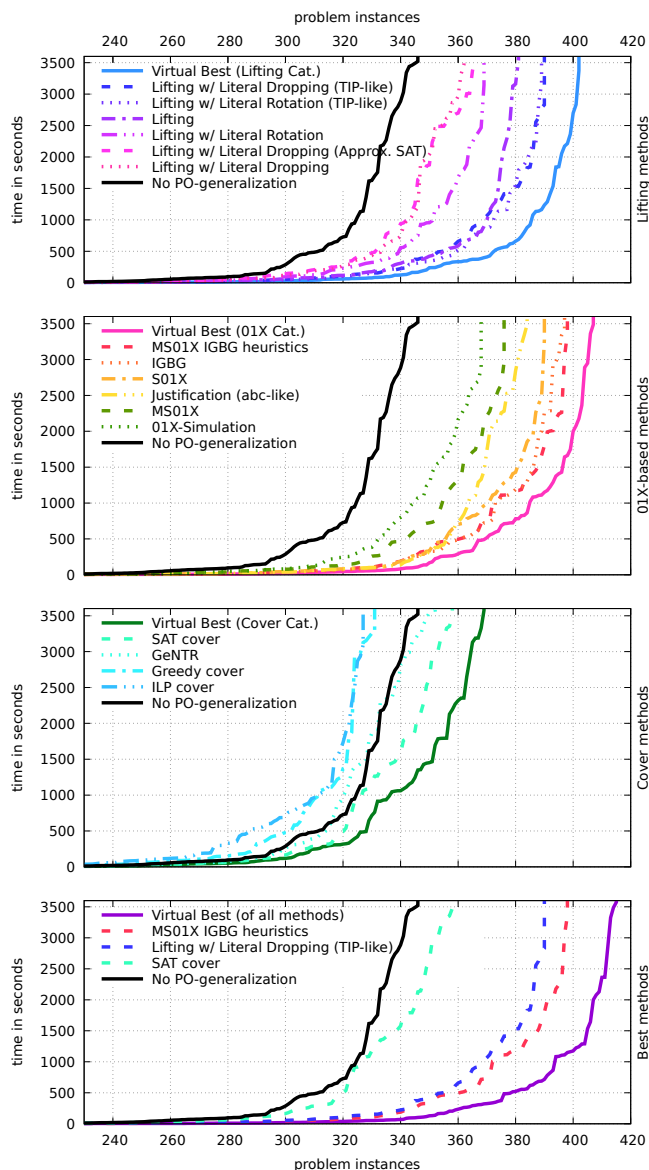[4]Downloaded from https://github.com/niklasso/tip, Sept. 2021

Fig. 5: Original PDR on all techniques from 1) the lifting, 2) the 01X-based, and 3) the cover approaches. The last plot displays the best of all three categories. HWMCC'15/17 Benchmarks.

	No PO-Gen.	SAT Cover	Lifting lit.drop. (TIP)	MS01X / IGBG heur.
SAT (Counterexample)	100	101	113	**118**
UNSAT (Safe)	246	257	276	**280**

TABLE II: Detailed results of best performing techniques from Fig. 5.

 amples (SAT) and proofs of safety (UNSAT). Apparently, PO generalization speeds up both SAT and UNSAT. The results also indicate, that the MS01X / IGBG heuristics is stronger than the other variants on both benchmark categories.

c) Full PDR runs, execution times, PO reduction ratios, and number of PO generalization attempts: In Table III we present the fraction of the overall execution time that the methods considered in Fig. 5 required during *full* PDR runs (i. e., the time required for generalization divided by the overall execution time). Moreover, we give information on their reduction ratios, and the number of generalization attempts

