Generalized Property Directed Reachability

Kryštof Hoder[(1)] and Nikolaj Bjørner[(2)]

(1) The University of Manchester (2) Microsoft Research, Redmond

Abstract. The IC3 algorithm was recently introduced for proving properties of finite state reactive systems. It has been applied very successfully to hardware model checking. We provide a specification of the algorithm using an abstract transition system and highlight its dual operation: model search and conflict resolution. We then generalize it along two dimensions. Along one dimension we address nonlinear fixed-point operators (push-down systems) and evaluate the algorithm on Boolean programs. In the second dimension we leverage proofs and models and generalize the method to Boolean constraints involving theories. [1]

1 Introduction

Efficient SAT and SMT solvers are at the heart of many program analysis, verification and test tools. Such tools reduce program representations and logics to first-order or propositional queries. An ongoing quest is how one can raise the level of abstraction and power of the logic engines. We pursue satisfiability modulo least fixed-points. The propositional fragment corresponds to Boolean Programs with procedure calls, equivalently monotone Datalog, and first-order existential fixed-points provide a precise match for Hoare Logic [3].

The IC3 algorithm [4] was recently used successfully for hardware model checking [4, 6]. We use the current popular, and descriptive, terminology *Property Directed Reachability* (PDR) to refer to IC3 and its derivatives. PDR has several intriguing characteristics. It simultaneously strengthens an abstraction of reachable states and prunes a search for counter examples, but in contrast to predicate abstraction methods [2] and methods based on interpolants [12, 8, 1], it maintains precise transition relations and only refines state abstractions. Importantly, it leverages induction proofs to strengthen invariant candidates.

We are motivated by software analysis, where handling procedure calls and theories is relevant. In the pursuit of this goal, we contribute the following:

- provide an abstract account of the PDR algorithm;
- generalize PDR to *nonlinear* fixed-point operators;
- further generalize PDR to theories, specifically Linear Real Arithmetic.

The paper is organized as follows: Section 2 motivates satisfiability modulo least fixed-points. Sections 3 (4) present the abstract account of (nonlinear) PDR. Section 5 generalizes PDR to theories and Section 6 describes our implementation and experiments.

[1] The original paper appears in SAT 2012

2 From Safety Verification to Least Fixed-points

To motivate the use of fixed-point operators and solving satisfiability modulo least fixed-points consider Lamport's two process Bakery algorithm. It ensures mutual exclusion between processes P_1 and P_2. They cannot simultaneously execute **critical**.

initially $y_1 := y_2 := 0$;

$$P_1 :: \left[\begin{array}{l}\textbf{loop forever do}\\ \left[\begin{array}{l}\ell_0 : y_1 := y_2 + 1;\\ \ell_1 : \textbf{await } y_2 = 0 \vee y_1 \leq y_2;\\ \ell_2 : \textbf{critical};\\ \ell_3 : y_1 := 0;\end{array}\right]\end{array}\right] \;||\; P_2 :: \left[\begin{array}{l}\textbf{loop forever do}\\ \left[\begin{array}{l}\ell_0 : y_2 := y_1 + 1;\\ \ell_1 : \textbf{await } y_1 = 0 \vee y_2 \leq y_1;\\ \ell_2 : \textbf{critical};\\ \ell_3 : y_2 := 0;\end{array}\right]\end{array}\right]$$

Mutual exclusion and other safety properties are proved by *induction* over the set of reachable states. The induction proof requires finding an *inductive invariant* that is true in the initial state, is maintained by each step of the system, and implies the safety property. The induction G-INV rule can be formalized following [11]. Programs denote *transition systems* $\mathcal{S} = \langle \boldsymbol{x}, \Theta, \rho(\boldsymbol{x}, \boldsymbol{x}') \rangle$, where $\boldsymbol{x}$ is a set of state variables, Θ a formula describing a set of initial configurations over $\boldsymbol{x}$ and $\rho(\boldsymbol{x}, \boldsymbol{x}')$ is a transition relation. For the Bakery algorithm, $\Theta := y_1 = y_2 = L = M = 0$ where L is program counter for P_1 and M is a program counter for P_2. The safety property for Bakery is $S := \neg(L = 2 \wedge M = 2)$. The predicate R serves as the inductive invariant in G-INV. Notice that the premises of G-INV are Horn clauses. We reformulate the Bakery verification problem on the left using Prolog convention of capitalization for universally quantified variables. The non-recursive predicate T is a shorthand for ρ. It exploits the symmetry of P_1 and P_2. The real challenge is to find a solution to the recursive predicate R. A solution exists iff there is a solution to the least fixed-point of the *strongest post-condition* predicate transformer that defines R. For transition systems, the predicate transformer follows the template:

$$\frac{\begin{array}{l}\Theta(\boldsymbol{x}) \;\rightarrow\; R(\boldsymbol{x})\\ R(\boldsymbol{x}) \;\wedge \rho(\boldsymbol{x}, \boldsymbol{x}') \;\rightarrow R(\boldsymbol{x}')\\ R(\boldsymbol{x}) \;\rightarrow\; S(\boldsymbol{x})\end{array}}{\mathcal{S} \models \Box S(\boldsymbol{x})}\;\textsf{G-INV}$$

$$\begin{array}{r}R(0,0,0,0)\\ R(L,M,Y_1,Y_2) \wedge T(L,L',Y_1,Y_2,Y_1') \;\rightarrow\; R(L',M,Y_1',Y_2)\\ R(L,M,Y_1,Y_2) \wedge T(M,M',Y_2,Y_1,Y_2') \;\rightarrow\; R(L,M',Y_1,Y_2')\\ T(0,1,Y_1,Y_2,Y_2+1)\\ Y_1 \leq Y_2 \vee Y_2 = 0 \;\rightarrow\; T(1,2,Y_1,Y_2,Y_1)\\ T(2,3,Y_1,Y_2,Y_1)\\ T(3,0,Y_1,Y_2,0)\\ R(2,2,Y_1,Y_2) \;\rightarrow\; \mathit{false}\end{array}$$

