Property Directed Polyhedral Abstraction*

Nikolaj Bjørner and Arie Gurfinkel

Abstract. This paper combines the benefits of Polyhedral Abstract Interpretation (poly-AI) with the flexibility of Property Directed Reachability (PDR) algorithms for computing safe inductive convex polyhedral invariants. We develop two algorithms that integrate Poly-AI with PDR and show their benefits on a prototype in Z3 using a preliminary evaluation. The algorithms mimic traditional forward Kleene and a chaotic backward iterations, respectively. Our main contribution is showing how to replace expensive convex hull and quantifier elimination computations, a major bottleneck in poly-AI, with demand-driven property-directed algorithms based on interpolation and model-based projection. Our approach integrates seamlessly within the framework of PDR adapted to Linear Real Arithmetic, and allows to dynamically decide between computing convex and non-convex invariants as directed by the property.

1 Introduction

Linear Real Arithmetic (LRA) enjoys a prominent rôle in symbolic model checking. Semantics of many program statements and properties can be expressed using LRA. In practice, it is often sufficient to limit the verification of such programs to a search for linear arithmetic invariants [20, 19, 15, 9, 22, 26, 10, 24, 7]. These methods, however, cover only a tiny fraction of the search space of LRA invariants, and even worse, miss simple invariants.

$x \leftarrow y \leftarrow z \leftarrow 0$
ℓ_0: **while** $*$ **do**
| $x \leftarrow x+1$; $y \leftarrow y+1$; $z \leftarrow z-2$
end
ℓ_1: **while** $*$ **do**
| $x \leftarrow x-1$; $y \leftarrow y-3$; $z \leftarrow z+2$
end
ℓ_2: **assert** $x \leq 0 \rightarrow z \geq 0 \wedge y \leq 0$

Fig. 1. Program Bouncy.

Consider for example the program Bouncy in Fig. 1. It increments and decrements variables x, y, z in tandem. There is a simple proof of the assertion by using convex polyhedra invariant: $\ell_0 \rightarrow 2x = 2y = -z$, $\ell_1 \rightarrow 2x = -z \wedge y \leq x$. On the other hand, an abstraction-refinement proof that starts from either end (the initial state or the assertion) gets stuck in this example deriving specialized assertions about exact values of each variable. Convex polyhedral invariants, however, are often insufficient. For example, they

* This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution. DM-0001643.

cannot express disequalities (e.g., $x \neq y$), and many disjunctive extensions (e.g.,[25, 4, 17, 1]) have been proposed to remedy this.

This paper embarks on the quest of devising practical property directed [8] polyhedral abstraction algorithms [13]. Our grander ambition is to enable practical model checking methods that search effectively the relevant space of all linear invariants. However, the goal of this paper is more modest: import the search for convex linear invariants, as done by abstract interpretation, into a property directed framework, and do so efficiently. The resulting approach should retain the advantages of restricting the search in an abstract space as well as limiting derived invariants to only the ones that are sufficient for establishing a given property. Indeed, we claim that the combination of polyhedral abstraction and property directed model checking allows to simultaneously address limitations of each approach when they are used in isolation.

The first step towards this goal is a modular account of PDR (Section 3), and the first complete description in Section 4 of $\mathcal{A}$PDR: PDR for LRA. We then develop two main ingredients, a *forward* procedure, $\mathcal{F}$PDR in Section 5, that produces convex polyhedra invariants; and a *backward*, $\mathcal{B}$PDR in Section 6, for complements of convex polyhedra. $\mathcal{F}$PDR mimics forward Kleene iteration, and $\mathcal{B}$PDR mimics backward chaotic iteration, respectively. We present the ingredients in isolation and show that they can be combined in Section 7 in a framework we call PolyPDR. A crucial enabler for PolyPDR is the *syntactic convex closure* method from [5] (Section 2). It allows us to avoid maintaining polyhedra explicitly, in contrast to main tools [3] for polyhedral abstraction that rely on computationally expensive steps that amount to quantifier elimination. To use syntactic convex closure effectively, we integrate a novel algorithm, CCSAT, that finds polyhedra invariants incrementally as *half-space interpolants* [2]. The resulting method inherits several features from polyhedral abstract interpretation and allows to refine the abstraction lazily based on a proof search. The $\mathcal{B}$PDR method dually computes co-convex polyhedra invariants. Section 8 reports on a preliminary evaluation on selected examples that are known to be difficult to $\mathcal{A}$PDR, yet are easy for $\mathcal{F}$PDR or $\mathcal{B}$PDR.

Verification with Interpolation-based MC versus Polyhedral AI. We believe that this work also sheds light on the relationship between abstract interpretation-based and interpolation-based approaches for discovering convex arithmetic invariants. Recall that a *Craig Interpolant* of two inconsistent formulas A and B is a formula I such that $A \to I$, $I \to \neg B$, and the free variables of I are common to A and B. Interpolation-based model checkers use interpolants as oracles to extract constraints relevant for verifying a given property. Table 1 summarizes interpolation procedures for LRA. In the table, *Bool*, *Mono*, *DNF*, and *HalfSpace* stand for Boolean combinations of linear inequalities, monomials (i.e., conjunctions of literals), disjunction of monomials, and a single linear inequality, respectively. Note that the procedures are partial — they are only defined when an interpolant of the particular kind exists. For example, a half-space interpolant might not exists even when A and B are inconsistent.

Name	Domain	Algorithm
SmtItp	$Bool \times Bool \to Bool$	MathSat5 [11]
Itp	$Mono \times Bool \to Mono$	GPDR [19]
HalfItp	$DNF \times DNF \to HalfSpace$	[2]
HalfItp	$Bool \times Mono \to HalfSpace$	CCSAT (Sec. 5.2)
PolyItp	$Mono \times Bool \to Mono$	—

Table 1. Interpolation algorithms for Linear Real Arithmetic (LRA).

The general interpolation procedure SmtItp does not guarantee that the interpolant is convex (or a monomial), even if the inputs are. This makes it difficult to compare it to AI. For other procedures, the key difference is that in AI all operations are typically restricted to the faces of the input polyhedra, whereas interpolation operates over linear combinations (so called Farkas consequences) of the input constraints. We show in Section 5.3 that this leads to a significant difference in the two approaches. To unify MC and polyhedral AI, we suggest it is necessary to restrict interpolants to a subset of faces of A that suffice to separate B. Such an interpolant, we call it PolyItp, can be implemented using Fourier-Motzkin-based decision procedures for LRA (e.g., [14, 23]), but we are not aware of any interpolation or verification procedures based on it.

2 Preliminaries: Closures and Polyhedral Abstraction

In this section we recall some main notions from Polyhedral Abstraction. The construction for *syntactic convex closures* [5] is central to our quest: it lets us write down the convex closure of two convex polyhedra as the solutions to a linear arithmetic formula. We also recall basic notions from polyhedral abstraction to set the stage for our property directed approach.

2.1 Convex Hulls and Syntactic Convex Closures

Let X be a subset of $\mathbb{Q}^n$. We write $\overline{X}$ for the topological closure of X. X is called *closed* if it is invariant under topological closure, i.e., $\overline{X} = X$. We write $CH(X)$ for the *convex hull* of X defined as the set of all affine combinations of points in X:

$$CH(X) \equiv \{\lambda \boldsymbol{x} + (1 - \lambda \boldsymbol{y}) \mid \boldsymbol{x}, \boldsymbol{y} \in X, 0 \leq \lambda \leq 1\}.$$

X is called *convex* if it is invariant under the convex hull. A convex hull of a closed set is not necessarily closed. In particular, a convex hull of a closed set and a point is not closed. For example,

$$CH(x = 0 \wedge y = 1 \ \vee \ x \geq 0 \wedge x = y) \ \equiv \ 0 \leq x \leq y < x + 1\,.$$

We write $CC(X) \equiv \overline{CH(X)}$ for the convex closure of X. Of course:

$$CC(x = 0 \wedge y = 1 \ \vee \ x \geq 0 \wedge x = y) \ \equiv \ 0 \leq x \leq y \leq x + 1\,.$$

A *closed polyhedron* $P(\boldsymbol{x}) \subseteq \mathbb{Q}^n$ is a set of solutions to a conjunction of linear non-strict inequalities, of the form $A\boldsymbol{x} \leq a$. P is closed and convex. In the rest of the paper, unless noted otherwise, we do not distinguish between the syntactic and semantic representation of P. We also restrict our attention to closed polyhedra, i.e., systems with non-strict inequalities only. While this is a significant limitation, in practice, we use systems over $\mathbb{Q}$ to approximate systems over $\mathbb{N}$. Hence, the restriction can be enforced before the relaxation.