during full PDR runs. We report on average and median values, as well as the standard deviation. For computing the reduction ratios and generalization attempts we consider only benchmarks solved by all those methods. It is interesting to correlate the data in Table III with the overall performance data in Fig. 5. For instance, *ILP cover* and *greedy cover* need a large fraction of the overall runtime without providing high reduction rates which explains their poor overall performance. Both lifting and *IGBG* are very runtime efficient (their average fractions of execution times are 2.2 % resp. 2.0 %) with good reduction rates, whereas the advantage of *IGBG* can be explained by its higher PO reduction ratios. Enhancing lifting with restricted ('TIP-like') literal rotation leads to a moderate increase in the fraction of runtime for PO generalization, but pays-off by an increased reduction rate (and an improved overall performance as well). *MS01X* shows the best average reduction rate (56.5 %), but has high average execution times (60.2 %). For this reason *MS01X* does not have the best overall performance as can be seen in Fig. 5. Using the *MS01X / IGBG* heuristics successfully reduces the average fraction of execution time to 20.5 % while keeping a high average reduction ratio of 55.0 %. By this the *MS01X / IGBG* heuristics ends up with the best overall performance. In general, lower reduction ratios seem to significantly increase the total number of required PO generalization attempts – as can be observed for all cover approaches in particular. Overall efficiency, however, very much depends on how costly these PO generalizations are in terms of runtime. An example would be MS01X which achieves the lowest number of PO generalization calls (median and average) due to its strong reduction ratio – since these calls are very costly though, it is still outperformed by more cost-efficient techniques. The large standard deviations as well as the differences between average and median values (especially for the number of PO generalization calls) show that the HWMCC benchmark set is pretty diverse. It apparently contains a large fraction of benchmarks which are easy for PDR, but contains difficult instances as well.

Note that – compared to the study with single POGPs – the average reduction ratios arrive at much higher values. This can be explained by the fact that for single POGPs we only considered instances which can also be solved by the MaxQBF approach. High reduction ratios result in a high number of satisfied MaxQBF soft clauses. A growing number of such clauses is quite challenging for the incorporated experimental MaxQBF solver leading to timeouts on many HWMCC benchmarks.

2) Reverse PDR: We recall from Sect. VI-B that generalization of POs in Reverse PDR is only possible with both the QBF-based methods and the structural method from [20].

a) Full Reverse PDR runs, overall performance: In the case of Reverse PDR, the structural generalization method outperforms both *MaxQBF* and *greedy QBF* in terms of solved instances. The structural method even completely dominates *greedy QBF*. This can be explained by the large computational effort invested by the stronger generalization methods. Though there exist some benchmarks where $MaxQBF^{free}$ performs better than the structural approach. Two of them cannot even be solved within the timeout using the structural approach. For

		Lifting	Lifting lit.drop. (TIP)	Lifting lit.drop.	Lifting lit.rot. (TIP)	Lifting lit.rot.	IGBG	MS01X	MS01X /IGBG heur.	S01X	Justifi- cation	01X Sim.	Greedy Cover	GeNTR	ILP	SAT Cover
% Exe. Time	Avg.	2.2	9.4	45.1	4.0	24.7	2.0	60.2	20.5	5.1	12.4	46.3	39.6	8.8	79.7	6.3
	Median	1.2	3.3	31.7	0.8	7.7	1.2	44.7	20.5	3.4	3.0	5.1	5.8	2.6	84.0	3.5
	Std. Dev.	1.9	8.6	29.5	6.5	27.8	19.2	26.0	22.7	8.5	14.5	26.0	11.8	9.2	23.9	6.5
% Red. Ratio	Avg.	50.0	53.5	56.9	54.4	54.4	53.0	56.5	55.0	51.9	54.9	54.9	13.6	9.1	17.3	16.0
	Median	58.0	60.7	67.5	62.9	62.9	60.6	66.0	64.5	57.9	62.6	62.7	5.8	1.4	9.6	9.4
	Std. Dev.	33.5	34.0	33.8	34.2	34.2	34.4	34.4	34.1	33.2	34.1	34.3	17.9	14.0	19.8	18.2
#Calls	Avg.	667.8	676.3	641.2	651.2	651.2	660.1	612.1	649.1	720.9	704.5	667.1	872.6	979.7	931.6	913.5
	Median	84	77	64	73	73	62	60	62	70	64	71	168	150	143	158
	Std. Dev.	1734.1	1794.4	1960.6	1676.5	1676.5	1757.4	1632.3	1755.1	2131.2	2140.8	2053.0	1889.4	2062.0	2141.8	2046.7

TABLE III: Fraction of overall execution time (in %), reduction ratio (in %), and the total number of generalization attempts. Average, Median, and Standard Deviation. Original PDR.