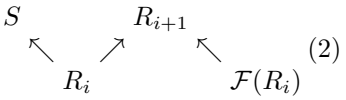
$$\mathcal{F}(R)(\boldsymbol{x}) := \exists \boldsymbol{x}_0 \;.\; \underbrace{\Theta(\boldsymbol{x}) \;\vee\; R(\boldsymbol{x}_0) \wedge \rho(\boldsymbol{x}_0, \boldsymbol{x})}_{\mathcal{T}[R(\boldsymbol{x}_0)]}, \tag{1}$$

where the quantifier-free body $\mathcal{T}$ of $\mathcal{F}$ is the *transition relation*, which now includes also the initial condition Θ. The predicate transformer (1) is *linear*; the argument R occurs in at most one (positive) position. Using the terminology of predicate transformers, finding an inductive invariant amounts to finding a *post*

fixed-point R, such that $\mathcal{F}(R) \to R$ and $R \to S$. Recall that the least fixed-point $\mu R.\mathcal{F}(R)$ (the infinite disjunction $\bigvee_{i \geq 0} \mathcal{F}^i(\mathit{false})$) is contained in any R that satisfies $\mathcal{F}(R) \to R$.

3 Abstract Property Directed Reachability

This section recalls the Property Directed Reachability algorithm using terminology of predicate transformers and we specify the algorithm as an abstract transition system. We build upon [6] as we arrive to a specification. The original IC3 algorithm verifies invariants of *linear* fixed-point operators. PDR maintains formulas $\Theta = R_0, \ldots, R_N$, such that for $0 \leq i < N$ invariant (2) holds. Initially we set $R_0 = \mathcal{F}(\mathit{false})$ and $N = 0$, so the invariant holds trivially. N is incremented if $R_N \to S$ is established.

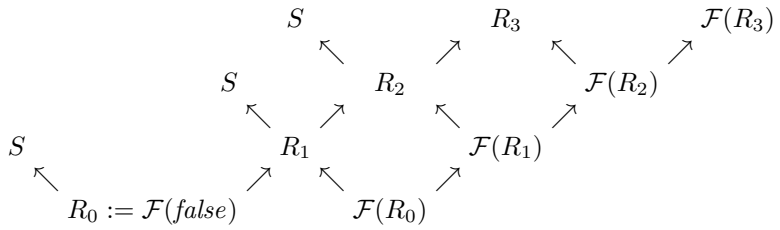


(2)

Fig. 1: **Invariant (2)**. Each arrow is an implication: $R_i \to S$, $R_i \to R_{i+1}$, $\mathcal{F}(R_i) \to R_{i+1}$.

For Bakery, we have $R_0 := \Theta := (L = M = Y_1 = Y_2 = 0)$ and $S := \neg(L = M = 2)$ so $R_0 \to S$, and we can increment $N := 1$, $R_1 := \mathit{true}$. The next action is to check if $R_1 \wedge \neg S$ is satisfiable. It is, and one *partial* model is denoted by $\mathcal{M} := L = 2 \wedge M = 2$ (it does not include assignments to Y_1, Y_2). The model $\mathcal{M}$ violates the safety property, but is $\mathcal{M}$ reachable from the current unfolding? They would be if $\mathcal{F}(R_0) \wedge \mathcal{M}$ was satisfiable. It is not satisfiable. One of several possible unsatisfiable cores is $L \neq 0 \wedge M \neq 0$. We then update R_1 with the negation: $R_1 := R_1 \wedge (L = 0 \vee M = 0)$ and now we can unfold again $N := 2$.

These steps give a taste of how PDR uses spurious counter examples, as partial models, to build up R_i. PDR also contains a clever mechanism for strengthening clauses in R_i by using *induction*, presented in the following. The R_i are sets of clauses. We shall however often (ab)use notation and use conjunction instead of set union and freely switch between viewing a set of formulas as a conjunction. The R_i denote sets of states (the set of models that satisfy R_i) and over-approximate the states reachable by unfolding the transition relation i times. All stages except the last imply the safety property. We can visualize the implications between the approximations using the picture below (for $N = 3$).



PDR relies on refining counter examples that are models. A model is a conjunction of equalities between variables and values. For example the model

$a = \mathit{true} \wedge b = \mathit{false} \wedge x = 3$ (more compactly written $a \wedge \neg b \wedge x = 3$), assigns a to *true*, b to *false* and x to 3. We use $\mathcal{M}$ as shorthand for models of the form $\boldsymbol{x} = \boldsymbol{c}$. A model $\mathcal{M}$ of a formula φ is allowed to be partial (omit assigning values to some variables) as long as φ is true under $\mathcal{M}$. When φ is a clause, then $\neg\varphi$ is treated as a conjunction of literals. So $\neg\varphi \subseteq \mathcal{M}$ means that all literals in φ are false in $\mathcal{M}$.

The following updates to R_i are made by the algorithm:

Valid For $i < N$, if $R_i \subseteq R_{i+1}$, then R_i is an inductive invariant. Return *Valid*.
Unfold If $R_N \to S$, then set $N \leftarrow N + 1, R_N \leftarrow \mathit{true}$.
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 conjoin φ to R_j, for each $j \leq i + 1$. While induction is a separate rule, it is useful to apply it immediately following Unfold and Conflict. The rule is sound because $\mathcal{F}$ is monotone and Invariant (2) ensures $R_j \to R_i$ for $j < i$. Therefore $\mathcal{F}(\mathit{false}) \to \varphi$ and $\mathcal{F}(R_j \wedge \varphi) \to \varphi$ for each $j < i$.