A very useful property of convex closure is that it can be computed by Linear Programming, using, what we call, a *syntactic convex closure*.

Definition 1 (Syntactic Convex Closure). *[5] Let $\{P_i(\boldsymbol{x}) = A_i\boldsymbol{x} \leq \boldsymbol{a}_i\}$ be a set of polyhedra. The* syntactic convex closure *$cc(\{P_i\})$ is defined as follows:*

$$cc(\{P_i\}) \equiv \left(\boldsymbol{x} = \sum_i \boldsymbol{z}_i\right) \wedge \left(1 = \sum_i \sigma_i\right) \quad \wedge \quad \bigwedge_i (A_i\boldsymbol{z}_i \leq \sigma_i\boldsymbol{a}_i \wedge \sigma_i \geq 0)$$

where $\{\boldsymbol{z}_i\}$ and $\{\sigma_i\}$ are fresh variables different from $\boldsymbol{x}$.

Convex closure can be computed by existentially quantifying all variables introduced by the syntactic convex closure transformation.

Theorem 1. *[5] Let $\{P_i(\boldsymbol{x}) = A_i\boldsymbol{x} \leq \boldsymbol{a}_i\}$ be a set of polyhedra. Then,*

$$CC(\{P_i\}) \equiv \exists V \cdot cc(\{P_i\})$$

where $V = \{\boldsymbol{z}_i\} \cup \{\sigma_i\}$.

This syntactic form is the basis of our approach.

2.2 Polyhedral Abstract Interpretation

We give a brief overview of polyhedral abstract domain that is necessary to understand our results. The reader is referred to [12, 13] for more details. The polyhedral abstract domain over $\mathbb{Q}^n$ is a tuple $\langle \mathcal{P}, \alpha, \gamma, \top, \bot, \sqcap, \sqcup, \nabla \rangle$, where $\mathcal{P}$ is the set of all polyhedra over $\mathbb{Q}^n$, and for $X \subseteq \mathbb{Q}^n$ and $P_1, P_2 \in \mathcal{P}$,

$$\alpha(X) = CC(X) \quad \gamma(P_1) = P_1 \quad P_1 \sqcup P_2 = CC(\{P_1, P_2\}) \quad P_1 \sqcap P_2 = P_1 \cap P_2$$

and ∇ is a operator satisfying extrapolation ($P_1 \sqcup P_2 \subseteq P_1 \nabla P_2$), and convergence: for any increasing sub-sequence of $\mathbb{Q}^n$, $X_0 \subseteq X_1 \subseteq \cdots$, the sequence Y_i, defined as follows,

$$Y_0 = X_0 \qquad\qquad Y_n = Y_{n-1} \nabla (Y_{n-1} \sqcup X_n)$$

is ultimately convergent, (i.e., there is an $N \in \mathbb{N}$ s.t. $Y_N = Y_{N+1}$). The standard polyhedra widening [13] ∇_s is defined as follows:

$$P_1 \nabla_s P_2 = \{H \text{ is a half-space of } P_1 \;\mid P_2 \rightarrow H\}$$

and is often extended to also keep the constraints of P_2 that are mutually redundant with those in P_1 [18]. Note that for simplicity, we assume that an abstract domain is a subset of a concrete one, making γ an identity.

Given post- and pre-transformers we can define abstract versions using convex closures as follows:

$$post_\alpha(X) = CC(post(X)) \qquad\qquad pre_\alpha(X) = CC(pre(X))$$

Forward abstract interpretation computes an over-approximation of the transitive closure of *post* by iterating the Kleene iteration sequence $\{Y_i\}$ until convergence, where

$$Y_0 = \alpha(X) \qquad Y_n = \begin{cases} Y_{n-1} \sqcup post_\alpha(Y_{n-1}) & \text{if } n \notin W \\ Y_{n-1} \nabla (Y_{n-1} \sqcup post_\alpha(Y_{n-1})) & \text{if } n \in W \end{cases} \tag{1}$$

and W is an infinite subset of $\mathbb{N}$ that determines the widening strategy. Note that each Y_i over-approximates the set of states reachable in i steps or less. Alternatively, abstract interpretation can be done using chaotic iteration strategy by computing the sequence $\{Z_i\}$:

$$Z_0 = \alpha(X) \quad s_n \in post(\gamma(Z_{n-1})) \quad Z_n = \begin{cases} Z_{n-1} \sqcup \alpha(s_{n-1}) & \text{if } n \notin W \\ Z_{n-1} \nabla (Z_{n-1} \sqcup \alpha(s_{n-1})) & \text{if } n \in W \end{cases} \tag{2}$$

Intuitively, the sequence $\{Z_i\}$ over-approximates the sequence $\{s_i\}$ of states reachable by iterative application of best abstract transformer $post_\alpha$ and concretization γ. Backward abstract interpretation is defined similarly to over-approximate transitive closure of *pre*.

3 Property Directed Reachability

This section introduces a modular, rule-based, description of property directed reachability. It simplifies the presentation of our refinements to PDR throughout the paper.

3.1 Symbolic Reachability

A symbolic reachability problem is given by a tuple:

$$\langle \boldsymbol{v}, Init, \rho, Bad \rangle \tag{3}$$

where $\boldsymbol{v}$ is a set of state variables. *Init* and *Bad* are formulae with free variables in $\boldsymbol{v}$ representing the initial and bad states, respectively, and $\rho(\boldsymbol{v}, \boldsymbol{v}')$ is a transition relation. The problem is to decide whether there is a state in *Init* that can reach a state in Bad. Formally, a bad state is reachable, if there is an N, such that the following formula is satisfiable:

$$Init(\boldsymbol{v}_0) \ \wedge \ \bigwedge_{i=0}^{N-1} \rho(\boldsymbol{v}_i, \boldsymbol{v}_{i+1}) \ \wedge \ Bad(\boldsymbol{v}_N) \tag{4}$$

The bad states are unreachable if there exists a formula I over $\boldsymbol{v}$, called an inductive invariant, such that

$$\big((I \wedge \rho) \ \vee \ Init'\big) \rightarrow I' \qquad\qquad I \rightarrow \neg Bad \tag{5}$$

We have used $Init'$ and I' for formulas where the variables $\boldsymbol{v}$ are replaced by primed versions $\boldsymbol{v}'$.

Example 1. The transition system for program Bouncy (Fig. 1) is given by $\boldsymbol{v} = x, y, z, \pi$, where π is a program counter, and $Init$, Bad, and ρ are defined as follows:

$$Init \equiv x = y = z = \pi = 0 \quad Bad \equiv x \leq 0 \wedge (z < 0 \vee y \geq 0) \tag{6}$$

$$\begin{aligned} \rho \equiv\ & (\pi = 0 \wedge \pi' = 0 \wedge x' = x + 1 \wedge y' = y + 1 \wedge z' = z - 2) \vee \\ & (\pi = 0 \wedge \pi' = 1 \wedge x' = x \wedge y' = y \wedge z' = z) \vee \\ & (\pi = 1 \wedge \pi' = 1 \wedge x' = x - 1 \wedge y' = y - 3 \wedge z' = z + 2) \vee \\ & (\pi = 1 \wedge \pi' = 2 \wedge x' = x \wedge y' = y \wedge z' = z) \end{aligned} \tag{7}$$

Bad is unreachable, and a certificate is

$$(\pi = 0 \rightarrow 2x = 2y = -z) \wedge ((\pi = 1 \vee \pi = 2) \rightarrow 2x = -z \wedge y \leq x) \tag{8}$$

3.2 A Rule Based Algorithm Description

The finite state model checking algorithm IC3 was introduced in [8]. It maintains sets of clauses $R_0, \ldots, R_i, \ldots, R_N$, called a *trace*, that are properties of states reachable in i steps from the initial states $Init$. Elements of R_i are called *lemmas*. In the following, we assume that R_0 is initialized to $Init$. After establishing that $Init \rightarrow \neg Bad$, the algorithm maintains the following invariants (for $0 \leq i < N$):

Invariant 1

$$R_i \rightarrow \neg Bad \qquad\qquad R_i \rightarrow R_{i+1} \qquad\qquad R_i \wedge \rho \rightarrow R'_{i+1}$$

That is, each R_i is safe, the trace is monotone, and R_{i+1} is inductive relative to R_i. In practice, the algorithm enforces monotonicity by maintaining $R_{i+1} \subseteq R_i$.