		Struct.	GreedyQBF	MaxQBF^{free}	MaxQBF^{fix}
% Exe. Time	Avg.	16.4	96.3	73.2	67.8
	Median	1.8	94.7	84.5	69.4
	Std. Dev.	8.7	16.0	28.3	33.4
% Red. Ratio	Avg.	8.6	12.8	11.4	12.8
	Median	0.5	3.2	3.2	3.2
	Std. Dev.	8.0	10.6	9.1	10.5
#Calls	Avg.	1481.1	1064.0	1066.0	1057.8
	Median	35	25	25	25
	Std. Dev.	4996.6	3341.2	3203.7	3302.2

TABLE IV: Fraction of overall execution time (in %), reduction ratio (in %), and the total number of generalization attempts. Average, Median, and Standard Deviation. *Reverse PDR*.

Fig. 7: HWMCC'19, with invariant constraints.

Fig. 6: MAXQBF^{free} / Structural [20]

a more detailed comparison of execution times, we refer to Fig. 6.

b) Full Reverse PDR runs, execution times and PO reduction ratios: Table IV shows the average fractions of the overall execution times used by the respective generalization approaches as well as their reduction ratios. The structural approach has the smallest reduction ratios, but (as expected) it requires much less runtime than the QBF / MaxQBF based methods. We note that reduction ratios with Reverse PDR are in general significantly smaller than with original PDR. This can be explained by the limitations of left-unique transition relations (see Sect. VI-B).

3) Invariant Constraints: For standard *ic3ref* and HWMCC'19 benchmarks with invariant constraints we observed incorrect results[5]. This can be explained by the observations on lifting made in Sect. IV-F. So we had to deactivate lifting for *ic3ref* to get correct results. In Fig. 7

[5]For instance: ic3ref reports Unsafe instead of Safe on wolf/2019C/qspiflash qflexpress divfive- p072.aig or wolf/2019C/qspiflash qflexpress divfive-p077.aig from HWMCC'19.
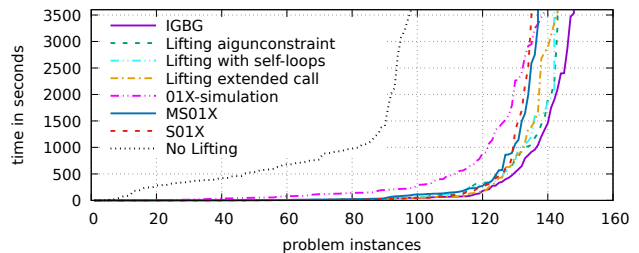
we compare the execution times of the three *lifting* variants from Sect. IV-F against the admissible 01X-techniques and *IGBG* as well as to *ic3ref* with deactivated lifting. By 'Lifting extended call' we denote the extended query $SAT?[m \wedge i \wedge \hat{T} \wedge (\neg C \vee \neg d')]$ without 'repairing' the transition relation. The lifting variants greatly outperform standard *ic3ref* with deactivated lifting. Furthermore, the lifting variants apparently come with only minimal overhead and similar execution times, thus *lifting* outperforms *01X-simulation*, *S01X* and *MS01X*. *IGBG* does not need any changes and achieves the best performance.

B. AI Planning

We adjusted an existing PDR implementation called *minireachIC3* [52] which has been used for planning tasks [8]. We used 1641 STRIPS benchmarks from past IPC events (from 1998 to 2011). To transform the STRIPS benchmarks into the input format (DIMSPEC) of *minireachIC3* we used the ∃-step parallel encoding scheme of the SAT-based planner *Mp* [30] as found best for *minireachIC3* [8]. Since the previous evaluation on IPC benchmarks from [8] came to the conclusion that Reverse PDR (for general transition relations, the reverted direction does not imply left-uniqueness) is the favorable configuration of *minireachIC3*, we also use this configuration.

We compared standard *minireachIC3* (which does not include PO generalization) with the three cover approaches, *GeNTR*, *MaxQBF* as well as *greedy QBF*. We observed that in AI Planning the QBF-based approaches are much more competitive than in Hardware Model Checking. However, many of the planning benchmarks seem to have only little potential for generalization of POs. The best average reduction ratio – achieved by *MaxQBF* – on 695 of 1641 solved IPC benchmarks is 4.01 %.

Nevertheless, we present the overall performance of the different PO generalization techniques for the complete set of

Technique	Solved	SAT	UNSAT	Timeout	Memout
SAT Cover	948	934	14	638	55
Greedy Cover	940	926	14	646	55
Standard	939	925	14	643	59
GeNTR	913	899	14	614	114
ILP Cover	900	886	14	722	19
MaxQBF	695	681	14	927	19
GreedyQBF	646	636	10	968	27

TABLE V: *minireachIC3* on IPC Benchmarks.