PDR includes a *dual* mode where it searches for a candidate counter-model to S. The candidate model is used to guide the strengthening of R_i.

Candidate If $\mathcal{M} \models R_N(\boldsymbol{x}) \wedge \neg S(\boldsymbol{x})$, then produce candidate $\langle \mathcal{M}, N \rangle$.
Decide If $\langle \boldsymbol{x} = \boldsymbol{c}, i + 1 \rangle$ for $0 \leq i < N$ is a candidate model and there is a subset $\tilde{\boldsymbol{x}}_{\boldsymbol{0}}$ of $\boldsymbol{x}_0$ and constants $\boldsymbol{c_0}$, such that $\boldsymbol{x} = \boldsymbol{c}, \tilde{\boldsymbol{x}}_{\boldsymbol{0}} = \boldsymbol{c_0} \models \mathcal{T}[R_i(\boldsymbol{x}_0)]$, then add the candidate model $\langle \tilde{\boldsymbol{x}} = \boldsymbol{c_0}, i \rangle$ (renaming $\tilde{\boldsymbol{x}}_{\boldsymbol{0}}$ to $\tilde{\boldsymbol{x}}$).
Model If $\langle \mathcal{M}, 0 \rangle$ is a candidate model, then report that S is violated.
Conflict For $0 \leq i < N$: given a candidate model $\langle \mathcal{M}, i+1 \rangle$ and clause φ, such that $\neg\varphi \subseteq \mathcal{M}$, if $\mathcal{F}(R_i) \to \varphi$, then conjoin φ to R_j, for $j \leq i + 1$.

3.1 PDR as an Abstract Transition System

Figure 2 summarizes the PDR algorithm as an abstract transition system. It maintains states of the form $M \parallel A$, where M is a candidate counter example trace that is a stack of models labeled by a level i, and A is the current abstraction comprising of the maximal level N and sets of clauses $R_0, \ldots, R_N$.

Example 1. Suppose we are given the safety property $S(x, y, z) \equiv \neg y$ and the predicate transformer $\mathcal{F} = \lambda R \lambda x y z \;.\; \exists x_0 y_0 z_0 \;.\; (x, y, z) = (1, 0, 0) \vee ((x, y, z) = (y_0, z_0, x_0) \wedge R(x_0, y_0, z_0)))$ that corresponds to the rules: $R(1, 0, 0)$. $R(x, y, z) \to R(y, z, x)$. We use 0 for *false* and 1 for *true*. We can check that $\neg S$ is reachable.

$$
\begin{array}{lcl}
\mathsf{Initialize} & \Longrightarrow & \epsilon \parallel [N \leftarrow 0, R_0 \leftarrow x \wedge \neg y \wedge \neg z] \\
\mathsf{Unfold} & \Longrightarrow & \epsilon \parallel [N \leftarrow 1, R_0, R_1 \leftarrow \mathit{true}] \\
\mathsf{Candidate} & \Longrightarrow & \langle y \wedge \neg z, 1 \rangle \parallel [N, R_0, R_1] \\
\mathsf{Conflict} & \Longrightarrow & \epsilon \parallel [N, R_0, R_1 \leftarrow \neg y] \;\; \text{since } y \wedge \neg z \models y, \;\; \mathcal{F}(R_0)(x, y, z) \to \neg y \\
\mathsf{Unfold} & \Longrightarrow & \epsilon \parallel [N \leftarrow 2, R_0, R_1, R_2 \leftarrow \mathit{true}] \\
\mathsf{Candidate} & \Longrightarrow & \langle y, 2 \rangle \parallel [N, R_0, R_1, R_2] \\
\mathsf{Decide} & \Longrightarrow & \langle z, 1 \rangle \langle y, 2 \rangle \parallel [N, R_0, R_1, R_2] \\
\mathsf{Decide} & \Longrightarrow & \langle x, 0 \rangle \langle z, 1 \rangle \langle y, 2 \rangle \parallel [N, R_0, R_1, R_2] \\
\mathsf{Model} & \Longrightarrow & \langle x, 0 \rangle \langle z, 1 \rangle \langle y, 2 \rangle \qquad \boxtimes
\end{array}
$$

$$\textsf{Initialize} \implies \epsilon \parallel [N \leftarrow 0, R_0 \leftarrow \mathcal{F}(\mathit{false})]$$

$$\textsf{Valid} \quad M \parallel A \implies \textsf{Valid} \qquad \textbf{if } \models R_{i-1} \subseteq R_i,\ i < N.$$

$$\textsf{Unfold} \quad M \parallel A \implies \epsilon \parallel A[R_{N+1} \leftarrow \mathit{true}, N \leftarrow N+1] \qquad \textbf{if } \models R_N \to S,$$

$$\textsf{Induction} \quad M \parallel A \implies M \parallel A[R_j \leftarrow R_j \wedge \varphi]_{j=1}^{i+1} \qquad \textbf{if } (\varphi \vee \psi) \in R_i,\ \varphi \notin R_{i+1},\ \models \mathcal{F}(R_i \wedge \varphi) \to \varphi$$

$$\textsf{Candidate} \quad \epsilon \parallel A \implies \langle \mathcal{M}, N \rangle \parallel A \qquad \textbf{if } \mathcal{M} \models R_N \wedge \neg S$$

$$\textsf{Decide} \quad \langle \mathcal{M}, i{+}1 \rangle M \parallel A \implies \langle \tilde{\boldsymbol{x}} = \boldsymbol{c_0}, i \rangle \langle \mathcal{M}, i{+}1 \rangle M \parallel A \qquad \textbf{if } \mathcal{M}, \tilde{\boldsymbol{x}}_{\boldsymbol{0}} = \boldsymbol{c_0} \models \mathcal{T}[R_i(\boldsymbol{x_0})]$$

$$\textsf{Model} \quad \langle \mathcal{M}, 0 \rangle M \parallel A \implies \textsf{Model } \langle \mathcal{M}, 0 \rangle M$$

$$\textsf{Conflict} \quad \langle \mathcal{M}, i{+}1 \rangle M \parallel A \Longrightarrow M \parallel A[R_j \leftarrow R_j \wedge \varphi]_{j=1}^{i+1} \qquad \textbf{if } \neg\varphi \subseteq \mathcal{M},\ \models \mathcal{F}(R_i) \to \varphi.$$