We introduce the following shorthand for convenience

$$\mathcal{F}(R) \equiv (R \ \wedge \ \rho) \ \vee \ Init' \tag{9}$$

Alg. 1 summarizes, in a simplified form, a variant of the IC3 algorithm. The algorithm maintains a queue of counter-examples Q. Each element of Q is a

Data: Q a queue of counter-examples. Initially, $Q = \emptyset$.
Data: N a level indication. Initially, $N = 0$.
Data: $R_0, R_1, \ldots, R_N$ is a trace. Initially, $R_0 = \mathit{Init}$.
repeat

Unreachable If there is an $i < N$ s.t. $R_{i+1} \to R_i$, return *Unreachable*.

Reachable If there is an m s.t. $\langle m, 0\rangle \in Q$ return *Reachable*.

Unfold If $R_N \to \neg \mathit{Bad}$, then set $N \leftarrow N + 1$, $R_N \leftarrow \top$.

Candidate If for some m, $m \to R_N \wedge \mathit{Bad}$, then add $\langle m, N\rangle$ to Q.

Decide If $\langle m, i+1\rangle \in Q$ and there are m_0 and m_1 s.t. $m_1 \to m$, $m_0 \wedge m_1'$ is satisfiable, and $m_0 \wedge m_1' \to \mathcal{F}(R_i) \wedge m'$, then add $\langle m_0, i\rangle$ to Q.

Conflict For $0 \leq i < N$: given a candidate model $\langle m, i+1\rangle \in Q$ and clause φ, such that $\neg\varphi \subseteq m$, if $\mathcal{F}(R_i \wedge \varphi) \to \varphi$, then add φ to R_j, for $j \leq i + 1$.

Leaf If $\langle m, i\rangle \in Q$, $0 < i < N$ and $\mathcal{F}(R_{i-1}) \wedge m'$ is unsatisfiable, then add $\langle m, i+1\rangle$ to Q.

Induction For $0 \leq i < N$, a clause $(\varphi \vee \psi) \in R_i$, $\varphi \notin R_{i+1}$, if $\mathcal{F}(R_i \wedge \varphi) \to \varphi$, then add φ to R_j, for each $j \leq i + 1$.

until ∞;

Algorithm 1: IC3/PDR.

tuple $\langle m, i\rangle$ where m is a monomial over $\boldsymbol{v}$ and $0 \leq i \leq N$. Intuitively, $\langle m, i\rangle$ means that a state m can reach a state in *Bad* in $N - i$ steps. Initially, Q is empty, $N = 0$ and $R_0 = \mathit{Init}$. Then, the rules are applied (possibly in a non-deterministic order) until either **Unreachable** or **Reachable** rule is applicable. **Unfold** rules extends the current trace and increases the level at which counterexample is searched. **Candidate** picks a set of bad states. **Decide** extends a counter-example from the queue by one step. **Conflict** blocks a counterexample and adds a new lemma. **Leaf** moves the counterexample to the next level. Finally, **Induction** generalizes a lemma inductively. A typical schedule of the rules is to first apply all applicable rules except for **Induction** and **Unfold**, followed by **Induction** at all levels, then **Unfold**, and then repeating the cycle.

Define *post* and *post** as follows:

$$\mathit{post}(R) = \exists \boldsymbol{v}_0 \cdot R(\boldsymbol{v}_0) \wedge \rho(\boldsymbol{v}_0, \boldsymbol{v}) \qquad \mathit{post}^*(R) = \bigvee_{0 \leq i < \omega} \mathit{post}^i(R) \tag{10}$$

The dual operators *pre* and *pre** are defined similarly. A direct consequence of Invariant 1 is that R_i over-approximates i applications of the forward image, e.g., R_i is an over-approximation of states reachable in at most i steps:

Proposition 1. $\bigvee_{j \leq i} \mathit{post}^j(\mathit{Init}) \;\to\; R_i$

Theorem 2. *If PDR (Alg. 1) returns from* **Reachable** *then property (4) holds. If PDR returns from* **Unreachable** *with* $R_{i+1} \to R_i$*, then* R_i *satisfies* (5).

We have omitted many important optimizations and generalizations instrumental for the efficiency of PDR. For example, when propagating the monomial

m in the **Decide** rule, it is useful to keep m_0 as general (i.e., weak) as possible to minimize backtracking during model search. Similarly, **Induction** can be applied to each new lemma created by the **Conflict** rule. These and other important insights are described in depth by others (e.g., [8, 19]).

4 $\mathcal{A}$PDR: PDR for Linear Real Arithmetic

In this section, we describe $\mathcal{A}$PDR, a generalization of PDR to Linear Real Arithmetic (LRA). The presentation is based on GPDR [19] and Spacer [21]. To our knowledge, this is the first complete description of $\mathcal{A}$PDR[1].

The input to $\mathcal{A}$PDR is a transition system $\langle \boldsymbol{v}, \mathit{Init}, \rho, \mathit{Bad}\rangle$, as in PDR, except that the variables $\boldsymbol{v}$ are rational and Init, Bad, and ρ are formulas in LRA. Naturally, the lemmas and the trace maintained by $\mathcal{A}$PDR are in LRA as well.

In principle, PDR as presented in Alg. 1 is applicable to LRA directly. However, **Decide** and **Conflict** rules are quite weak for LRA. In particular, they do not guarantee even a bounded progress of the algorithm – in LRA, PDR might diverge within a fixed level [21].

$\mathcal{A}$PDR extends PDR with two new rules, **Decide**$^{\mathcal{A}}$ and **Conflict**$^{\mathcal{A}}$ that replace **Decide** and **Conflict** rules, respectively. The new rules are shown in Algorithm 2. In the rules, we use P and $P_\downarrow$ to indicate a conjunction and $P^\uparrow$ a disjunction of linear inequalities, respectively. The **Decide**$^{\mathcal{A}}$ is based on *Model Based Projection* (Mbp) that under-approximates existential quantification. MBP was introduced in [21] and is defined as follows. Let φ be a formula, $U \subseteq \mathit{Vars}(\varphi)$ a subset of variables of φ, and P a model of φ. Then, $\psi \in \textsc{Mbp}(U, P, \varphi)$ is a model based projection if (a) ψ is a monomial, (b) $\mathit{Vars}(\psi) \subseteq \mathit{Vars}(\varphi) \setminus U$, (c) $P \models \psi$, (d) $\psi \rightarrow \exists V \cdot \varphi$. Furthermore, for a fixed U and a fixed φ, Mbp is finite. In [21], an MBP function is defined for LRA based on Loos-Weispfenning quantifier elimination. Note that finiteness of MBP ensures that **Decide**$^{\mathcal{A}}$ can only be applied finitely many times for a fixed set of lemmas R_i.

