Inferring Invariants with Quantifier Alternations: Taming the Search Space Explosion*

Jason R. Koenig[1] (✉), Oded Padon[2], Sharon Shoham[3], and Alex Aiken[1]

[1] Stanford University, Stanford, CA, USA {jrkoenig,aaiken}@stanford.edu
[2] VMware Research, Palo Alto, CA, USA oded.padon@gmail.com
[3] Tel Aviv University, Tel Aviv, Israel sharon.shoham@gmail.com

Abstract. We present a PDR/IC3 algorithm for finding inductive invariants with quantifier alternations. We tackle scalability issues that arise due to the large search space of quantified invariants by combining a breadth-first search strategy and a new syntactic form for quantifier-free bodies. The breadth-first strategy prevents inductive generalization from getting stuck in regions of the search space that are expensive to search and focuses instead on lemmas that are easy to discover. The new syntactic form is well-suited to lemmas with quantifier alternations by allowing both limited conjunction and disjunction in the quantifier-free body, while carefully controlling the size of the search space. Combining the breadth-first strategy with the new syntactic form results in useful inductive bias by prioritizing lemmas according to: (i) well-defined syntactic metrics for simple quantifier structures and quantifier-free bodies, and (ii) the empirically useful heuristic of preferring lemmas that are fast to discover. On a benchmark suite of primarily distributed protocols and complex Paxos variants, we demonstrate that our algorithm can solve more of the most complicated examples than state-of-the-art techniques.

Keywords: invariant inference · quantifier alternation · PDR/IC3

1 Introduction

Invariant inference is a long-standing problem in formal methods, due to the desire for verified systems without the cost of manually writing invariants. For complex unbounded systems the required invariants often involve quantifiers, including quantifier alternations. For example, an invariant for a distributed system may need to quantify over an unbounded number of nodes, messages, etc. Furthermore, it may need to nest quantifiers in alternation (between $\forall$ and $\exists$) to capture the system's correctness arguments. For example, one crucial invariant of the Paxos consensus protocol [22] is "every decision must come from a quorum of votes", i.e. $\forall decision.\exists quorum.\forall node.\ node \in quorum \Rightarrow node\ voted\ for\ decision.$

* This research received funding from the European Union's Horizon 2020 research and innovation programme grant No. [759102-SVIS], BSF grant No. 2016260, ISF grant No. 1810/18, and NSF grants CCF-1160904 and CCF-1409813.

D. Fisman and G. Rosu (Eds.): TACAS 2022, LNCS 13243, pp. 338–356, 2022.
https://doi.org/10.1007/978-3-030-99524-9_18

We show that automatically inferring such invariants is possible for systems beyond the current state of the art by addressing several scalability issues that arise as the complexity of systems and their invariants increases.

Many recent successful invariant inference techniques, including ours, are based on PDR/IC3 [3,5]. PDR/IC3 is an algorithmic framework for finding inductive invariants *incrementally*, rather than attempting to find the entire inductive invariant at once. PDR/IC3 progresses by building a collection of *lemmas*, organized into *frames* labeled by number of steps from the initial states, until eventually some of these lemmas form an inductive invariant. New lemmas are generated by *inductive generalization*, where a given (often backward reachable) state is generalized to a formula that excludes it and is inductive *relative* to a previous frame. Inductive generalization therefore plays a key role in PDR/IC3 implementations. Specifically, extending PDR/IC3 to a new domain of lemmas requires a suitable inductive generalization procedure.

Techniques for inductive generalization, and more broadly for generating formulas for inductive invariants, are varied, including interpolation [25], quantifier elimination [20], model-based techniques [18], and syntax guided synthesis [6,31]. Almost all of these existing techniques target either quantifier-free or universally quantified invariants. While it is sometimes possible to manually transform a transition system to eliminate some of the need for quantifiers [8], doing so is difficult and requires some knowledge of the fully quantified invariant.

We present a system that can infer quantified invariants with alternations based on *quantified separation*, which was introduced in [19]. Roughly, a separation query asks whether there is a quantified formula, a *separator*, that evaluates to true on a given set of models and to false on another given set of models. While [19] used separation (as a black box) to implement inductive generalization and described the first PDR/IC3 implementation that finds invariants with quantifier alternations, it did not scale to challenging protocols such as Paxos and its variants. These protocols require invariants with many symbols and quantifiers, and the search space for quantified separators explodes as the number of symbols in the vocabulary and number of quantifiers increases. In contrast, this work presents a technique that can automatically find such complex invariants.

When targeting complex invariants, there are two main challenges for inductive generalization: (i) the run time of each individual query; and (ii) overfitting, i.e., learning a lemma that eliminates the given state but does not advance the search for an inductive invariant. We tackle both problems via two strategies: the first integrates inductive generalization with separation in a breadth-first way, and the second defines a new form, k-term pDNF, for the quantifier-free Boolean structure of the separators.

Integrating quantified separation with inductive generalization enables us to effectively use a breadth-first rather than a depth-first search strategy for the quantifiers of potential separators: we search in multiple parts of the search space simultaneously rather than exhaustively exploring one region before moving to the next. Beyond enabling parallelism, and thus faster wall-clock times, this restructuring can change which solution is found by allowing easy-to-search re-

gions to find a solution first. We find that these easier-to-find formulas generalize better (i.e., avoid overfitting).

Using k-term pDNF narrows the search space for lemmas with quantifier alternations. Universally quantified invariants can be split into universally quantified clauses by transformation into conjunctive normal form (CNF). Accordingly, most PDR/IC3 based techniques find invariants as conjunctions of possibly quantified clauses. However, invariants with quantifier alternations may require conjunction inside quantified lemmas (e.g., consider $\forall x.\exists y.p(y) \wedge r(x,y)$). Using multiple clauses per lemma (k-clause CNF) creates a significantly larger search space, impeding scalability. Using disjunctive normal form (DNF) suffers from the same problem. We introduce k-term pDNF, a class of Boolean formulas inspired by human-written invariants that allows both limited conjunction and disjunction while keeping the search space manageable. Many of the lemmas arising in our evaluation that require many clauses in CNF are only 2-term pDNF. We modify separation to search for lemmas of this form, leading to a reduced search space compared to CNF or DNF, resulting in both faster inductive generalization and less overfitting.