Fig. 2: Abstract transition system specification of PDR

The example exercises several features of PDR. It starts with a Candidate counter example model to the last state and Decide pushes models down to the initial state. There is some freedom in choosing models to push down. Such models can be partial; they just need to force the transition relations. If models can be pushed all the way down, there is a counter example trace, otherwise a Conflict gets detected along the way. The induction rule is specified as a separate rule, but it can be applied immediately after Conflict to minimize the new clause. A good analogy is how subsumption is used when processing conflict clauses in modern SAT solvers. Induction also serves the purpose of pushing up clauses from $(\varphi \vee \psi) \in R_i$ to R_{i+1} by taking $\psi = \mathit{false}$. Such propagation can be applied immediately after Conflict and before Unfold.

Correctness of the algorithm follows from four observations:

Lemma 1 (Invariant (2)). *The rules from Figure 2 maintain Invariant (2).*

Lemma 2 (Validity). *When $\models R_i \subseteq R_{i+1}$, then S is invariant.*

Proof. Let us add this condition to the implications from invariant (2) and we get that R_i is a post-fixed point that is contained in S: $\mathcal{F}(R_i) \to R_{i+1} \to R_i \to S$. Thus, R_i satisfies the premises of G-INV and therefore S is invariant.

Lemma 3 (Satisfiability). *When $\langle \mathcal{M}, 0 \rangle$ is reached, then S is violated with a path of length N.*

Corollary 1 (Correctness of PDR). *If PDR terminates with* Valid*, then S is invariant. If PDR terminates with* Model *M, then M is a trace leading to a violation of S.*

It is also the case that each step makes progress by either extending models or strengthening states. The set of possible different states R_i is bounded by the set of possible models (assuming that clauses are normalized) so the algorithm terminates for finite domains. Therefore,

Theorem 1 (Termination on Finite Domains). *Any derivation sequence terminates with a verdict* Valid *or* Model *when $\mathcal{F}$ is finite domain.*

Note that PDR represents traces explicitly, so while reachability of Boolean systems is PSPACE, PDR may nevertheless consume exponential space.

4 Nonlinear PDR

Nonlinear transformers are important in the context of checking software with procedures. The Static Driver Verifier [2] implements a model checker for programs with procedure calls.

$\texttt{mc(x)} = \texttt{if x} > \texttt{100 then x} - \texttt{10 else mc(mc(x} + \texttt{11))}$
$\texttt{assert } \forall \texttt{x.mc(x)} \geq 91$

$X > 100 \;\rightarrow\; mc(X, X - 10)$
$X \leq 100 \wedge mc(X + 11, Y) \wedge mc(Y, R) \;\rightarrow\; mc(X, R)$
$mc(X, R) \;\rightarrow\; R \geq 91$

Nonlinear predicate transformers correspond to general Horn clauses. An example program with procedure calls and resulting non-linear Horn clauses comes from McCarthy's 91 function and the accompanying assertion.

We therefore consider nonlinear predicate transformers of the form

$$\mathcal{F}(R)(\boldsymbol{x}) = \exists \boldsymbol{x_0}, \boldsymbol{x_1} \,.\, \underbrace{\Theta(\boldsymbol{x}) \;\vee\; [R(\boldsymbol{x_0}) \;\wedge\; R(\boldsymbol{x_1}) \;\wedge\; \rho(\boldsymbol{x_0}, \boldsymbol{x_1}, \boldsymbol{x})]}_{\mathcal{T}[R(\boldsymbol{x}_0), R(\boldsymbol{x}_1)]} \tag{3}$$

We use the template (3) when presenting algorithms for nonlinear PDR. The terminology of predicate transformers was useful for formulating the main invariant (2), and we find it particularly instrumental for generalizing PDR to general Horn clauses. The extension to nonlinear predicate transformers with more than two occurrences of R, and systems of nonlinear predicate transformers is relatively straight-forward.

4.1 State

In contrast to linear predicate transformers, counter examples for nonlinear transformers unfold into trees. A compressed view of counter examples is as DAGs, and the potential savings of using DAGs can be exponential. A challenge is to create and maintain such counter examples. We propose an approach where states that are known to be reachable are put in a cache, and PDR inserts nodes into a DAG. So it inspects the current DAG to see if a new (potentially) reachable state is already being expanded before creating a new node. States are compared syntactically. A more powerful alternative is to represent the cache as a formula and check cache containment semantically, but we found no practical

use for such added power: counter examples for recursive predicates from programs can be expected to have a small tree unfolding. We present this approach in the following.

The state of the algorithm is maintained as a triple $\mathcal{D} \parallel A \parallel C$, where:

$\mathcal{D}$, the model search DAG represents a partial unfolding of a counter example. It is initially the empty DAG ϵ. Nodes are labeled with queries $\langle \mathcal{M}, i \rangle$, where i is a level and $\mathcal{M}$ is a partial model. We use L as a shorthand for $\langle \mathcal{M}, i \rangle$; use $\mathcal{D}[L \bullet \{\mathcal{D}'\ \mathcal{D}''\}]$ to refer to an internal node L with two children; and $\mathsf{model}(\mathcal{D})$ to access the model at the root of a DAG.