The **Conflict**$^{\mathcal{A}}$ rule is based on *Craig interpolation* (Itp). Given two formulas $A[\boldsymbol{x}, \boldsymbol{z}]$ and $B[\boldsymbol{y}, \boldsymbol{z}]$ such that $A \wedge B$ is unsatisfiable, a Craig interpolant $I[\boldsymbol{z}] = \textsc{Itp}(A[\boldsymbol{x}, \boldsymbol{z}], B[\boldsymbol{y}, \boldsymbol{z}])$, is a formula such that $A[\boldsymbol{x}, \boldsymbol{z}] \rightarrow I[\boldsymbol{z}]$ and $I[\boldsymbol{z}] \rightarrow \neg B[\boldsymbol{y}, \boldsymbol{z}]$. Note that in the context of **Conflict**$^{\mathcal{A}}$, B is always a monomial. In this case, we further require that the interpolant is a clause (i.e., a negation of a monomial). An algorithm for extracting LRA clause interpolants from the theory lemmas produced during DPLL(T) proof is given in [19]. There is an important difference between **Conflict** and **Conflict**$^{\mathcal{A}}$ rules. While by the definition of Itp, in **Conflict**$^{\mathcal{A}}$ $\mathcal{F}(R_i) \rightarrow P^\uparrow$, the corresponding requirement of **Conflict** is weaker: $\mathcal{F}(R_i \wedge P^\uparrow) \rightarrow P^\uparrow$. It is not clear how to extend this to LRA.

An appealing feature of PDR is that it generates separate lemmas to block spurious counter-examples. These lemmas can be strengthened and leverage mutual induction. In propositional PDR, the space of lemmas is bounded by the

[1] Previous versions omit important aspects of IC3, such as priority queues, inductive blocking. The addition of model based projection helps ensuring termination at fixed levels.

Decide$^{\mathcal{A}}$ If $\langle P, i+1\rangle \in Q$ and there is a model $m(\boldsymbol{v}, \boldsymbol{v}')$ s.t. $m \models \mathcal{F}(R_i) \wedge P'$, add $\langle P_\downarrow, i\rangle$ to Q, where $P_\downarrow \in \textsc{Mbp}(\boldsymbol{v}', m, \mathcal{F}(R_i) \wedge P')$.

Conflict$^{\mathcal{A}}$ For $0 \leq i < N$, given a counterexample $\langle P, i+1\rangle \in Q$ s.t. $\mathcal{F}(R_i) \wedge P'$ is unsatisfiable, add $P^\uparrow = \textsc{Itp}(\mathcal{F}(R_i)(\boldsymbol{v}_0, \boldsymbol{v}), P)$ to R_j for $j \leq i+1$.

Algorithm 2: $\mathcal{A}$PDR.

number of propositional variables. This guarantees convergence. Clearly, this is not the case for arithmetic. However, we can show that $\mathcal{A}$PDR guarantees to explore increasingly longer execution paths.

Theorem 3. *In any infinite execution of $\mathcal{A}$PDR, the rule **Unfold** is enabled infinitely often.*

Several other approaches have been suggested to lift IC3 to arithmetic. [9] extracts lemmas as a side-effect of an incremental quantifier-elimination procedure that enumerates satisfiable cubes, then eliminates variables from the cubes; [20] develops IC3 for timed automata. More recent attention has been focused on combination with predicate abstraction and arithmetic [10, 7]. The abstraction is refined (using interpolants) if the concrete interpretation is able to strengthen inductive lemmas or block abstract counter-examples, otherwise preference is given to a search over existing abstract predicates. In this setting, the interpolation queries also include formulas from the abstract domain.

5 $\mathcal{F}$PDR: Deriving Convex Invariants

In this section, we present our first major contribution – an algorithm, called $\mathcal{F}$PDR, to compute convex invariants. The algorithm terminates when it either finds a convex polyhedral invariant, or an abstract counter-example that cannot be refuted by the best polyhedral abstract transformer $post_\alpha$. Conceptually, the main difference between $\mathcal{F}$PDR and $\mathcal{A}$PDR is that $\mathcal{F}$PDR uses an abstract post-image $post_\alpha$ instead of the concrete *post* of $\mathcal{A}$PDR. Furthermore, $\mathcal{F}$PDR restricts $R_0, \ldots, R_N$ to be convex polyhedra, i.e., conjunctions of linear inequalites. $\mathcal{F}$PDR uses the same data structures as $\mathcal{A}$PDR but maintains a stronger invariant:

Invariant 2 ($\mathcal{F}$PDR) *$\neg Bad \leftarrow R_i \rightarrow R_{i+1} \leftarrow post_\alpha(R_i)$ and for $0 \leq i \leq N$, R_i are convex polyhedra.*

To realize $\mathcal{F}$PDR, we extend $\mathcal{A}$PDR with two new rules, **Conflict$^{\mathcal{F}}$** and **Decide$^{\mathcal{F}}$** shown in Alg. 3. The new rules create abstract counter-example traces that may not correspond to concrete traces. We differentiate abstract states by inserting them into AQ instead of Q, which is not used in $\mathcal{F}$PDR.

To understand the rules, recall that the best abstract transformer for polyhedra is defined as $post_\alpha(R_i)[\boldsymbol{v}] = CC(\exists \boldsymbol{v}_0 \cdot \mathcal{F}(R_i)(\boldsymbol{v}_0, \boldsymbol{v}))$. The only difference

Data: AQ a queue of abstract counter-examples. Initially, $AQ = \emptyset$.
Reachable$^{\mathcal{F}}$ If there is an m s.t. $\langle m, 0\rangle \in AQ$ return *AbstractReachable*.
Decide$^{\mathcal{F}}$ If $\langle P, i+1\rangle \in AQ$ and there is a model $m(\boldsymbol{v}, \boldsymbol{v}')$ s.t.
$m \models CC(\mathcal{F}(R_i)) \wedge P'$, add $\langle P_\downarrow, i\rangle$ to AQ, where
$P_\downarrow = \textsc{Mbp}(\boldsymbol{v}', m, CC(\mathcal{F}(R_i)) \wedge P')$.
Conflict$^{\mathcal{F}}$ For $0 \leq i < N$, given a counterexample $\langle P, i+1\rangle \in AQ$ s.t.
$CC(\mathcal{F}(R_i)) \wedge P'$ is unsatisfiable, add $P^\uparrow = \textsc{HalfItp}(CC(\mathcal{F}(R_i))(\boldsymbol{v}_0, \boldsymbol{v}), P)$ to R_j for $j \leq i+1$.

Algorithm 3: $\mathcal{F}$PDR.

between $\mathcal{F}$PDR and $\mathcal{A}$PDR rules is that $\mathcal{F}$PDR uses convex closure of the formulas representing the post-image. Furthermore, the **Conflict**$^{\mathcal{F}}$ rule uses half-space interpolant $\textsc{HalfItp}(A, B)$ of [2] that restricts interpolants to a single inequality (i.e., a half-space). **Conflict**$^{\mathcal{F}}$ is well defined because both A and B are convex. Hence, by Farkas lemma, there exists a half-space separating them. Invariant 2 follows immediately from the rules.

In the rest of this section, we establish the main properties of $\mathcal{F}$PDR show how to implement the rules in Alg. 3 efficiently, and, discuss the relationship between $\mathcal{F}$PDR and polyhedral Abstract Interpretation.

5.1 Properties

$\mathcal{F}$PDR over-approximates the abstract iteration sequence (1).

Proposition 2. *Let $R_0, \ldots, R_N$ be a trace of $\mathcal{F}$PDR and $0 \leq i \leq N$. Then, $\left(\bigsqcup_{j \leq i} post_\alpha^j(Init)\right) \rightarrow R_i$.*

Proposition 2 is an immediate consequence of **Conflict**$^{\mathcal{F}}$ rule. Note the analogy with Proposition 1.

Since the abstract post-image over-approximates the concrete post-image, whenever $\mathcal{F}$PDR returns from **Unreachable**, it has found a concrete inductive invariant that certifies that *Bad* is unreachable from *Init*.