Domain	Inst.	Stan-dard	Max QBF	Greedy QBF	Greedy Cover	Ge-NTR	ILP Cover	SAT Cover
2002-DEPOTS	20	4	**8 (4)**	**6 (2)**	4 (0)	4 (0)	4 (0)	4 (1)
2006-PIPESW.	45	6	6 (1)	3	6 (0)	**7 (1)**	4 (0)	6 (1)
2008-SCANALY.	30	22	16 (0)	15 (0)	22 (2)	**23 (2)**	**23 (1)**	**23 (3)**
2011-BARMAN	20	12	**13 (2)**	0	**14 (3)**	12 (0)	7 (2)	10 (6)
2011-FLOORT.	20	17	**20 (3)**	15 (0)	16 (1)	17 (0)	18 (2)	19 (2)
2011-SCANALY.	20	14	9 (0)	6 (0)	12 (0)	13 (0)	**15 (1)**	**15 (3)**

TABLE VI: Solved instances of generalization techniques on selected IPC domains. The number of uniquely solved instances (w. r. t. Standard) is written in parentheses.

benchmarks in Table V. Since many of the planning benchmarks have only little potential for generalization of POs and show only small average reduction ratios, expensive methods like MaxQBF result in high cost for many benchmarks, but do not help much. However, SAT-based cover, which is much less expensive, is able to consistently improve on standard *minireachIC3*. It solves 948 instances overall (compared to 939 instances with standard *minireachIC3*). On the other hand, compared to SAT-based cover, the MaxQBF approach is able to solve 21 unique instances in total and additionally achieves better execution times on a number of benchmarks.

As presented in Table VI there are planning domains – that allow significant reduction of POs – for which the generalization approaches greatly improve the performance of *minireachIC3*: Among the cover approaches and *GeNTR* (which is implicitly a cover approach as well) SAT-based cover is the best and clearly outperforms standard *minireachIC3*. Furthermore, for those planning domains trading computation time against generalization capabilities pays off and MaxQBF performs even better than the cover approaches on three domains. It dominates standard *minireachIC3* on 2002-DEPOTS, 2006-PIPESWORLD, 2011-BARMAN, and 2011-FLOORTILE. Numbers in bold face indicate the best performance for the respective domain.

For a detailed analysis we chose two examples where MaxQBF in *minireachIC3* had a strong effect and analyze further why this is the case.

Firstly, we analyze the problem *depotprob4398* of the IPC domain *DEPOTS-2002*. This could be solved within $51.33\,s$ by using *MaxQBF* for PO generalization, finding a plan of length 13 (longer than the trace, due to PO forward pushing), whereas standard *minireachIC3* ran into timeout $(3,600\,s)$. With *MaxQBF*, only 6 time frames were opened, while the standard version ran into timeout when it was still working on time frame number 9. The version using *MaxQBF* had to learn only $1,226$ clauses with an average size of 17.2, whereas the standard version had $86,932$ learned clauses with an average size of 23.0 after $3,600\,s$.

Secondly, we consider the problem *instance-1* of the IPC domain *BARMAN-2011*. With *MaxQBF* we were able to find a

plan of length 142 after $82.37\,s$, again standard *minireachIC3* ran into timeout $(3,600\,s)$. With *MaxQBF*, *minireachIC3* advanced to time frame 15, without *MaxQBF*, it had 13 open time frames within $3,600\,s$. The *MaxQBF* version required $1,289$ learned clauses with an average size of 19.1 literals, while the standard version reached the timeout with $104,503$ learned clauses having an average size of 20.2 ($103,400$ of them were learned in only two time frames 5 and 6).