A, the property state of the form $[N, R_0, \ldots, R_N]$.

C, the cache of reachable states. It contains a set of partial interpretations $\mathcal{M}$ that imply $\mathcal{F}^n(\mathit{false})$ for some $n \geq 0$. Consequently, every completion of $\mathcal{M}$ is contained in the least fixed-point and is therefore reachable.

4.2 Algorithm Specification

Figure 3 contains the new rules we need for the nonlinear variant of PDR. Rules Initialize, Valid, Induction, Unfold, Candidate are unchanged from Figure 2, with the exception that we add a column for the cache C and we replace the stack of models M by a DAG $\mathcal{D}$.

$$\textsf{Decide} \quad \mathcal{D}[\langle \mathcal{M}, i+1 \rangle] \parallel A \parallel C \implies \mathcal{D}[\langle \mathcal{M}, i+1 \rangle \bullet \{\langle \tilde{\boldsymbol{x}} = \boldsymbol{c_0}, i \rangle\ \langle \tilde{\tilde{\boldsymbol{x}}} = \boldsymbol{c_1}, i \rangle\}] \parallel A \parallel C \quad \textbf{if } \mathcal{M}, \tilde{\boldsymbol{x}}_{\boldsymbol{0}} = \boldsymbol{c_0}, \tilde{\tilde{\boldsymbol{x}}}_{\boldsymbol{1}} = \boldsymbol{c_1} \models \mathcal{T}[R_i(\boldsymbol{x_0}), R_i(\boldsymbol{x_1})]$$

$$\textsf{Model} \quad \mathcal{D} \parallel A \parallel C \implies \mathsf{Model}\ \mathcal{D} \quad \textbf{if } \langle \mathcal{M}, N \rangle \in \mathcal{D}, \mathcal{M} \in C$$

$$\textsf{Conflict} \quad \mathcal{D}[L \bullet \{\mathcal{D}'\ \mathcal{D}''\}] \parallel A \parallel C \implies \mathcal{D}[L] \parallel A[R_j \leftarrow R_j \wedge \varphi]_{j=1}^{i+1} \parallel C \quad \textbf{if } \neg\varphi \subseteq \mathsf{model}(\mathcal{D}'), \models \mathcal{F}(R_i) \rightarrow \varphi.$$

$$\textsf{Base} \quad \mathcal{D}[\langle \mathcal{M}, i \rangle] \parallel A \parallel C \implies \mathcal{D} \parallel A \parallel C \cup \{\mathcal{M}\} \quad \textbf{if } \mathcal{M} \models R_0.$$

$$\textsf{Close} \quad \mathcal{D}[\langle \mathcal{M}, i+1 \rangle \bullet \{\mathcal{D}'\ \mathcal{D}''\}] \parallel A \parallel C \implies \mathcal{D} \parallel A \parallel C \cup \{\mathcal{M}\} \quad \textbf{if } \mathsf{model}(\mathcal{D}'), \mathsf{model}(\mathcal{D}'') \in C\ .$$

Fig. 3: Abstract nonlinear transitions

Decide extends a leaf L in $\mathcal{D}$ with two children. The nodes correspond to partial models for the variables that are arguments to the recursive predicates in $\mathcal{F}$. To differentiate two possibly different subsets of $\boldsymbol{x}$ we use $\tilde{\boldsymbol{x}}$ and $\tilde{\tilde{\boldsymbol{x}}}$. The children are possibly pointers to nodes that already exist in $\mathcal{D}$ (so that we don't expand the same model twice). Model declares a counter example when all the

leaves and internal nodes have been validated. This amounts to that the root of $\mathcal{D}$ is in the cache C. Conflicts are similar, Conflict backtracks from a leaf when the (partial) model annotating the leaf contradicts the constraints at its level.

There are two new rules. The rules are Base and Close. Their role is to propagate cache hits upwards in the model DAG. At the base level, a model $\mathcal{M}$ is added to the cache C if it implies R_0. The Close rule removes children from an internal node if each child is reachable. The model annotating the internal node is then also reachable, so added to C.

Correctness follows analogously to the basic PDR algorithm, as we maintain the following properties for a state $\mathcal{D} \parallel A \parallel C$:

1. $R_0 \equiv \mathcal{F}(\mathit{false})$.
2. Invariant (2) holds.
3. Every member $\mathcal{M} \in C$ is contained in $\mathcal{F}^N(\mathit{false})$.
4. Every internal node $\langle \mathcal{M}, i+1\rangle$ with children $\langle \tilde{\boldsymbol{x}} = \boldsymbol{c_0}, i\rangle, \langle \tilde{\tilde{\boldsymbol{x}}} = \boldsymbol{c_1}, i\rangle$, it is the case that $\mathcal{M}, \tilde{\boldsymbol{x}}_{\boldsymbol{0}} = \boldsymbol{c_0}, \tilde{\tilde{\boldsymbol{x}}}_{\boldsymbol{1}} = \boldsymbol{c_1} \models \mathcal{T}[R_i(\boldsymbol{x_0}), R_i(\boldsymbol{x_1})]$.