Proposition 3. *Let $R_0, \ldots, R_N$ be a trace of $\mathcal{F}$PDR and $0 < i \leq N$ be such that $R_i \rightarrow R_{i-1}$, then $post^*(Init) \cap Bad = \emptyset$*

Finally, $\mathcal{F}$PDR returns from **Reachable**$^{\mathcal{F}}$ only if there does not exist an unreachability certificate that can be established using the best abstract post-image That is, every abstract iteration sequence (1), independently of the widening operator or other strategy heuristics, reaches a bad state.

Proposition 4. *Traces found by $\mathcal{F}$PDR are contained in the abstraction:*

$$\langle P, 0\rangle \in AQ \quad \textit{implies } post_\alpha^N(Init) \cap Bad \neq \emptyset.$$

Proof. By construction, $\langle P, N\rangle \cap Bad \neq \emptyset$. Then, by induction on the size of N that $\langle P, i\rangle \in AQ$ implies that $post_\alpha^{N-i}(P) \cap Bad \neq \emptyset$. □

Propositions 2 and 3 establish soundness of $\mathcal{F}$PDR. Proposition 4 provides an interesting form of completeness: $\mathcal{F}$PDR is guaranteed to terminate when the polyhedral abstract domain is too weak to refute a counterexample (i.e., a false alarm). However, $\mathcal{F}$PDR might still diverge when *Bad* is unreachable even if the abstract domain is strong enough to refute every finite counterexample.

5.2 Implementation

The main bottleneck in implementing the $\mathcal{F}$PDR rules in Alg. 3 is deciding satisfiability $CC(\varphi) \wedge P$ of a convex closure $CC(\varphi)$ of an arbitrary formula φ and a monomial P, where both φ and P are over LRA. A naïve algorithm is to (a) compute a DNF of φ, (b) compute the convex closure $\psi = CC(\varphi)$ of the disjuncts, and (c) check satisfiability of $\psi \wedge P$. This however, is not efficient: both the explicit computation of the DNF and the convex closure are exponential in the size of φ. Instead, we propose a novel algorithm CCSAT that avoids an explicit convex closure computation by a combination of the syntactic convex closure construction and interpolation.

The pseudo-code for algorithm CCSAT(φ, P) is shown in Alg. 4. The inputs to CCSAT are a formula $\varphi[\boldsymbol{v}, \boldsymbol{v}']$ and a monomial $P[\boldsymbol{v}]$. The output is either UNSAT and an interpolant between $CC(\varphi)$ and P, or SAT and a model-based projection of $\boldsymbol{v}$ from $CC(\varphi) \wedge P$. CCSAT replaces an expensive up-front convex closure computation with an iterative approximation using syntactic convex closure *cc* (see Def. 1). The algorithm maintains the set M of implicants of φ such that $CC(M)$ under-approximates $CC(\varphi)$. In each iteration, checking whether $CC(M)$ and φ are consistent is reduced to an SMT-check using the syntactic representation $cc(M)$ of the convex closure $CC(M)$. Note that $cc(M)$ is an SMT-formula that is linear in $|M|$ and is easy to compute. If $cc(M)$ and P are consistent, their model is used to derive the model-based projection. Otherwise, interpolation is used to construct an over-approximation $P^{\uparrow}$ of $cc(M)$. Crucially, since both $cc(M)$ and P are monomials, even a general interpolation procedure ITP of [19] guarantees that $P^{\uparrow}$ is a half-space. Thus, no special HALFITP procedure is needed. If $P^{\uparrow}$ contains φ, then $P^{\uparrow}$ is an interpolant between $CC(\varphi)$ and P', and CCSAT terminates. Otherwise, CCSAT picks another implicant m of φ that contains at least one point outside of $P^{\uparrow}$, adds it to M, and repeats the loop.

Example 2. We illustrate a run of CCSAT(φ, P), where $\varphi[x, y]$ and $P[x, y]$ are defined as follows:

$$\varphi \equiv ((0 \leq y \leq 1) \wedge (0 \leq x \leq 4) \wedge (x \leq 1 \vee x \geq 2)) \vee ((2 \leq y \leq 3) \wedge (2 \leq x \leq 3))$$
$$P \equiv x = 5 \wedge y = 4$$

First, an implicant $m_1 = (0 \leq y \leq 1) \wedge (0 \leq x \leq 1)$ is chosen and blocked by $P_1^{\uparrow} = (y \leq 3)$. Second, $m_2 = (2 \leq x \leq 3) \wedge (2 \leq y \leq 3)$ is chosen and blocked by $P_2^{\uparrow} = (x \leq 4)$. Since $\varphi \rightarrow P_2^{\uparrow}$, the algorithm terminates with (UNSAT, $P_2^{\uparrow}$).

Input: $\varphi[\boldsymbol{v}, \boldsymbol{v}'], P[\boldsymbol{v}]$
$M \leftarrow \emptyset$
while $cc(M) \wedge P' \models \bot$ **do**
  $P^{\uparrow}[\boldsymbol{v}'] \leftarrow \textsc{Itp}(cc(M), P')$
  if $\varphi \wedge \neg P^{\uparrow}[\boldsymbol{v}'] \models \bot$ **then**
    return UNSAT, $P^{\uparrow}[\boldsymbol{v}]$
  else
    $m \leftarrow \mathit{implicant}(\varphi)$ such that $m \wedge \neg P^{\uparrow}[\boldsymbol{v}'] \not\models \bot$
    $M \leftarrow M \cup \{m\}$
  end
end
let m be s.t. $m \models cc(M) \wedge P'$
$P_{\downarrow} \leftarrow \textsc{Mbp}(\boldsymbol{v}', m, cc(M) \wedge P')$
return SAT, $P_{\downarrow}$

Algorithm 4: CCSAT: Decides satisfiability of $CC(\varphi) \wedge P'$. It produces either a half-interpolant or a model-based projection.

The soundness of CCSAT follows immediately from the exit condition of the while loop. Running time is bounded by the number of distinct propositional implicants of φ.

Proposition 5. CCSAT *terminates.*

Proof. For all $m_i, m_j \in M$, by construction, there exists a polyhedron $P^{\uparrow}$ such that $m_i \to P^{\uparrow}$ and $m_j \to \neg P^{\uparrow}$. Thus, all elements of M are distinct. Furthermore, φ has only finitely many distinct propositional implicants. □

The rules in Alg 3 are implemented by first using CCSAT to decide whether $CC(\mathcal{F}(R_i)) \wedge P$ is satisfiable, and then applying either the **Decide**$^{\mathcal{F}}$ or the **Conflict**$^{\mathcal{F}}$ rule, as applicable.

In conclusion, we remark that CCSAT is interesting in its own right as an alternative algorithm for computing half-space (or *beautiful*) interpolants of [2]. In particular, let φ be a formula and $P_0, \ldots, P_k$ be monomials over LRA. Then, $\text{CCSAT}(\varphi, cc(\{P_0, \ldots, P_k\}))$ is a half-space interpolant of φ and $\bigvee_{i=0}^{k} P_i$, if such an interpolant exists.

5.3 Discussion

What is the relationship between $\mathcal{F}$PDR and the traditional Kleene iteration sequence (1)? Both compute convex invariants, but can one simulate the other? Let K be a natural number. For simplicity, consider a convergent Kleene sequence $Y_0, \ldots, Y_K$ in which widening is only applied at the last step. That is, $\forall i \geq k \cdot Y_i = Y_K$, and $W = \{K\}$. Similarly, take an N-step execution of $\mathcal{F}$PDR with $N \geq K$, so that R_K is well defined. Let $\mathit{Inv}(R_K)$ stand for an inductive subset of R_K, i.e., a subset that satisfies the first equation of (5). We are interested in two questions: (Q1) given K and a run of $\mathcal{F}$PDR, is there a Kleene sequence such that $Y_K = \mathit{Inv}(R_K)$; and (Q2) given K and a convergent Kleene sequence, is there a run of PDR such that $Y_K = Inv(R_K)$. While we do not give complete answers, in the rest of the section we explore some special cases.