We evaluate our technique on a benchmark suite that includes challenging distributed protocols. Inferring invariants with quantifier alternations has recently drawn significant attention, with recent works, [19,11], presenting techniques based on PDR/IC3 that find invariants with quantifier alternations but do not scale to complex protocols such as Paxos. Very recently, [14] and [12] presented enumeration-based and PDR/IC3-based techniques, respectively, which find the invariant for simple variants of Paxos, but do not scale to more complex variants. Our experiments show that our separation-based approach significantly advances the state-of-the-art, and scales to several Paxos variants which are unsolved by prior works. We also present an ablation study that investigates the individual effect of key features of our technique.

This work makes the following contributions:

1. An algorithm for inductive generalization in PDR/IC3 (Section 3) based on quantified separation that explores the search space in a parallel, breadth-first way and thus focuses on lemmas that are easy to discover without requiring *a priori* knowledge of the search space.
2. A syntactic form of lemmas (k-pDNF, Section 4) that is well-suited for invariants with quantifier alternations.
3. A combined system (Section 5) able to infer the invariants of challenging protocols with quantifier alternations, including complex Paxos variants.
4. A comprehensive evaluation (Section 6) on a large benchmark suite including complex Paxos variants, comparisons with a variety of state-of-the-art tools, and an ablation study exploring the effects of key features of our technique.

2 Background

We review first-order logic, quantified separation, the invariant inference problem, and PDR/IC3.

First-Order Logic. We consider formulas in many-sorted first-order logic with uninterpreted functions and equality. A *signature* consists of a finite set of sorts and sorted constant, relation, and function symbols. A first-order *structure* over a given signature consists of a *universe* set of sorted elements along with interpretations for each symbol. A structure is finite when its universe is finite. We use the standard definitions for *term*, *atomic formula*, *literal*, *quantifier-free formula*. *Quantified* formulas may contain universal ($\forall$) and existential ($\exists$) quantifiers with sorted variables (e.g. $\forall x{:}s_1.\, p$). A formula is in *prenex normal form* if it consists of a (possibly empty) quantification *prefix* followed by a quantifier-free *matrix*. Any formula can be mechanically transformed into an equivalent prenex formula. A structure M satisfies a formula p, written $M \models p$, if the formula is true when the symbols in p are interpreted according to M under the usual semantics. If such an M exists, then p is *satisfiable* and M is a *model* of p.

Quantified Separation. To generate candidate lemmas, we use *quantified separation* [19]. Given a set of *structure constraints* and a predetermined space of formulas, separation produces a separator formula p from the space that satisfies the constraints, or reports UNSEP if no such p exists. The constraints are either *positive* (a structure M where $M \models p$), *negative* (a structure M where $M \not\models p$) or *implication* (a pair of structures M, M' where $M \models p \Rightarrow M' \models p$). Separation producing prenex formulas under some assumptions (satisfied by practical examples) is NP-complete [19], and can be solved by translation to SAT.

Invariant Inference. The invariant inference problem is to compute an *inductive invariant* for a given *transition system*, which shows that only *safe* states are reachable from the *initial* states. We consider a transition system to be a set of *states* as structures over some signature satisfying an axiom Ax, some initial states satisfying *Init*, a transition formula *Tr* which can contain primed symbols (x') representing the post-state, and safe states satisfying *Safe*. We define *bad* states as $\neg Safe$. We define single-state implication, written $A \Rightarrow B$, as $\textsc{Unsat}(A \wedge Ax \wedge \neg B)$ and two-state implication across transitions, written $A \Rightarrow \mathrm{wp}(B)$,[4] as $\textsc{Unsat}(A \wedge Ax \wedge Tr \wedge Ax' \wedge \neg B')$. An *inductive invariant* is a formula I satisfying:

$$Init \Rightarrow I \qquad (1) \qquad I \Rightarrow \mathrm{wp}(I) \qquad (2) \qquad I \Rightarrow Safe \qquad (3)$$

Together, (1) and (2) mean that I is satisfied by all reachable states, and (3) ensures the system is safe. We only consider invariant inference for safe systems.

PDR/IC3. PDR/IC3 is an invariant inference algorithm first developed for finite state model checking [3] and later extended to various classes of infinite-state systems. We describe PDR/IC3 as in [17]. PDR/IC3 maintains *frames* F_i as conjunctions of formulas (*lemmas*) representing overapproximations of the states reachable in at most i transitions from *Init*. Finite frames ($i = 0, \ldots, n$) and the frame at infinity ($i = \infty$) satisfy:

[4] Our use of wp is inspired by predicate transformers, but we define it via satisfiability.

$$Init \Rightarrow F_0 \quad (4) \qquad F_i \Rightarrow F_{i+1} \quad (5) \qquad F_n \Rightarrow F_\infty \quad (6)$$
$$F_i \Rightarrow \text{wp}(F_{i+1}) \quad (7) \qquad F_\infty \Rightarrow \text{wp}(F_\infty) \quad (8)$$

Conditions (4), (5), and (6) mean $Init \Rightarrow F_i$ for all i, and we ensure this by restricting frames to subsets of the prior frame, when taken as sets of lemmas. Conditions (7) and (8) say each frame is *relatively inductive* to the prior frame, except F_∞ which is relatively inductive to itself and thus inductive for the system. To initialize, the algorithm adds the (conjuncts of) *Init* and *Safe* as lemmas to F_0. The algorithm then proceeds by adding lemmas to frames using either *pushing* or *inductive generalization* while respecting this meta-invariant, gradually tightening the bounds on reachability until $F_\infty \Rightarrow Safe$. We can push a lemma $p \in F_i$ to F_{i+1}, provided $F_i \Rightarrow \text{wp}(p)$. When a formula is pushed, the stronger F_{i+1} may permit us to push one or more other formulas, possibly recursively, and so we always push until a fixpoint is reached. Any mutually relatively inductive set of lemmas do not have a finite fixpoint, and we detect these sets (by checking for $F_i = F_{i+1}$) and move them to F_∞.

If the algorithm cannot push a lemma p_a beyond frame i, there is a model of $\neg(F_i \Rightarrow \text{wp}(p_a))$, which is a transition $s \to t$ where $s \in F_i$ and $t \not\models p_a$. We call the pre-state s a *pushing preventer* of p_a. To generate new lemmas, we *block* the pushing preventer s in F_i by first recursively blocking all predecessors of s that are still in F_{i-1}, and then using an inductive generalization (IG) query to *learn* a new lemma that eliminates s. An IG query finds a formula p satisfying:

$$s \not\models p \quad (9) \qquad Init \Rightarrow p \quad (10) \qquad F_{i-1} \wedge p \Rightarrow \text{wp}(p) \quad (11)$$

If we can learn such a lemma, it can be added to F_i and all previous frames, and removes at least the state s stopping p_a from being pushed. Classic PDR/IC3 always chooses to block the pushing preventer of a safety property (lemma from *Safe*) or a predecessor thereof, but other strategies have been considered [17]. The technique used to solve IG queries controls what kind of invariants we are able to discover. In this work we use separation to solve for p, which lets us infer invariants with quantifier alternations.