Example 2. Consider a nonlinear system $R(\mathit{true}, \mathit{true})$. $R(x_0, y_0) \wedge R(x_1, y_1) \rightarrow R(x_0 \oplus x_1, y_0 \oplus y_1)$. $R(\mathit{true}, \mathit{false}) \rightarrow \mathit{false}$. A sample run of the algorithm proceeds as follows:

Initialize	$\Longrightarrow$	$\epsilon \parallel A_0 \parallel \{\}$	for $A_0 = [N \leftarrow 0, R_0 \leftarrow x \wedge y]$
Unfold	$\Longrightarrow$	$\epsilon \parallel A_1 \parallel \{\}$	for $A_1 = A_0[N \leftarrow 1, R_1 \leftarrow \mathit{true}]$
Candidate	$\Longrightarrow$	$\langle x \wedge \neg y, 1\rangle \parallel A_1 \parallel \{\}$	
Conflict	$\Longrightarrow$	$\epsilon \parallel A_2 \parallel \{\}$	for $A_2 = A_1[R_1 \leftarrow R_1 \wedge (\neg x \vee y)]$
Unfold	$\Longrightarrow$	$\epsilon \parallel A_3 \parallel \{\}$	for $A_3 = A_2[N \leftarrow 2, R_2 \leftarrow \mathit{true}]$
Candidate	$\Longrightarrow$	$\langle x \wedge \neg y, 2\rangle \parallel A_3 \parallel \{\}$	
Decide	$\Longrightarrow$	$\langle x \wedge \neg y, 2\rangle \bullet \{\langle x \wedge y, 1\rangle \ \langle \neg x \wedge y, 1\rangle\} \parallel A_3 \parallel \{\}$	
Base	$\Longrightarrow$	$\langle x \wedge \neg y, 2\rangle \bullet \{\langle x \wedge y, 1\rangle \ \langle \neg x \wedge y, 1\rangle\} \parallel A_3 \parallel \{x \wedge y\}$	
Conflict	$\Longrightarrow$	$\langle x \wedge \neg y, 2\rangle \parallel A_4 \parallel \{x \wedge y\}$	for $A_4 = A_3[R_j \leftarrow R_j \wedge (x \vee \neg y)]_{j=1}^{2}$
Induction	$\Longrightarrow$	$\ldots \parallel A_4[R_j \leftarrow R_j \wedge (\neg x \vee y)]_{j=1}^{2} \parallel \{x \wedge y\}$	
Valid	$\Longrightarrow$	Valid	

Note how Decide develops two branches. When one child is in conflict then both children are collapsed. Note also how Induction is used to push $(\neg x \vee y)$ up to level 2. The property is inductive when combined with the property $(x \vee \neg y)$. At this point $R_2 \rightarrow R_1$ (e.g., $R_1 \subseteq R_2$) so the procedure terminates with Valid. $\boxtimes$

5 Theories - The case of Linear Real Arithmetic

We generalize PDR to handle non-Boolean constraints. The problem goes from PSPACE to highly intractable. Nevertheless, we identify a subclass, timed push-down systems, that are handled by our generalization. Our approach is to lift the Conflict and Decide rules and instantiate the generalization to the theory of

linear real arithmetic. Central to our approach is the use of models for guiding the creation of conflict clauses as interpolants. The interpolants are a minimal set of constraints implied by the existing abstraction that suffice to exclude a spurious counter example. When iterated over all spurious counter examples, our procedure does in fact produce interpolants for systems of non-recursive Horn clauses [8]. Our incremental approach is appealing compared to an approach that computes interpolants in from an eager unfolding: intermediary results from spurious counter examples act as conflict clauses for future traversals. We use the calculus from Section 3 to keep definitions simpler.

5.1 Conflicts

Recall Conflict applies when there is a $\varphi \subseteq \neg\mathcal{M}$ such that $\mathcal{F}(R_i) \rightarrow \varphi$. The Conflict rule therefore applies when $\mathcal{F}(R_i) \rightarrow \neg\mathcal{M}$. The propositional version lets us add any subset of $\neg\mathcal{M}$ that is implied by $\mathcal{F}(R_i)$. The clause φ is also an interpolant by construction. A problem with using a subset of $\neg\mathcal{M}$ for infinite domains is that the number of potential counter-models is unbounded, so blocking one of an unbounded set of models does not help to ensure convergence. In principle one can take any clause *Post* such that

$$\mathcal{F}(R_i) \;\rightarrow\; \mathit{Post}, \qquad \mathit{Post} \;\rightarrow\; \neg\mathcal{M}\ . \tag{4}$$

This suggests a G-Conflict rule (formulated for linear fixed-points) as:

$$\textsf{G-Conflict}\quad \langle \mathcal{M}, i+1\rangle M \parallel A \implies M \parallel A[R_j \leftarrow R_j \wedge \mathit{Post}]_{j=1}^{i+1} \quad \textbf{if } \models \mathcal{F}(R_i) \rightarrow \mathit{Post},\quad \mathit{Post} \rightarrow \neg\mathcal{M}.$$

Where *Post* is any clause that uses the variables $\boldsymbol{x}$ and implies $\neg\mathcal{M}$. Notice that we require *Post* to be a single clause. At the other extreme, one could think of taking $\mathit{Post} := \mathcal{F}(R_i)$, the strongest post-condition that is independent of $\mathcal{M}$. The resulting algorithm would have to rely on quantifier elimination to convert the result into a set of clauses and for making effective use of Induction. The rule G-Conflict without further conditions is not informative.

Arithmetical Conflicts We instantiate G-Conflict for the theory of Linear Real Arithmetic (LRA) and show that we obtain a decision procedure for safety properties of timed push-down systems. The main idea is to compute the *strongest* conflict clause modulo linear real arithmetic from unsatisfiability of $\mathcal{M} \wedge \mathcal{F}(R_i)$. The conflict clause is by construction an interpolant and the way it is extracted can be described as a specialized interpolation procedure. On the right is the stage $N = 4$ where PDR pushes a counter example down for Bakery. It

$$\begin{array}{c} L = 2 \wedge M = 2 \models \mathcal{F}(R_3) \wedge \neg S \\ \uparrow \\ L = 1 \wedge M = 2 \wedge Y_2 = 0 \models \mathcal{F}(R_2) \\ \uparrow \\ L = 1 \wedge M = 1 \wedge Y_1 = 1 \wedge Y_2 = 0 \models \mathcal{F}(R_1) \\ \vdots \\ L = 0 \wedge M = 1 \wedge Y_2 = 0 \models \neg\mathcal{F}(R_0) \end{array}$$

reaches a conflict because $L = 0 \wedge M = 1 \wedge Y_2 = 0 \wedge \mathcal{T}[R_0(\boldsymbol{x_0})]$ is unsatisfiable. The justification includes the clause $\neg(Y_2 \leq 0 \wedge Y_1 \geq 0 \wedge Y_2 \geq Y_1 + 1)$. The last two literals are from $\mathcal{T}[R_0(\boldsymbol{x_0})]$. They resolve to $Y_2 > 0$, justifying the stronger conflict clause $\neg(L = 0 \wedge M = 1 \wedge Y_2 \leq 0)$.