We use the following transition system as a running example:

$$Init(x,y,z) \equiv x - y \leq 0 \wedge x + y \leq 0 \wedge z = 1/2 \quad Bad(x,y,z) \equiv x \geq 2 \tag{11}$$

$$\begin{aligned}\rho(x,y,z,x',y',z') \equiv y' = y \wedge (x \leq 1 \rightarrow x' = x + z \wedge z' = 1/2 \times z) \wedge \\ (x > 1 \rightarrow x' = x \wedge z' = z)\end{aligned} \tag{12}$$

Note that the set of reachable states is $(x - y < 1) \wedge (x + y < 1)$.

First, consider an execution of $\mathcal{F}$PDR that converges with an inductive invariant $x \leq 3/2 \wedge z \leq 1/2$. A Kleene sequence with standard widening cannot converge on this invariant for any value of K. In particular, the strongest Y_i is of the form $x - y \leq s(i) \wedge x + y \leq s(i)$, where $s(i) = \sum_{j=1}^{j \leq i} 2^{-j}$. Since the standard widening only drops constraints, any Kleene sequence converges to $\top$. The key difference here is that the Kleene iteration with standard widening is restricted to the faces of the polyhedra appearing in the sequence Y_i, while $\mathcal{F}$PDR is limited only by interpolation (i.e., any linear combinations of constraints appearing in R_{K-1} and in the transition relation ρ). In this particular example, other choices for widening can easily simulate $\mathcal{F}$PDR. Moreover, with a suitable (but not necessarily efficiently computable) widening operator, a Kleene sequence can simulate any other method for discovering convex invariants.

Second, consider a variant of the example above, where z is not changed: i.e, replace $z' = 1/2 \times z$ by $z' = z$ in (12). In this example, Kleene iteration converges to the exact set of reachable states in 2 steps. No widening is required. On the other hand, $\mathcal{F}$PDR, as presented, does not simulate the Kleene iteration. Once again, the issue is that $\mathcal{F}$PDR is not restricted to the faces of the polyhedra involved. In fact, our formulation of the **Conflict**$^{\mathcal{F}}$ rule further restricts the set of lemmas to half-spaces of the form $P^{\uparrow} = \textsc{HalfItp}(\varphi, P)$. Alternatively, we can redefine **Conflict**$^{\mathcal{F}}$ to use $P^{\uparrow} = \textsc{PolyItp}(\varphi, P)$, where $\textsc{PolyItp}(A, B)$ is a polyhedral interpolant consisting of some faces of A (we assume that A is convex). Note that $\textsc{PolyItp}$ can be implemented, for example, by quantifying out local variables from the subset of A inconsistent with B. We believe that with this redefinition of **Conflict**$^{\mathcal{F}}$, $\mathcal{F}$PDR can simulate the Kleene iteration. However, an efficient implementation of $\textsc{PolyItp}$ that avoids explicit quantifier elimination remains open. In summary, $\mathcal{F}$PDR and Kleene iteration are quite distinct algorithms for computing convex inductive invariants. Their existing implementation are unlikely to simulate one another. We leave further theoretical and practical exploration of this question to future work.

We conclude this section with an interesting connection between $\mathcal{F}$PDR and widening refinement for AI (e.g., [16, 1]). While there is no explicit widening in $\mathcal{F}$PDR, it is implicit in the choice of half-spaces added by **Conflict**$^{\mathcal{F}}$. Whenever some half-spaces are not added in a given iteration (i.e., too much widening), further iterations refine the trace, until all imprecisions introduced by a sub-optimal choices in all previous applications of the **Conflict**$^{\mathcal{F}}$ rule are removed. This mimics the more elaborate algorithms of [16, 1].

6 $\mathcal{B}$PDR: co-convex invariants

Not all necessary invariants can be expressed as convex polyhedra. Take for example,

$$\begin{aligned} &\mathit{Init} \equiv x = y = 0 \quad \mathit{Bad} \equiv x > 1000 \ \wedge \ y > 1000 \\ &\rho \equiv (x < 100 \vee y < 100) \wedge x' \leq x + 1 \wedge y' \leq y + 1 \end{aligned}$$

The inductive invariant $x \leq 100 \vee y \leq 100$ is not convex, but its complement is. We call such invariants *co-convex*. In this section, we devise a property directed algorithm $\mathcal{B}$PDR, that finds co-convex invariants. Dually to $\mathcal{F}$PDR, $\mathcal{B}$PDR mimics chaotic iteration (2) with the best abstract pre-image.

The rules for $\mathcal{B}$PDR, are shown in Alg. 5. As before, the algorithm maintains a trace $R_0, \ldots, R_N$, but each R_i is restricted to a *single clause* (disjunction of inequalities). We assume that Bad is convex, otherwise, take $CC(\mathit{Bad})$ as the new set of bad states. Thus, $\neg \mathit{Bad}$ is co-convex. $\mathcal{B}$PDR maintains the following invariant:

Invariant 3 $\neg \mathit{Bad} \leftarrow \neg CC(S) \leftarrow R_i \rightarrow R_{i+1} \leftarrow \mathcal{F}(R_i)$. $\forall 0 \leq i \leq N \cdot R_i$ *is co-convex.*

$\mathcal{B}$PDR is based on the observation that the transitive closure $\mathit{pre}^*_\alpha(\mathit{Bad})$ of the abstract pre-image is convex. Thus, instead of maintaining a *queue* Q of counterexamples, $\mathcal{B}$PDR maintains a *set* S s.t. the convex closure $CC(S)$ of S under-approximates $\mathit{pre}^*_\alpha(\mathit{Bad})$, i.e., $CC(S) \subseteq \mathit{pre}^*_\alpha(\mathit{Bad})$. In each iteration, $\mathcal{B}$PDR either extends S by adding a state that reaches the convex closure of S in 1 or 0 steps (**Decide**$^{\mathcal{B}}$ and **Candidate**$^{\mathcal{B}}$ rules), or strengthen some R_i (**Conflict**$^{\mathcal{B}}$ rule). Since there is no queue, **Reachable**$^{\mathcal{B}}$ checks whether there are states in the intersection of Init and convex closure $CC(S)$ of bad-reaching states. Furthermore, **Leaf** is unnecessary and **Induction** is disabled. **Decide**$^{\mathcal{B}}$ is very similar to **Decide**$^{\mathcal{F}}$ of $\mathcal{F}$PDR. The only difference is that convex closure is applied to the bad states. **Conflict**$^{\mathcal{B}}$ is more complex. First, since there is no queue of counterexamples, we must find the smallest i at which the rule is applicable. Second, since the trace R_i of $\mathcal{B}$PDR is restricted to single clauses, the rule can only change the content of R_i. To guarantee monotonicity of the trace, we stutter the transition relation, i.e., we use $R'_{i-1} \vee \mathcal{F}(R_{i-1})$ as the transformer instead of $\mathcal{F}(R_{i-1})$. Finally, we compute lemmas by *backward* interpolation. We let the bad states be the A-part of the interpolant, and use the backward interpolation property: $I = \textsc{Itp}(A, B)$ iff $\neg I = \textsc{Itp}(B, A)$. Note that since $CC(S)$ is convex, the interpolant $P^\uparrow$ is convex, and the backward interpolant $\neg P^\uparrow$ is co-convex.

Unlike $\mathcal{F}$PDR, implementing $\mathcal{B}$PDR rules is straightforward. Since in Alg. 5 CC is only applied to the set S of convex polyhedra, all applications of CC are simply replaced by its syntactic version cc.