VIII. CONCLUSIONS AND FUTURE WORK

We presented a comprehensive study on the generalization of POs in PDR. We discussed the complexity of the problem as well as limitations and applicability of various techniques on different domains. It turned out that techniques which have not been used in PDR so far as well as new and more exact techniques based on MaxSAT are able to beat well-known standard techniques for the generalization of POs like lifting and 01X-simulation. We expect to be able to improve the obtained results even more using further improvements of solver technology like incremental reuse of cardinality constraints in MaxSAT solvers. An exact solution for general transition relations could be provided using a reduction to MaxQBF. Research on MaxQBF solving is still in its infancy and we had to use a rather immature solver for our experiments. We hope that with new applications we can stimulate further research in this direction. We successfully explored the cooperation between different generalization approaches with a combination of *MS01X* and *IGBG*. We believe that there is room for further improvements by combining various techniques. One option is to dynamically switch between different methods based on their success, another option is to perform first generalizations with weaker and less expensive methods, followed by stronger methods building on the results of the former. Moreover, we assume that our results can be useful for other domains requiring SMT with theory reasoning.

REFERENCES

[1] A. R. Bradley, "Sat-based model checking without unrolling," in *VMCAI*, 2011, pp. 70–87.

[2] A. Cimatti and A. Griggio, "Software model checking via IC3," in *CAV*, 2012, pp. 277–293.

[3] K. Hoder and N. Bjørner, "Generalized property directed reachability," in *SAT*, 2012, pp. 157–171.

[4] T. Welp and A. Kuehlmann, "QF BV model checking with property directed reachability," in *DATE*, 2013, pp. 791–796.

[5] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta, "Parameter synthesis with IC3," in *FMCAD*, 2013, pp. 165–168.

[6] J. Birgmeier, A. R. Bradley, and G. Weissenbacher, "Counterexample to induction guided abstraction refinement (CTIGAR)," in *CAV*, 2014, pp. 831–848.

[7] A. Komuravelli, A. Gurfinkel, and S. Chaki, "Smt-based model checking for recursive programs," in *CAV*, 2014, pp. 17–34.

[8] M. Suda, "Property directed reachability for automated planning." *J. Artif. Intell. Res.(JAIR)*, pp. 265–319, 2014.

[9] K. Suenaga and T. Ishizawa, "Generalized property-directed reachability for hybrid systems," in *VMCAI*, 2020, pp. 293–313.

[10] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *TACAS*, 1999, pp. 193–207.

[11] N. Eén, A. Mishchenko, and R. K. Brayton, "Efficient implementation of property directed reachability," in *FMCAD*, 2011, pp. 125–134.

[12] A. Griggio and M. Roveri, "Comparing different variants of the ic3 algorithm for hardware model checking," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 35, no. 6, pp. 1026–1039, 2016.

[13] Z. Hassan, A. R. Bradley, and F. Somenzi, "Better generalization in IC3," in *FMCAD*, 2013, pp. 157–164.

[14] A. Ivrii and A. Gurfinkel, "Pushing to the top," in *FMCAD*, 2015, pp. 65–72.

[15] A. Biere and K. Heljanko, "Hardware model checking competition," 2015. [Online]. Available: http://fmv.jku.at/hwmcc15/

[16] A. Biere, T. van Dijk, and K. Heljanko, "Hardware model checking competition," 2017. [Online]. Available: http://fmv.jku.at/hwmcc17/

[17] A. Biere and M. Preiner, "Hardware model checking competition," 2019. [Online]. Available: http://fmv.jku.at/hwmcc19/

[18] H. Chockler, A. Ivrii, A. Matsliah, S. Moran, and Z. Nevo, "Incremental formal verification of hardware," in *FMCAD*, 2011, pp. 135–143.

[19] 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.

[20] T. Seufert and C. Scholl, "Combining PDR and reverse PDR for hardware model checking," in *DATE*, 2018, pp. 49–54.

[21] ——, "fbpdr: In-depth combination of forward and backward analysis in property directed reachability," in *DATE*, 2019, pp. 456–461.

[22] K. Ravi and F. Somenzi, "Minimal assignments for bounded model checking," in *TACAS*, 2004, pp. 31–45.

[23] J. Roorda and K. Claessen, "Sat-based assistance in abstraction refinement for symbolic trajectory evaluation," in *CAV*, 2006, pp. 175–189.

[24] A. Franzén, "Efficient solving of the satisfiability modulo bit-vectors problem and some extensions to SMT," Ph.D. dissertation, University of Trento, Italy, 2010.

[25] D. Déharbe, P. Fontaine, D. L. Berre, and B. Mazure, "Computing prime implicants," in *FMCAD*, 2013, pp. 46–52.

[26] A. Niemetz, M. Preiner, and A. Biere, "Turbo-charging lemmas on demand with don't care reasoning," in *FMCAD*, 2014, pp. 179–186.