In general, assume $\mathcal{M}$ is of the form $\bigwedge_i k_i \leq x_i \leq k_i$ where x_i are variables and k_i are numerals of type Real. The G-Conflict rule applies when $\mathcal{M} \wedge \mathcal{T}[R_i(\boldsymbol{x_0})]$ is unsatisfiable and there is a resolution proof Π that derives the empty clause. In the following we make two important assumptions for our construction, first we assume that all literals in Π are already in $\mathcal{M} \wedge \mathcal{T}[R_i(\boldsymbol{x_0})]$. This is the case for proofs produced by the DPLL(T) framework [13]. Second, we assume that all literals in $\mathcal{M}$ are used in unit-resolution with input clauses. This can be enforced by permuting $\mathcal{M}$ up in proofs. The leaves of Π comprise of the inequalities (unit-literals) from $\mathcal{M}$, clauses from $\mathcal{T}[R_i(\boldsymbol{x_0})]$ and T-axioms. In the theory of LRA, the T-axioms are of the form $\bigvee_i \neg(A_i \boldsymbol{x} - \boldsymbol{b}_i \leq 0)$, where A_i are row vectors and $\boldsymbol{b}_i$ are constants. Recall that we can represent strict inequalities $t > s$ using non-strict inequalities by using an infinitesimal ϵ constant for $t \geq s + \epsilon$. Let us write $A\boldsymbol{x} \leq \boldsymbol{b}$ for the conjunction $\bigwedge_i A_i \boldsymbol{x} \leq \boldsymbol{b}_i$. Farkas' lemma implies that there is a corresponding set of non-negative coefficients $\boldsymbol{\lambda}$, such that $\boldsymbol{\lambda} \cdot A \cdot \boldsymbol{x}$ is a numeric constant and $\boldsymbol{\lambda} \cdot A \cdot \boldsymbol{x} > \boldsymbol{\lambda b}$. These coefficients are produced as a side-effect of the Simplex procedure. Proof-objects exposed by Z3 [9] include the coefficients.

The method for creating *Post* is now as follows: conjoin every literal from $\mathcal{M}$ that resolves against a clause from $\mathcal{T}[R_i(\boldsymbol{x_0})]$ in Π. Furthermore, for every T-axiom we partition the literals into two groups, the first group contains the literals that resolve against a literal from $\mathcal{M}$, the second comprises of literals that resolve against clauses from $\mathcal{T}[R_i(\boldsymbol{x_0})]$. Rewrite the inequality as $\begin{bmatrix} C \\ D \end{bmatrix} x \leq \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{d} \end{bmatrix}$, where the inequalities with coefficients $C, \boldsymbol{c}$ resolve against $\mathcal{M}$ and the remaining inequalities resolve against $\mathcal{T}[R_i(\boldsymbol{x_0})]$. The coefficients from Farkas' lemma are $\boldsymbol{\lambda}_C$ and $\boldsymbol{\lambda}_D$ respectively, such that:

$$\boldsymbol{\lambda}_C C\boldsymbol{x} + \boldsymbol{\lambda}_D D\boldsymbol{x} > \boldsymbol{\lambda}_C \boldsymbol{c} + \boldsymbol{\lambda}_D \boldsymbol{d}, \tag{5}$$

and therefore:

$$D\boldsymbol{x} \leq \boldsymbol{d} \;\rightarrow\; \boldsymbol{\lambda}_D D\boldsymbol{x} \leq \boldsymbol{\lambda}_D \boldsymbol{d}, \quad \boldsymbol{\lambda}_D D\boldsymbol{x} \leq \boldsymbol{\lambda}_D \boldsymbol{d} \;\rightarrow\; \boldsymbol{\lambda}_C C\boldsymbol{x} > \boldsymbol{\lambda}_C \boldsymbol{c} \;. \tag{6}$$

Then replace the theory axiom in Π by

$$\neg(D\boldsymbol{x} \leq \boldsymbol{d}) \;\vee\; \boldsymbol{\lambda}_D D\boldsymbol{x} \leq \boldsymbol{\lambda}_D \boldsymbol{d} \tag{7}$$

and conjoin $\boldsymbol{\lambda}_D D\boldsymbol{x} > \boldsymbol{\lambda}_D \boldsymbol{d}$ to *Post*. This literal is implied by the original literals $C\boldsymbol{x} > \boldsymbol{c}$ from $\mathcal{M}$. Denote by Farkas-Conflict the rule that extracts formula *Post* corresponding to a weakening of $\mathcal{M}$ determined by Π'.

Notice that the new literals that are produced by applying Farkas lemma are linear combinations of literals that occur in $\mathcal{F}(R_i)$. The number of linearly independent linear combinations of such literals is naturally bounded by the dimension of the vector space, so the Farkas conflict rule can only be applied a finite number of times for each unfolding N.