$\mathcal{B}$PDR satisfies similar properties to $\mathcal{F}$PDR, but relative to the pre-image. In particular, whenever $\mathcal{B}$PDR returns from **Unreachable**, it has found a concrete inductive invariant:

Reachable$^{\mathcal{B}}$ If $Init \wedge CC(S)$ is satisfiable, return $AbstractReachable$
Candidate$^{\mathcal{B}}$ If for some P, $P \to R_N \wedge Bad$, then $S \leftarrow S \cup \{P\}$.
Decide$^{\mathcal{B}}$ If there is an $0 < i \leq N$ and a model $m(\boldsymbol{v}, \boldsymbol{v}')$ s.t.
$m \models \mathcal{F}(R_i) \wedge CC(S)'$, then $S \leftarrow S \cup \{P_\downarrow\}$, where
$P_\downarrow = \textsc{Mbp}(\boldsymbol{v}', m, \mathcal{F}(R_i) \wedge CC(S)')$.
Conflict$^{\mathcal{B}}$ If there exists a minimal $0 < i \leq N$ s.t.
$(R'_{i-1} \vee \mathcal{F}(R_{i-1})) \wedge CC(S)' \models \bot$. Then, $R_i \leftarrow \neg P^\uparrow[\boldsymbol{v}]$, $N \leftarrow i+1$, and
$R_N \leftarrow \top$, where $P^\uparrow[\boldsymbol{v}'] = \textsc{Itp}(CC(S)', R'_{i-1} \vee \mathcal{F}(R_{i-1}))$.

Algorithm 5: $\mathcal{B}$PDR.

Proposition 6. *Let $R_0, \ldots, R_N$ be a trace of $\mathcal{B}$PDR and $0 < i \leq N$ be such that $R_i \to R_{i+1}$, then $Init \cap pre^*(Bad) = \emptyset$.*

Similarly, $\mathcal{B}$PDR returns from **Reachable**$^{\mathcal{B}}$ only if there is no invariant that can be established using best abstract pre-image. That is, every backward chaotic iteration sequence (2) started from *Bad* states, reaches a state in *Init*.

Proposition 7. *Traces found by $\mathcal{B}$PDR are contained in the abstraction:*

$$Init \cap CC(Bad) \neq \emptyset \quad \textit{implies} \quad Init \cap pre^*_\alpha(Bad) \neq \emptyset\,.$$

It is also interesting to see whether $\mathcal{B}$PDR simulates backward chaotic iteration. Here, the correspondence is much more direct. The choice of s_i in (2) is in one-to-one correspondence with the choice of $P_\downarrow$ in **Decide**$^{\mathcal{B}}$. Widening choices in (2) correspond to constraints dropped by the interpolation during computation of $P^\uparrow$ in **Conflict**$^{\mathcal{B}}$. In practice, the key difference is again in the choice of the lemmas found by interpolation. On one hand, the chaotic iteration with standard widening is restricted to the faces of the polyhedra involved. On the other hand, $\mathcal{B}$PDR is restricted to half-spaces found by interpolation.

7 Combinations

In the previous sections, we have presented 3 algorithms, $\mathcal{A}$PDR, $\mathcal{F}$PDR, and $\mathcal{B}$PDR, for computing linear, convex, and co-convex sufficient inductive invariants, respectively. In this section, we present a uniform framework that combines the three algorithms.

First, note that **Conflict**$^{\mathcal{B}}$ rule of $\mathcal{B}$PDR is significantly different from the corresponding rules of $\mathcal{A}$PDR and $\mathcal{F}$PDR. Unlike in $\mathcal{A}$PDR and $\mathcal{F}$PDR, **Conflict**$^{\mathcal{B}}$ only modifies one element R_i of the trace, and ensures that each R_i contains a single clause. This, however, is only necessary to prune the search space to be co-convex invariants. To unify $\mathcal{B}$PDR with the other algorithms, we replace **Conflict**$^{\mathcal{B}}$ with **Conflict**$^{\mathcal{AB}}$ shown in Alg. 6. Note that **Conflict**$^{\mathcal{AB}}$ still uses the convex closure $CC(S)$ of bad-reaching states S, but it adds the new lemma $P^\uparrow$ to all levels below i. $\mathcal{B}$PDR remains sound with the new rule. However, it

Conflict$^{\mathcal{AB}}$ If there exists a $0 < i \leq N$ s.t. $\mathcal{F}(R_{i-1}) \wedge CC(S)' \models \bot$.
$R_j \leftarrow R_j \wedge \neg P^{\uparrow}[\boldsymbol{v}]$ for $0 < j \leq i$, where $P^{\uparrow}[\boldsymbol{v}'] = \textsc{Itp}(CC(S)', \mathcal{F}(R_{i-1}))$.

Conflict$^{\mathcal{AFB}}$ If there exists a $0 < i \leq N$ s.t. $CC(\mathcal{F}(R_{i-1})) \wedge CC(S)' \models \bot$.
$R_j \leftarrow R_j \wedge P^{\uparrow}[\boldsymbol{v}]$ for $0 < j \leq i$, where
$P^{\uparrow}[\boldsymbol{v}'] = \textsc{HalfItp}(CC(\mathcal{F}(R_{i-1})), CC(S)')$.

Algorithm 6: Additional conflict rules for $\mathcal{B}$PDR.

no longer mimics backward chaotic iteration, and produces more than just co-convex invariants.

Second, we add a new rule **Conflict**$^{\mathcal{AFB}}$, shown in Alg. 6 that combines the corresponding rules of $\mathcal{F}$PDR and $\mathcal{B}$PDR by taking the convex closures of both the post-image and the bad-reaching states. Note that in this case, interpolation guarantees that the corresponding lemma is a single inequality (i.e., a half-space). The rule is implemented efficiently using CCSAT from Section 5.2.

Finally, the combined algorithm, called PolyPDR, is obtained by combining all the rules of PDR (Alg. 1), $\mathcal{A}$PDR (Alg. 2), $\mathcal{F}$PDR (Alg. 3), $\mathcal{B}$PDR (Alg. 5), and the new $\mathcal{B}$PDR rules (Alg. 6), except for **Conflict**$^{\mathcal{B}}$, **Reachable**$^{\mathcal{F}}$, and **Reachable**$^{\mathcal{B}}$. PolyPDR maintains 3 kinds of counterexamples: a queue of concrete counterexamples Q from PDR, a queue of abstract counterexamples AQ from $\mathcal{F}$PDR, and a set of abstract counterexamples S from $\mathcal{B}$PDR. States from Q can reach a state in *Bad*, states in AQ can abstractly reach a state in *Bad* via the abstract post-image, and states in S are reachable from *Bad* via the abstract pre-image. The soundness of PolyPDR follows directly from the soundness of individual algorithms: it either finds a concrete counterexample in Q, or finds a concrete or an abstract sufficient inductive invariant.

We suggest two schemes to apply the rules of PolyPDR to combine the effects of abstract and concrete reasoning: *pre-processing* and *in-processing*. The pre-processing scheme starts with enabling only the rules of $\mathcal{F}$PDR and $\mathcal{B}$PDR, and applying them until either the algorithm terminates, or the pre-conditions of **Reachable**$^{\mathcal{F}}$ or **Reachable**$^{\mathcal{B}}$ become true (i.e., an abstract counterexample is found). Then, the rules of $\mathcal{F}$PDR and $\mathcal{B}$PDR are disabled and the rules of $\mathcal{A}$PDR are enabled. This scheme is similar to first running an abstract interpreter to discover an inductive invariant, and then using $\mathcal{A}$PDR to strengthen it or find a counterexample. The two stages, abstract and concrete, communicate by the lemmas learned in the trace.

The in-processing scheme also starts with enabling only $\mathcal{F}$PDR and $\mathcal{B}$PDR rules. Then, whenever the pre-conditions for **Reachable**$^{\mathcal{F}}$ or **Reachable**$^{\mathcal{B}}$ become true, abstract counterexamples AQ and S are reset. Next, the control is given to $\mathcal{A}$PDR rules, until the **Unfold** rule is applied. At this point, the $\mathcal{A}$PDR rules are disabled, the rules of $\mathcal{F}$PDR and $\mathcal{B}$PDR are enabled, and the cycle repeats. This scheme mimics the abstraction-refinement loop of Vinta [1]. First, an abstract interpreter is used to compute an inductive, but not (necessarily) sufficient invariant. Then, the concrete reasoning is used to refine the invariant and rule out false alarms. Whenever the concrete strengthening is not inductive, the abstract reasoning is repeated starting from it. Again, the communication between the abstract and concrete reasoning is captured by the lemmas computed in the trace.