3 Breadth-First Inductive Generalization with Separation

Inductive generalization is the core of PDR/IC3, and improving it comes in two flavors: making individual queries faster, and generating better lemmas that are more general. We address both of these concerns by restructuring the search to be *breadth-first* rather than *depth-first*. We first discuss naively solving an IG query with separation (as in [19]), then present an algorithm that restructures the search in a breadth-first manner.

3.1 Naive Inductive Generalization with Separation

An IG query is solved in [19] with separation by a simple refinement loop, which performs a series of separation queries with an incrementally growing set of

structure constraints. Starting with a negative constraint s for the state to block, we ask for a separator p and check if eqs. (10) and (11) hold for p using a standard SMT solver. If both hold, p is a solution to the IG query. Otherwise, the SMT solver produces a model which becomes either a positive constraint (corresponding to an initial state p violates) or an implication constraint (a transition edge that shows p is not relatively inductive to F_{i-1}), respectively.

At a high level, the SAT-based algorithm for separation from [19] uses Boolean variables to encode the kind ($\forall/\exists$) and sort of each quantifier, and additional variables for the presence of each syntactically valid literal in each clause in the matrix, which is in CNF. It then translates each structure constraint into a Boolean formula over these variables such that satisfying assignments encode formulas with the correct truth value for each structure. The details of the translation to SAT are not relevant here, except a few key points: (i) separation considers each potential quantifier prefix essentially independently, (ii) complex IG queries can result in hundreds or thousands of constraints, and (iii) prefixes, as partitions of the space of possible separators, vary greatly in how quickly they can be explored. Further, with the black box approach where the prefixes are considered internally by the separation algorithm, even if the separation algorithm uses internal parallelism as suggested in [19], there is still a serialization step when a new constraint is required. As a consequence of (ii) and (iii), a significant failure mode of this naive approach is that the search becomes stuck generating more and more constraints for difficult parts of the search space that ultimately do not contain an easy-to-discover solution to the IG query.

3.2 Prefix Search at the Inductive Generalization Level

To fix the problems with the naive approach, we propose lifting the choice of prefix to the IG level, partitioning a single large separation query into a query for each prefix. Each sub-query can be explored in parallel, and each can proceed independently by querying for new constraints (using eqs. (10) and (11) as before) without serializing by waiting for other prefixes. We call this a *breadth-first* search, because the algorithm can spend approximately equal time on many parts of the search space, instead of a *depth-first* search which exhausts all possibilities in one region before moving on to the next. When regions have greatly varying times to search, the breadth-first approach prevents expensive regions from blocking the search in cheaper regions. This improvement relies on changing the division between separation and inductive generalization: without the knowledge of the formulas (eqs. (10) and (11)) that generate constraints, the separation algorithm cannot generate new constraints on its own.

A complicating factor is that in addition to prefixes varying in difficulty, sometimes there are entire classes of prefixes that are difficult. For example, many IG queries have desirable universal-only solutions, but spend a long time searching for separators with alternations, as there are far more distinct prefixes with alternations than those with only universals. To address this problem, we define possibly overlapping sets of prefixes, called *prefix categories*, and ensure the algorithm spends approximately equal time searching for solutions in

```
def IG(s: state, i: frame):
    ∀P. C(P) = {Negative(s)};
    for i = 1 ... N in parallel:
        while true:
            P = next-prefix();
            while true:
                p = separate C(P);
                if p is UNSEP:
                    break
                elif any c ∈ R_C(P) and p ⊭ c:
                    add c to C(P)
                elif (c := SMT check eqs. (10) and (11)) ≠ UNSAT:
                    add c to C(P)
                else:
                    return p as solution
```

Fig. 1. Pseudocode for our proposed inductive generalization algorithm.

each category (e.g., universally quantified invariants, invariants with at most one alternation and at most one repeated sort). Within each category, we order prefixes to further bias towards likely solutions: first by smallest quantifier depth, then fewest alternations, then those that start with a universal, and finally by smallest number of existentials.

3.3 Algorithm for Inductive Generalization

We present our algorithm for IG using separation in Figure 1. Our algorithm has a fixed number N of worker threads which take prefixes from a queue subject to prefix restrictions, and perform a separation query with that prefix. Each worker thread calls next-prefix() to obtain the next prefix to consider, according to the order discussed in the previous section. To solve a prefix P, a worker performs a refinement loop as in the naive algorithm, building a set of constraints $C(P)$ until a solution to the IG query is discovered or separation reports UNSEP.

While we take steps to make SMT queries for new constraints as fast as possible (Section 5.4), these queries are still expensive and we thus want to re-use constraints between prefixes where it is beneficial. Re-using every constraint discovered so far is not a good strategy as the cost of checking upwards of hundreds of constraints for every candidate separator is not justified by how frequently they actually constrain the search. Instead, we track a set of *related constraints* for a prefix P, $R_C(P)$. We define related constraints in terms of *immediate sub-prefixes* of P, written $S(P)$, which are prefixes obtained by dropping exactly one quantifier from P, i.e. the quantifiers of $P' \in S(P)$ are a subsequence of those in P with one missing. We then define $R_C(P) = \cup_{P' \in S(P)} C(P')$, i.e. the related constraints of P are all those used by immediate sub-prefixes. While $S(P)$ considers only immediate sub-prefixes, constraints may propagate from non-immediate sub-prefixes as the algorithm progresses.

Constraints from sub-prefixes are used because the possible separators for those queries are also possible separators for the larger prefix. Thus the set of constraints from sub-prefixes will definitely eliminate some potential separators, and in the usual case where the sub-prefixes have converged to UNSEP, will rule

out an entire section of the search space. We also opportunistically make use of known constraints for the same prefix generated in prior IG queries, as long as those constraints still satisfy the current frame.

Overall, the algorithm in Figure 1 uses parallelism across prefixes to generate independent separation queries in a breadth-first way, while carefully sharing only useful constraints. From the perspective of the global search for an inductive invariant the algorithm introduces two forms of inductive bias: (i) explicit bias arising from controlling the order and form of prefixes (Section 3.2), and (ii) implicit bias towards formulas which are easy to discover.