[27] M. Sauer, S. Reimer, I. Polian, T. Schubert, and B. Becker, "Provably optimal test cube generation using quantified boolean formula solving," in *ASP-DAC*, 2013, pp. 533–539.

[28] S. Reimer, M. Sauer, P. Marin, and B. Becker, "QBF with soft variables," *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, vol. 70, 2014.

[29] K. Scheibler, D. Erb, and B. Becker, "Accurate CEGAR-based ATPG in presence of unknown values for large industrial designs," in *DATE*, 2016, pp. 972–977.

[30] J. Rintanen, "Planning as satisfiability: Heuristics," *Artif. Intell.*, vol. 193, pp. 45–86, 2012.

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

[32] J. P. M. Silva and K. A. Sakallah, "GRASP - A New Search Algorithm for Satisfiability," in *ICCAD*, 1996, pp. 220–227.

[33] R. K. Brayton and A. Mishchenko, "ABC: an academic industrial-strength verification tool," in *CAV*, 2010, pp. 24–40.

[34] K. Scheibler, F. Winterer, T. Seufert, T. Teige, C. Scholl, and B. Becker, "ICP and IC3," in *DATE*, 2021.

[35] Z. Fu and S. Malik, "On solving the partial max-sat problem," in *SAT*, 2006, pp. 252–265.

[36] C. M. Li and F. Manya, "Maxsat, hard and soft constraints." *Handbook of satisfiability*, vol. 185, pp. 613–631, 2009.

[37] C. Ansótegui, M. L. Bonet, and J. Levy, "Sat-based maxsat algorithms," *Artificial Intelligence*, vol. 196, pp. 77–105, 2013.

[38] A. Jain, V. Boppana, R. Mukherjee, J. Jain, M. Fujita, and M. S. Hsiao, "Testing, verification, and diagnosis in the presence of unknowns," in *VTS*, 2000, pp. 263–270.

[39] A. Biere, K. Heljanko, and S. Wieringa, "Aiger 1.9 and beyond," *Available at fmv. jku. at/hwmcc11/beyond1. pdf*, 2011.

[40] R. M. Karp, "Reducibility among combinatorial problems," in *Proceedings of a symposium on the Complexity of Computer Computations*, ser. The IBM Research Symposia Series, 1972, pp. 85–103.

[41] A. Ignatiev, M. Janota, and J. Marques-Silva, "Quantified maximum satisfiability: - A core-guided approach," in *SAT*, 2013, pp. 250–266.

[42] T. Seufert, F. Winterer, C. Scholl, K. Scheibler, T. Paxian, and B. Becker, "Results, benchmarks and binaries," 2022. [Online]. Available: https://nc.informatik.uni-freiburg.de/index.php/s/BskzZx7fgRNq2D7

[43] A. Bradley, "Ic3 reference implementation," 2013. [Online]. Available: https://github.com/arbrad/IC3ref

[44] N. Eén and N. Sörensson, "An extensible SAT-solver," in *SAT*, 2003, pp. 502–518.

[45] T. Paxian, S. Reimer, and B. Becker, "Pacose: An iterative sat-based maxsat solver," *MaxSAT Evaluation 2019*, p. 9, 2019.

[46] G. Audemard and L. Simon, "On the glucose SAT solver," *International Journal on Artificial Intelligence Tools*, vol. 27, no. 01, p. 1840001, 2018.

[47] T. Paxian, S. Reimer, and B. Becker, "Dynamic polynomial watchdog encoding for solving weighted maxsat," in *SAT*, 2018, pp. 37–53.

[48] F. Lonsing and A. Biere, "Depqbf: A dependency-aware QBF solver," *J. Satisf. Boolean Model. Comput.*, vol. 7, no. 2-3, pp. 71–76, 2010.

[49] T. Schubert, M. Lewis, and B. Becker, "Antom – solver description," *SAT Race*, 2010.

[50] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2020. [Online]. Available: http://www.gurobi.com

[51] N. Eén and N. Sörensson, "Temporal induction by incremental SAT solving," *Electron. Notes Theor. Comput. Sci.*, vol. 89, no. 4, pp. 543–560, 2003.

[52] M. Suda, "minireachic3," 2014. [Online]. Available: https://github.com/quickbeam123/minireachIC3