8 Evaluation

We have implemented variants of $\mathcal{F}$PDR and $\mathcal{B}$PDR algorithms in Z3. For the $\mathcal{F}$PDR variant, we have extended $\mathcal{A}$PDR with the $\textbf{Decide}^{\mathcal{F}}$ rule, but not the $\textbf{Conflict}^{\mathcal{F}}$ rule. This makes our $\mathcal{F}$PDR algorithm a generalization step for $\mathcal{A}$PDR. Whenever a candidate model is blocked by $\textbf{Conflict}^{\mathcal{A}}$, we check whether the learned lemma $P^{\uparrow}$ can be generalized to be convex. For the $\mathcal{B}$PDR variant, we have implemented a hybrid algorithm by adding the rule $\textbf{Conflict}^{\mathcal{AB}}$ to $\mathcal{A}$PDR. Furthermore, our $\mathcal{B}$PDR implementation is limited to the incomplete projection-based generalization strategy of [19], instead of the complete MBP-based strategy presented here. Hence, it sometimes diverges without making progress (i.e., gets into an infinite execution in which **Unfold** rule is never applied). Our implementation and benchmarks are available in the `cc` branch of `https://z3.codeplex.com/SourceControl/network/forks/arie/zag`.

To answer the main question posed in the Introduction, we have selected several benchmarks that are easy for polyhedral abstraction, but are hard for PDR-based approaches, from [2] and Z3 regression test suite. While the examples are small, they illustrate well the benefits of the new approach. The results are summarized in Fig. 2. In the figure, ϵ and ∞ mean "solved in under a second" and "did not terminate", respectively. In all cases, except for `ev`-series of examples, Z3 was configured with the default configuration options and restricted to Linear Arithmetic (an optional UTVPI solver was disabled using `fixedpoint.use_utvpi=false` command line option). For `ev`-series, Z3 is further restricted to projection-based generalization strategy of [19] using command line option `fixedpoint.use_model_generalizer=true`. The original Z3 algorithm diverges on all examples except for `d03` and `david`. $\mathcal{F}$PDR performs the best. However, generalizing using convex closures interferes with default algorithm for lemma generation in Z3. This makes `david` hard for $\mathcal{F}$PDR. $\mathcal{B}$PDR often diverges. For some cases (`ev`, `ev1`) this is due to the fact that *Bad* is not convex. For others (`xyz2`, `addadd`) this is a problem with our use of projection-based generalization. Finally, the `gcnr` example, originally from [16], and also used in [22, 2], remains unsolved.

Name	Z3	$\mathcal{F}$PDR	$\mathcal{B}$PDR
`addadd`	∞	ϵ	∞
`d03`	ϵ	ϵ	ϵ
`david`	ϵ	∞	ϵ
`ev-down`	∞	ϵ	ϵ
`ev-up`	∞	ϵ	ϵ
`ev`	∞	ϵ	∞
`ev1`	∞	ϵ	∞
`updown`	∞	ϵ	ϵ
`xyz`	∞	ϵ	ϵ
`xyz2`	∞	ϵ	∞
`gcnr`	∞	∞	∞

Fig. 2. Results.

We believe that this evaluation, albeit limited and preliminary, demonstrates the advantages of our framework. It shows the clear benefits of integrating polyhedral abstraction as a component within $\mathcal{A}$PDR.

9 Summary

This paper developed property directed model checking procedures using polyhedral abstraction. We showed how to combine syntactic convex closures with interpolation to incrementally compute abstractions, and we correspondences between Kleene, chaotic abstract interpretation and property directed reachability. We evaluated the new approaches on exemplary benchmarks. This work sheds furter light on the synergy of polyhedral abstraction and interpolation-based model checking.

References

1. A. Albarghouthi, A. Gurfinkel, and M. Chechik. Craig Interpretation. In *SAS*, pages 300–316, 2012.
2. A. Albarghouthi and K. L. McMillan. Beautiful interpolants. In Sharygina and Veith [27], pages 313–329.
3. R. Bagnara, P. M. Hill, and E. Zaffanella. The parma polyhedra library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *CoRR*, abs/cs/0612085, 2006.
4. R. Bagnara, P. M. Hill, and E. Zaffanella. Widening operators for powerset domains. *STTT*, 9(3-4):413–414, 2007.
5. F. Benoy, A. King, and F. Mesnard. Computing Convex Hulls with a Linear Solver. *TPLP*, 5(1-2):259–271, 2005.
6. A. Biere and R. Bloem, editors. *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*, volume 8559 of *Lecture Notes in Computer Science*. Springer, 2014.
7. J. Birgmeier, A. R. Bradley, and G. Weissenbacher. Counterexample to induction-guided abstraction-refinement (CTIGAR). In Biere and Bloem [6], pages 831–848.
8. A. R. Bradley. SAT-Based Model Checking without Unrolling. In *VMCAI*, pages 70–87, 2011.
9. A. Cimatti and A. Griggio. Software Model Checking via IC3. In *CAV*, pages 277–293, 2012.
10. A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. IC3 Modulo Theories via Implicit Predicate Abstraction. In *TACAS*, pages 46–61, 2014.
11. A. Cimatti, A. Griggio, and R. Sebastiani. Efficient Generation of Craig Interpolants in Satisfiability Modulo Theories. *ACM Trans. Comput. Log.*, 12(1):7, 2010.
12. P. Cousot and R. Cousot. Abstract Interpretation Frameworks. *J. Log. Comput.*, 2(4):511–547, 1992.
13. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In A. V. Aho, S. N. Zilles, and T. G. Szymanski, editors, *POPL*, pages 84–96. ACM Press, 1978.

14. L. M. de Moura and D. Jovanovic. A Model-Constructing Satisfiability Calculus. In R. Giacobazzi, J. Berdine, and I. Mastroeni, editors, *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, volume 7737 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2013.
15. S. Grebenshchikov, N. P. Lopes, C. Popeea, and A. Rybalchenko. Synthesizing software verifiers from proof rules. In *PLDI*, 2012.
16. B. S. Gulavani, S. Chakraborty, A. V. Nori, and S. K. Rajamani. Refining abstract interpretations. *Inf. Process. Lett.*, 110(16):666–671, 2010.
17. A. Gurfinkel and S. Chaki. Boxes: A Symbolic Abstract Domain of Boxes. In R. Cousot and M. Martel, editors, *SAS*, volume 6337 of *Lecture Notes in Computer Science*, pages 287–303. Springer, 2010.
18. N. Halbwachs. *Détermination automatique de relations linéaires vérifiées par les variables d'un programme.* PhD thesis, Grenoble, 1979.
19. K. Hoder and N. Bjørner. Generalized Property Directed Reachability. In *SAT*, pages 157–171, 2012.
20. R. Kindermann, T. A. Junttila, and I. Niemelä. SMT-Based Induction Methods for Timed Systems. In M. Jurdzinski and D. Nickovic, editors, *FORMATS*, volume 7595 of *Lecture Notes in Computer Science*, pages 171–187. Springer, 2012.
21. A. Komuravelli, A. Gurfinkel, and S. Chaki. SMT-Based Model Checking for Recursive Programs. In Biere and Bloem [6], pages 17–34.
22. A. Komuravelli, A. Gurfinkel, S. Chaki, and E. M. Clarke. Automatic Abstraction in SMT-Based Unbounded Software Model Checking. In Sharygina and Veith [27], pages 846–862.
23. K. Korovin and A. Voronkov. Solving Systems of Linear Inequalities by Bound Propagation. In N. Bjørner and V. Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2011.
24. K. L. McMillan. Lazy annotation revisited. In *CAV*, pages 243–259, 2014.
25. X. Rival and L. Mauborgne. The trace partitioning abstract domain. *ACM Trans. Program. Lang. Syst.*, 29(5), 2007.
26. P. Rümmer, H. Hojjat, and V. Kuncak. Disjunctive interpolants for horn-clause verification. In *CAV*, pages 347–363, 2013.
27. N. Sharygina and H. Veith, editors. *CAV*, volume 8044 of *Lecture Notes in Computer Science*. Springer, 2013.