4 *k*-Term Pseudo-DNF

We now consider the search space for quantifier-free matrices, and introduce a syntactic form that shrinks the search space while still allowing common invariants with quantifier alternations to be expressed.

Conjunctive and disjunctive normal forms (CNF and DNF) are formulas that consist of a conjunction of *clauses* (CNF) or a disjunction of *cubes* (DNF), where clauses and cubes are disjunctions and conjunctions of literals, respectively: For example, $(a \vee b \vee \neg c) \wedge (b \vee c)$ is in CNF and $(a \wedge \neg c) \vee (\neg a \wedge b)$ is in DNF. We further define k-clause CNF and k-term DNF as formulas with at most k clauses and cubes, respectively.

In [19] separation is performed by finding a matrix in k-clause CNF, biasing the search by minimizing the sum of the number of quantifiers and k. We find that both CNF and DNF are not good fits for the formulas in human-written invariants. For example, consider the following formula from Paxos:

$$\forall r_1, r_2, v_1, v_2, q. \exists n. r_1 < r_2 \wedge \text{proposal}(r_2, v_2) \wedge v_1 \neq v_2 \\ \rightarrow \text{member}(n, q) \wedge \text{left-round}(n, r_1) \wedge \neg\text{vote}(n, r_1, v_1)$$

To write this in CNF, we need to distribute the antecedent over the conjunction, obtaining the 3-clause formula:

$$(r_1 < r_2 \wedge \text{proposal}(r_2, v_2) \wedge v_1 \neq v_2 \rightarrow \text{member}(n, q)) \wedge \\ (r_1 < r_2 \wedge \text{proposal}(r_2, v_2) \wedge v_1 \neq v_2 \rightarrow \text{left-round}(n, r_1)) \wedge \\ (r_1 < r_2 \wedge \text{proposal}(r_2, v_2) \wedge v_1 \neq v_2 \rightarrow \neg\text{vote}(n, r_1, v_1))$$

When written without $\rightarrow$, this matrix has the form $\neg a \vee \neg b \vee c \vee (d \wedge e \wedge \neg f)$, which is already in DNF. Under the k-term DNF, however, the formula requires a single-literal cube for each antecedent literal, i.e. $k = 4$. Because of the quantifier alternation, we cannot split this formula into cubes or clauses, and so a search over either CNF or DNF must consider a significantly larger search space.To solve these issues, we define a variant of DNF, k-term pseudo-DNF (k-pDNF), where one cube is negated, yielding as many individual literals as needed:

Definition 1 (k-term pseudo-DNF). *A quantifier-free formula φ is in k-term pseudo-DNF for $k \geq 1$ if $\varphi \equiv \neg c_1 \vee c_2 \vee \ldots \vee c_k$, where $c_1, \ldots, c_k$ are*

cubes. Equivalently, φ is in k-term pDNF if there exists $n \geq 0$ such that $\varphi \equiv \ell_1 \vee \ldots \vee \ell_n \vee c_2 \vee \ldots \vee c_k$, where $\ell_1, \ldots, \ell_n$ are literals and $c_2, \ldots, c_k$ are cubes.

Note that 1-term pDNF is equivalent to 1-clause CNF, i.e. a single clause. 2-term pDNF correspond to formulas of the form (cube) $\rightarrow$ (cube). Such formulas are sufficient for all but a handful of the lemmas required for invariants in our benchmark suite. An exception is the following, which has one free literal and two cubes (so it is 3-term pDNF):

$$\begin{aligned}&\forall v_1.\ \exists n_1, n_2, n_3, v_2, v_3.\\ &\quad (d(v_1) \rightarrow \neg m(n_1) \wedge u(n_1, v_1)) \vee\\ &\quad (\neg m(n_2) \wedge \neg m(n_3) \wedge u(n_2, v_2) \wedge u(n_3, v_3) \wedge v_2 \neq v_3)\end{aligned}$$

For a fixed k, k-clause CNF, k-term DNF, and k-term pDNF all have the same-size search space, as the SAT query inside the separation algorithm will have one indicator variable for each possible literal in each clause or cube. The advantage of pDNF is that it can express more invariant lemmas with a small k, reducing the size of the search space while still being expressive. We can also see pDNF as a compromise between CNF and DNF, and we find that pDNF is a better fit to the matrices of invariants with quantifier alternation.

5 An Algorithm for Invariant Inference

We now take a step back to consider the high-level PDR/IC3 structure of our algorithm. We have described how our algorithm performs inductive generalization (Sections 3 and 4), which is the central ingredient. We next discuss blocking states that are not backward reachable from a bad state as a heuristic for finding additional useful lemmas. We then discuss how we can search for formulas in the EPR logic fragment and techniques to increase the robustness of SMT solvers. Finally, we give a complete description of our proposed algorithm.

5.1 May-proof-obligations

In classic PDR/IC3, the choice of pushing preventer to block is always that of a safety property. [17] proposed a heuristic that in our terminology is to block the pushing preventer of other existing lemmas, under the heuristic assumption that current lemmas in lower frames are part of the final invariant but lack a supporting lemma to make them inductive. The classic blocked states are known as *must-proof-obligations*, as they are states that must be eliminated somehow to prove the safety property. In contrast, these heuristic states are *may-proof-obligations*, as they may or may not be necessary to block. Our algorithm selects these lemmas at random, biased towards lemmas with smaller matrices.

To block a state, we first recursively block its predecessors in the prior frame, if they exist. For may-proof-obligations,[5] this recursion can potentially reach all

[5] For unsafe transition systems, this can also occur for must-proof-obligations.

the way to an initial state in F_0, and thus proves that the entire chain of states is reachable— i.e., the states cannot be blocked. This fact shows that the original lemma is not part of any final invariant and cannot be pushed past its current frame; it also provides a positive structure constraint useful for future IG queries.

5.2 Multi-block Generalization

After an IG query blocking state s is successful, the resulting lemma p may cause the original lemma that created s to be pushed to the next frame. If not, there will be a new pushing preventer s'. If s' is in the same frame, we can ask whether there is a single IG solution formula p_1 which blocks both s and s'. If we can find such a p_1, it is more likely to generalize past s and s', and we should prefer p_1. This is straightforward to do with separation: we incrementally add another negative constraint to the existing separation queries. To implement *multi-block generalization*, we continue an IG query if the new pushing preventer is suitable (i.e. exists and is in the same frame), accumulating as many negative constraints as we can until we do not have a suitable state or we have spent as much time as the original query. This timeout guarantees we do not spend more than half of our time on generalization, and protects us in the case that the new set of states cannot be blocked together with a simple formula.

5.3 Enforcing EPR

Effectively Propositional Reasoning (EPR, [28]) is a fragment of many-sorted first-order logic in which satisfiability is decidable and satisfiable formulas always have a finite model. The essence of EPR is to limit function symbols, both in the signature and from the Skolemization of existentials, to ensure only a finite number of ground terms can be formed. EPR ensures this property by requiring that there be no cycles in the directed graph with an edge from each domain sort to the codomain sort for every (signature and Skolem) function symbol. For example, $(\forall x{:}S.\, \varphi_1) \vee (\exists y{:}S.\, \varphi_2)$ is in EPR, but $\forall x{:}S.\, \exists y{:}S.\, \varphi_3$ is not in EPR as the Skolem function for y introduces an edge from sort S to itself. The acyclicity requirement means that EPR is not closed under conjunction, and so is best thought of as a property of a whole SMT query rather than of individual lemmas. Despite these restrictions, EPR can be used to verify complex distributed protocols [28].

For invariant inference with PDR/IC3, the most straightforward way to enforce acyclicity is to decide *a priori* which edges are allowed, and to not infer lemmas with disallowed Skolem edges. In practice, enforcing EPR means simply skipping prefixes during IG queries that would create disallowed edges. Without this fixed set of allowed edges, adding a lemma to a frame may prevent a necessary lemma from being added to the frame in a later iteration, as PDR/IC3 lacks a way to remove lemmas from frames. Requiring the set of allowed edges as input is a limitation of our technique and other state-of-the-art approaches (e.g. [14]). We hope that future work expands the scope of decidable logic fragments, so that systems require less effort to model in such a fragment. It is also possible

that our algorithm could be wrapped in an outer search over the possible acyclic sets of edges.

Because separation produces prenex formulas, some EPR formulas would be disallowed without additional effort (e.g. a prenex form of $(\forall x{:}S.\, \varphi_1) \vee (\exists y{:}S.\, \varphi_2)$ is $\forall x{:}S.\, \exists y{:}S.\, (\varphi_1) \vee (\varphi_2)$). In our implementation, we added an option where separation produces prenex formulas that may not be in EPR directly, but where the scope of the quantifiers can be *pushed down* into the pDNF disjunction to obtain an EPR formula. Extra SAT variables are introduced that encode whether a particular quantified variable appears in a given disjunct, and we add the constraint that the quantifiers are nested consistently and in such a way as to be free of disallowed edges. Because this makes separation queries more difficult, we only enable this mode for the single example that requires non-prenex EPR formulas.

5.4 SMT Robustness

Even with EPR restrictions, some SMT queries we generate are difficult for the SMT solvers we use (Z3 [4] and CVC5[6]), sometimes taking minutes, hours, or longer. This wide variation of solving times is significant because separation, and thus IG queries, cannot make progress without a new structure constraint. We adopt several strategies to increase robustness: periodic restarts, running multiple instances of both solvers in parallel, and *incremental queries*. Our incremental queries send the formulas to the SMT solver one at a time, asserting a subset of the input. An UNSAT result from a subset can be returned immediately, and a SAT result can be returned if there is no un-asserted formula violated by the model. Otherwise, one of the violating formulas is asserted, and the process repeats. This process usually avoids asserting all the discovered lemmas from a frame, which significantly speeds up many of the most difficult queries, especially those with dozens of lemmas in a frame or those not in EPR.

5.5 Complete Algorithm

Figure 2 presents the pseudocode for our algorithm, which consists of two parallel tasks (learning and heuristic), each using half of the available parallelism to discharge IG queries, and pushing to fixpoint after adding any lemmas. In this listing, the to-block(ℓ) function computes the state and frame to perform an IG query in order to push ℓ (i.e. the pushing preventer of ℓ or a possibly multi-step predecessor thereof). The heuristic task additionally may find reachable states, and thus mark lemmas as bad. We cancel an IG query when it is solved by a lemma learned or pushed by another task. If the algorithm terminates, then the conjunction of F_∞ is inductive according to the underlying SMT solver.

Our algorithm is parameterized by the logic used for inductive generalization, and thus the form of the invariant. We support universal, EPR, and full first-order logic (FOL) modes. Universal mode restricts the matrices to clauses, and

[6] Successor to CVC4 [1].

```
def P-Fol-Ic3():
    F_0 = init ∪ safety;
    push();
    start Learning(), Heuristic();
    wait for invariant;
def Multiblock(ℓ: lemma, s: state, i):
    S = {s};
    while not timed out:
        p = IG(S, i);
        speculatively add p to frame i;
        s', i' = to-block(ℓ);
        remove p from frame i;
        if i = i':
            add s' to S;
        else:
            break
    add p to frame i;
    push();
def Learning():
    while true:
        s, i = to-block(safety);
        Multiblock(safety, s, i);
def Heuristic():
    while true:
        ℓ = random lemma before
          safety;
        s, i = to-block(ℓ);
        if i = 0:
            mark s reachable;
            mark bad lemmas;
        else:
            Multiblock(ℓ, s, i);
```

Fig. 2. Pseudocode for our proposed inference algorithm, P-Fol-Ic3.

considers predecessors of superstructures when computing to-block() (as in [18]). EPR mode also takes as input the set of allowed edges. In FOL mode, there are no restrictions on the prefix.

6 Evaluation

We evaluate our algorithm and compare with prior approaches on a benchmark of invariant inference problems. We discuss the benchmark, our experimental setup, and the results.

6.1 Invariant Inference Benchmark

Our benchmark is composed of invariant inference problems from prior work on distributed protocols [29,28,7,27,30,2,9], written in or translated to the mypyvy tool's input language [26]. Our benchmark contains a total of 30 problems (Table 1), ranging from simple (toy-consensus, firewall) to complex (stoppable-paxos-epr, bosco-3t-safety). Some problems admit invariants that are purely universal, and others use universal and existential quantifiers, with some in EPR. All our examples are safe transition systems with a known human-written invariant.

6.2 Experimental Setup

We compare our algorithm to the techniques Swiss [14], IC3PO [11,12], fol-ic3 [19], and PDR$^{\forall}$ [18]. We performed our experiments on a 56-thread machine with 64 GiB of RAM, with each experiment restricted to 16 hardware threads, 20GiB of RAM, and a 6 hour time limit.[7] To account for noise caused by randomness in seed selection, we ran each algorithm 5 times and report the number

[7] Specifically, an dual-socket Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz.

Table 1. Experimental results, giving both the median wall-clock time (seconds) of run time and the number of trials successful, out of five. If there were less than 3 successful trials, we report the slowest successful trial, indicated by (>). A dash (-) indicates all trials failed or timed out after 6 hours (21600 seconds). A blank indicates no data.

Example	EPR	**Our**	#	Swiss	#	IC3PO	#	fol-ic3	#	PDR$^{\forall}$	#
lockserv	∀	19	5	9573	4	5	5	7	5	6	5
toy-consensus-forall	∀	4	5	22	5	4	5	11	5	4	5
ring-id	∀	7	5	192	5	81	5	28	5	20	5
sharded-kv	∀	8	5	17291	5	4	5	19	5	6	5
ticket	∀	23	5	-	0	-	0	240	5	22	5
learning-switch	∀	76	5	1744	4	29	5	-	0	94	5
consensus-wo-decide	∀	50	5	52	5	6	5	33	5	29	5
consensus-forall	∀	1908	5	80	5	15	5	1125	5	104	5
cache	∀	2492	4	-	0	3906	5	-	0	2628	5
paxos-forall	∀	885	5	-	0	-	0	-	0	555	5
flexible-paxos-forall	∀	1961	5	-	0	1654	5	-	0	423	5
stoppable-paxos-forall	∀	7779	5	-	0	-	0	-	0	-	0
fast-paxos-forall	∀	-	0	-	0	-	0	-	0	20176	3
vertical-paxos-forall	∀	-	0	-	0	-	0	-	0	-	0
firewall	–	4	5	-	0	3	5	9	5		
sharded-kv-no-lost-keys	✓	4	5	9	5	4	5	5	5		
toy-consensus-epr	✓	4	5	10	5	4	5	49	5		
ring-id-not-dead	–	19	5	-	0	-	0	221	3		
consensus-epr	✓	37	5	57	5	28	5	-	0		
client-server-ae	✓	4	5	11	5	4	5	442	5		
client-server-db-ae	–	16	5	46	5	37	5	6639	4		
hybrid-reliable-broadcast	–	178	5	-	0	-	0	937	5		
paxos-epr	✓	920	5	14332	4	-	0	-	0		
flexible-paxos-epr	✓	418	5	4928	5	-	0	-	0		
multi-paxos-epr	✓	4272	4	-	0	-	0	-	0		
stoppable-paxos-epr	✓	>18297	2	-	0	-	0	-	0		
fast-paxos-epr	✓	9630	3	-	0	-	0	-	0		
vertical-paxos-epr	✓	-	0	-	0	-	0	-	0		
block-cache-async	–	-	0	-	0	-	0	-	0		
bosco-3t-safety	✓	>11019[1]	1	-	0	-	0	-	0		

[1]With EPR push down enabled.

of successes and the median time. PDR$^{\forall}$, IC3PO, and fol-ic3 are not designed to use parallelism, while Swiss and our technique make use of parallelism. For IC3PO, we use the better result from the two implementations [11] and [12], and give reported results for those we could not replicate. For our technique, we ran the tool in universal-only, EPR, or full FOL mode as appropriate. For k-pDNF, we use $k = 1$ for universal prefixes and $k = 3$ otherwise.

6.3 Results and Discussion

We present the results of our experiments in Table 1. In general, for examples that converge with both prior approaches and our technique, we match or exceed existing results, with significant performance gains for some problems such as client-server-db-ae relative to the previous separation-based approach. Along with other techniques, we solve paxos-epr and flexible-paxos-epr, which are the simplest variants of Paxos in our benchmark, but nonetheless represents a significant jump in complexity over the examples solved by the prior generation of PDR/IC3 techniques. Paxos and its variants are notable for having invariants

Table 2. Ablation study. Columns are interpreted as in Table 1.

Example	**Our**	#	No pDNF	#	No EPR	#	No Inc. SMT	#	No Gen.	#
lockserv	19	5					34	5	13	5
toy-consensus-forall	4	5					5	5	4	5
ring-id	7	5					11	5	13	5
sharded-kv	8	5					11	5	7	5
ticket	23	5					42	5	21	5
learning-switch	76	5					338	5	288	5
consensus-wo-decide	50	5					50	5	51	5
consensus-forall	1908	5					2154	5	558	5
cache	2492	4					>16826	2	13116	5
paxos-forall	885	5					1071	5	10488	4
flexible-paxos-forall	1961	5					1014	5	>4168	2
stoppable-paxos-forall	7779	5					2820	5	>18727	1
fast-paxos-forall	-	0					>16573	1	-	0
vertical-paxos-forall	-	0					-	0	-	0
firewall	4	5	4	5			4	5	4	5
sharded-kv-no-lost-keys	4	5	4	5	4	5	5	5	4	5
toy-consensus-epr	4	5	5	5	5	5	5	5	5	5
ring-id-not-dead	19	5	37	5			44	5	52	5
consensus-epr	37	5	126	5	724	5	45	5	233	5
client-server-ae	4	5	3	5	4	5	4	5	4	5
client-server-db-ae	16	5	13	5			20	5	10	5
hybrid-reliable-broadcast	178	5	98	5			173	5	629	5
paxos-epr	920	5	10135	4	>2895	1	609	5	3201	5
flexible-paxos-epr	418	5	13742	3	-	0	775	5	799	5
multi-paxos-epr	4272	4	>15176	1	-	0	15854	3	7326	4
stoppable-paxos-epr	>18297	2	-	0	-	0	>20659	1	>11946	1
fast-paxos-epr	9630	3	-	0	-	0	8976	3	>20871	2
vertical-paxos-epr	-	0	-	0	-	0	-	0	-	0
block-cache-async	-	0	-	0			-	0	>20038	2
bosco-3t-safety	>11019	1	-	0	-	0	>8581	1	>16689	1

with two quantifier alternations (∀∃∀) and a maximum quantifier depth of 6 or 7. We uniquely solve multi-, fast-, and stoppable-paxos-epr, which add significant complexity in the number of sorts, symbols, and quantifier depth required. Due to variations in seeds and the non-determinism of parallelism, our technique was only successful in some trials, but these results nevertheless demonstrate that our technique is capable of solving these examples. Our algorithm is unable to solve vertical-paxos-epr, as this example requires a 7 quantifier formula that is very expensive for our IG solver.

For universal-only examples, our algorithm is able to solve all but one of the examples[8] solved by other techniques, and is able to solve one that others cannot. In some cases (e.g. consensus-forall), our solution is slower than other approaches, but on the whole our algorithm is competitive in a domain it is not specialized for. In addition, we significantly outperform the existing separation-based algorithm (fol-ic3) by solving several difficult examples (cache, paxos-forall).

6.4 Ablation Study

Table 2 presents an ablation study investigating effect of various features of our technique. The first column of Table 2 repeats the full algorithm results, and the remaining columns report the performance with various features disabled

[8] fast-paxos-forall, which is solved by our technique in the ablation study, albeit rarely.

Table 3. Parallel vs sequential comparison. Each of 5 trials ran with 3 or 48 hour timeouts, respectively. The number of successes, and the average number of IG queries in each trial (including failed ones) are given.

	Successes		IG Queries	
Example	Par.	Seq.	Par.	Seq.
paxos-epr	5	5	61	76
flexible-paxos-epr	5	5	64	72
multi-paxos-epr	3	1	67	84

individually. The most important individual contributions come from k-pDNF matrices and EPR. Using a 5-clause CNF instead of pDNF matrix (No pDNF) causes many difficult examples to fail and some (e.g., flexible-paxos-epr) to take significantly longer even when they do succeed.[9] Similarly, using full FOL mode instead of EPR (No EPR) leads to timeouts for all but the simplest Paxos variants. Incremental SMT queries (No Inc. SMT) make the more difficult Paxos variants, and the universal cache example, succeed much more reliably. Multi-block generalization (No Gen.) makes many problems faster or more reliable, but disabling it allows block-cache-async to succeed.

To isolate the benefits of parallelism, we ran several examples in both parallel and serial mode with a proportionally larger timeout (Table 3). In both modes we use a single prefix category containing all prefixes, with the same static order over prefixes.[10] Beyond the wall-clock speedup, the parallel IG algorithm affects the quality of the learned lemmas, that is, how well they generalize and avoid overfitting. To estimate the quality of generalization, we count the total number of IG queries performed by each trial and report the average over the five trials. In all examples, the parallel algorithm learns fewer lemmas overall, which suggests it generalizes better. We attribute this improved generalization to the implicit bias towards lemmas that are faster to discover. For the more complicated example (multi-paxos-epr), this difference has an impact on the success rate.

7 Related Work

Extensions of PDR/IC3. The PDR/IC3 [3,5] algorithm has been very influential as an invariant inference technique, first for hardware (finite state) systems and later for software (infinite state). There are multiple extensions of PDR/IC3 to infinite state systems using SMT theories [16,20]. [18] extended PDR/IC3 to universally quantified first-order formulas using the model-theoretic notion of *diagrams*. [13] applies PDR/IC3 to find universally quantified invariants over arrays and also to manage quantifier instantiation. Another extension of PDR/IC3 for universally quantified invariants is [23], where a quantified invariant is generalized from an invariant of a bounded, finite system. This technique of generalization from a bounded system has also been extended to quantifiers with al-

[9] With a single clause, there is no difference between CNF and k-pDNF so results are only given for existential problems.

[10] To make the comparison cleaner, we also disabled multi-block generalization.

ternations [11]. Recently, [31] suggested combining synthesis and PDR/IC3, but they focus on word-level hardware model checking and do not support quantifier alternations. Most of these works focus on quantifier-free or universally quantified invariants. In contrast, we address unique challenges that arise when supporting lemmas with quantifier alternations.

The original PDR/IC3 algorithm has also been extended with techniques that use different heuristic strategies to find more invariants by considering additional proof goals and collecting reachable states [15,17]. Our implementation benefits from some of these heuristics, but our contribution is largely orthogonal as our focus is on inductive generalization of quantified formulas. Generating lemmas from multiple states, similar to multi-block generalization, was explored in [21].

[24] suggests a way to parallelize PDR/IC3 by combining a portfolio approach with problem partitioning and lemma sharing. Our parallelism is more tightly coupled into PDR/IC3, as we parallelize the inductive generalization procedure.

Quantified Separation. Quantified separation [19] was recently introduced as a way to find quantified invariants with quantifier alternations. While [19] introduced a way to combine separation and PDR/IC3, it has limited scalability and cannot find the invariants of complex protocols such as Paxos. Our work here is motivated by these scalability issues. In contrast to [19], our technique is able to find complex invariants by avoiding expensive but useless areas of the search space using a breadth-first strategy and a multi-dimensional inductive bias. While [19] searches for quantified lemmas in CNF, we introduce and use k-term pDNF. k-term pDNF can express the necessary lemmas of many distributed protocols more succinctly, resulting in better scalability.

Synthesis-Based Approaches to Invariant Inference. Synthesis is a common approach for automating invariant inference. ICE [10] is a framework for learning inductive invariants from positive, negative, and implication constraints. Our use of separation is similar, but it is integrated into PDR/IC3's inductive generalization, so unlike ICE we find invariants incrementally.

Enumeration-Based Approaches. Another approach is to use enumerative search, for example [6], which only supports universal quantification. Enumerative search has been extended to quantifier alternations in [14], which is able to infer the invariants of complex protocols such as some Paxos variants.

8 Conclusion

We have presented an algorithm for quantified invariant inference that combines separation and inductive generalization. Our algorithm uses a breadth-first strategy to avoid regions of the search space that are expensive. We also explore a new syntactic form that is well-suited for lemmas with alternations. We show via a large scale experiment that our algorithm advances the state of the art in quantified invariant inference with alternations, and finds significantly more invariants on difficult problems than prior approaches.

References

1. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanovi'c, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proceedings of the 23rd International Conference on Computer Aided Verification (CAV '11). Lecture Notes in Computer Science, vol. 6806, pp. 171–177. Springer (Jul 2011), https://dl.acm.org/doi/10.5555/2032305.2032319, Snowbird, Utah
2. Berkovits, I., Lazic, M., Losa, G., Padon, O., Shoham, S.: Verification of threshold-based distributed algorithms by decomposition to decidable logics. In: Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II. pp. 245–266 (2019). https://doi.org/10.1007/978-3-030-25543-5_15
3. Bradley, A.R.: SAT-based model checking without unrolling. In: Jhala, R., Schmidt, D. (eds.) Verification, Model Checking, and Abstract Interpretation. pp. 70–87. Springer Berlin Heidelberg, Berlin, Heidelberg (2011), https://link.springer.com/chapter/10.1007/978-3-642-18275-4_7
4. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. TACAS'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg (2008), https://dl.acm.org/citation.cfm?id=1792734.1792766
5. Eén, N., Mishchenko, A., Brayton, R.K.: Efficient implementation of property directed reachability. In: International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011. pp. 125–134 (2011), https://dl.acm.org/citation.cfm?id=2157675
6. Fedyukovich, G., Prabhu, S., Madhukar, K., Gupta, A.: Quantified invariants via syntax-guided synthesis. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11561, pp. 259–277. Springer (2019). https://doi.org/10.1007/978-3-030-25540-4_14
7. Feldman, Y.M., Padon, O., Immerman, N., Sagiv, M., Shoham, S.: Bounded quantifier instantiation for checking inductive invariants. In: Proceedings, Part I, of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems - Volume 10205. pp. 76–95. Springer-Verlag, Berlin, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_5
8. Feldman, Y.M.Y., Padon, O., Immerman, N., Sagiv, M., Shoham, S.: Bounded quantifier instantiation for checking inductive invariants. Log. Methods Comput. Sci. **15**(3) (2019). https://doi.org/10.23638/LMCS-15(3:18)2019
9. Feldman, Y.M.Y., Wilcox, J.R., Shoham, S., Sagiv, M.: Inferring inductive invariants from phase structures. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification. pp. 405–425. Springer International Publishing, Cham (2019), https://link.springer.com/chapter/10.1007/978-3-030-25543-5_23
10. Garg, P., Löding, C., Madhusudan, P., Neider, D.: ICE: A Robust Framework for Learning Invariants. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification. pp. 69–87. Springer International Publishing, Cham (2014), https://link.springer.com/chapter/10.1007/978-3-319-08867-9_5
11. Goel, A., Sakallah, K.: On symmetry and quantification: A new approach to verify distributed protocols. In: Dutle, A., Moscato, M.M., Titolo, L., Muñoz, C.A., Perez, I. (eds.) NASA Formal Methods. pp. 131–150. Springer International Publishing, Cham (2021), https://link.springer.com/chapter/10.1007/978-3-030-76384-8_9

12. Goel, A., Sakallah, K.A.: Towards an automatic proof of Lamport's Paxos. In: 2021 Formal Methods in Computer Aided Design (FMCAD). pp. 112–122 (2021). https://doi.org/10.34727/2021/isbn.978-3-85448-046-4_20
13. Gurfinkel, A., Shoham, S., Vizel, Y.: Quantifiers on demand. In: Lahiri, S.K., Wang, C. (eds.) Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11138, pp. 248–266. Springer (2018). https://doi.org/10.1007/978-3-030-01090-4_15
14. Hance, T., Heule, M., Martins, R., Parno, B.: Finding invariants of distributed systems: It's a small (enough) world after all. In: Mickens, J., Teixeira, R. (eds.) 18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021. pp. 115–131. USENIX Association (2021), https://www.usenix.org/conference/nsdi21/presentation/hance
15. Hassan, Z., Bradley, A.R., Somenzi, F.: Better generalization in IC3. In: 2013 Formal Methods in Computer-Aided Design. pp. 157–164 (2013). https://doi.org/10.1109/FMCAD.2013.6679405
16. Hoder, K., Bjørner, N.: Generalized property directed reachability. In: Cimatti, A., Sebastiani, R. (eds.) Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7317, pp. 157–171. Springer (2012). https://doi.org/10.1007/978-3-642-31612-8_13
17. Ivrii, A., Gurfinkel, A.: Pushing to the top. In: 2015 Formal Methods in Computer-Aided Design (FMCAD). pp. 65–72 (2015). https://doi.org/10.1109/FMCAD.2015.7542254
18. Karbyshev, A., Bjørner, N., Itzhaky, S., Rinetzky, N., Shoham, S.: Property-directed inference of universal invariants or proving their absence. J. ACM **64**(1), 7:1–7:33 (Mar 2017). https://doi.org/10.1145/3022187
19. Koenig, J.R., Padon, O., Immerman, N., Aiken, A.: First-order quantified separators. In: Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 703–717. PLDI 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3385412.3386018
20. Komuravelli, A., Gurfinkel, A., Chaki, S.: Smt-based model checking for recursive programs. Formal Methods Syst. Des. **48**(3), 175–205 (2016). https://doi.org/10.1007/s10703-016-0249-4
21. Krishnan, H.G.V., Chen, Y., Shoham, S., Gurfinkel, A.: Global guidance for local generalization in model checking. In: Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II. pp. 101–125 (2020). https://doi.org/10.1007/978-3-030-53291-8_7
22. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. **16**(2), 133–169 (may 1998). https://doi.org/10.1145/279227.279229
23. Ma, H., Goel, A., Jeannin, J., Kapritsos, M., Kasikci, B., Sakallah, K.A.: I4: incremental inference of inductive invariants for verification of distributed protocols. In: Brecht, T., Williamson, C. (eds.) Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019. pp. 370–384. ACM (2019). https://doi.org/10.1145/3341301.3359651
24. Marescotti, M., Gurfinkel, A., Hyvärinen, A.E.J., Sharygina, N.: Designing parallel PDR. In: Stewart, D., Weissenbacher, G. (eds.) 2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017. pp. 156–163. IEEE (2017). https://doi.org/10.23919/FMCAD.2017.8102254

25. McMillan, K.L.: Lazy annotation revisited. In: Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings. pp. 243–259 (2014). https://doi.org/10.1007/978-3-319-08867-9_16
26. mypyvy repository. https://github.com/wilcoxjay/mypyvy
27. Padon, O., Hoenicke, J., Losa, G., Podelski, A., Sagiv, M., Shoham, S.: Reducing liveness to safety in first-order logic. Proc. ACM Program. Lang. **2**(POPL) (Dec 2017). https://doi.org/10.1145/3158114
28. Padon, O., Losa, G., Sagiv, M., Shoham, S.: Paxos made EPR: decidable reasoning about distributed protocols. Proceedings of the ACM on Programming Languages **1**(OOPSLA), 1–31 (Oct 2017). https://doi.org/10.1145/3140568
29. Padon, O., McMillan, K.L., Panda, A., Sagiv, M., Shoham, S.: Ivy: Safety verification by interactive generalization. In: Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 614–630. PLDI '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2908080.2908118
30. Taube, M., Losa, G., McMillan, K.L., Padon, O., Sagiv, M., Shoham, S., Wilcox, J.R., Woos, D.: Modularity for decidability of deductive verification with applications to distributed systems. In: Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 662–677. PLDI 2018, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3192366.3192414
31. Zhang, H., Gupta, A., Malik, S.: Syntax-guided synthesis for lemma generation in hardware model checking. In: Henglein, F., Shoham, S., Vizel, Y. (eds.) Verification, Model Checking, and Abstract Interpretation - 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17-19, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12597, pp. 325–349. Springer (2021). https://doi.org/10.1007/978-3-030-67067-2_15