5.2 Timed push-down systems

Basic timed transition systems are of the form $\mathcal{S} = \langle \boldsymbol{x}, \mathcal{C}, \Theta, \rho \ \vee \ \rho_{tick} \rangle$, where $\boldsymbol{c} \subseteq \boldsymbol{x}$ is a designated set of clock variables, and $\boldsymbol{d} := \boldsymbol{x} \setminus \boldsymbol{c}$ are finite domain data-variables. There is a transition $\rho_{tick} : \exists \delta . \boldsymbol{c'} = \boldsymbol{c} + \delta \wedge \boldsymbol{d'} = \boldsymbol{d}$ that advances time on the clock variables. Other transitions are allowed to reset clocks to 0 and modify the data-variables. We consider a slight extension of timed transition systems with push-down capabilities. Reachable states can be described as:

$$R(\boldsymbol{c}, \boldsymbol{d}) \wedge \boldsymbol{c'} = \boldsymbol{c} + \delta \wedge \varphi(\boldsymbol{c'}, \boldsymbol{d}) \rightarrow R(\boldsymbol{c'}, \boldsymbol{d}) \tag{8}$$

$$R(\boldsymbol{c}, \boldsymbol{d}) \wedge \wedge_i \boldsymbol{c}'_i = \mathit{reset}?(\boldsymbol{c}_i) \rightarrow R(\boldsymbol{c'}, \boldsymbol{d}) \tag{9}$$

$$R(\boldsymbol{c}, \boldsymbol{y}) \wedge R(\boldsymbol{c}, \boldsymbol{z}) \wedge \varphi(\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{y}, \boldsymbol{z}) \rightarrow R(\boldsymbol{c}, \boldsymbol{d}) \tag{10}$$

where $\mathit{reset}?(c)$ is either c or 0 and the occurrences of clocks in φ is restricted to difference arithmetic formulas of the form $\boldsymbol{c}_i - \boldsymbol{c}_j \leq k$ for k a constant.

Theorem 2 (Timed Push-down System Reachability). *Generalized PDR with* Farkas-Conflict *decides timed push-down system reachability.*

Proof (Idea). Use the observation that Farkas-Conflict produces only literals in the transitive closure of the difference constraints from the timed push-down system. Assume $\mathcal{F}$ is a description of a timed push-down system that uses the difference constraints $\Delta = \{y_{i_1} - y_{j_1} \leq k_1, y_{i_2} \leq k_2, \ldots\}$ where each y_i is from $\boldsymbol{x_0}$. Add to Δ also the inequalities $y_i \geq 0, y_i \leq 0$ for each y_i from $\boldsymbol{x_0}$. As usual in difference arithmetic we can treat Δ as a directed graph whose edges are weighted by the difference constraints. Let Δ^* be the transitive closure that contains inequalities for every loop-free path in Δ. Suppose that $\boldsymbol{x} = \boldsymbol{c} \wedge \mathcal{T}[R_i(\boldsymbol{x}_0)]$ is unsatisfiable (the premise of G-Conflict) with proof Π and let $C : \bigvee_i \neg(A_i \boldsymbol{x} - \boldsymbol{b}_i \leq 0)$ be a clause in Π that is justified by Farkas lemma. Consider the most interesting case where C contains at most two literals $x_i \geq k_i, x_j \leq k_j$ from $\mathcal{M}[\boldsymbol{x}]$ and C contains the atoms $x_i = y_i + \delta, x_j = y_j + \delta$ (or $x_i = y_i, x_j = y_j$) together with literals from Δ^*. Since difference logic tautologies correspond to paths in a weighted graph, the literal $\boldsymbol{\lambda}_D D\boldsymbol{x} > \boldsymbol{\lambda}_D \boldsymbol{d}$ obtained from Farkas' lemma cancel out the coefficient δ and use a weight that corresponds to a directed path in Δ^*. In each case, every spurious counter example was blocked by a combination of literals in Δ^*. Since Δ^* is finite this process terminates.

The Farkas-Conflict rule also suffices for some non-timed transition systems. It can prove the mutual exclusion property of our initial Bakery algorithm example.

5.3 Decisions

Farkas-Conflict is useful for many scenarios, but it is easy to come up with Horn clauses where it is insufficient. For example, the inductive invariant $2x = y$ that is required to establish the satisfiability of the Horn clauses cannot be found using Farkas-Conflict.

$$R(x, y) \ \rightarrow \ R(x + 1, y + 2). \quad R(0, 0). \quad R(x, y) \wedge 2x \neq y \ \rightarrow \ \mathit{false}. \tag{11}$$

A remedy to this limitation is to generalize the Decide rule. The approach is motivated as a way of producing relevant predicates, similar to what predicate abstraction achieves. We cannot help to note some dualities between the Decide and the Conflict rules: Conflict strengthens invariants and uses over-approximations of strongest post-conditions; Decide weakens counter examples and uses under-approximations of pre-conditions. Recall the basic Decide rule:

$$\textsf{Decide}\ \ \langle \mathcal{M}, i{+}1\rangle M \parallel A \implies \langle \tilde{\boldsymbol{x}} = \boldsymbol{c_0}, i\rangle \langle \mathcal{M}, i{+}1\rangle M \parallel A \qquad \textbf{if } \mathcal{M}, \tilde{\boldsymbol{x}}_{\boldsymbol{0}} = \boldsymbol{c_0} \models \mathcal{T}[R_i(\boldsymbol{x_0})]$$

In order to retain predicates that are relevant to the counter example trace, we can use any pre-condition Pre such that

$$\tilde{\boldsymbol{x}}_{\boldsymbol{0}} = \boldsymbol{c_0} \ \rightarrow\ Pre[\tilde{\boldsymbol{x}}_{\boldsymbol{0}}], \qquad Pre[\tilde{\boldsymbol{x}}_{\boldsymbol{0}}] \ \rightarrow\ \exists \boldsymbol{x}\ .\ \mathcal{M}[\boldsymbol{x}] \wedge \mathcal{T}[R_i(\boldsymbol{x_0})]\ . \tag{12}$$

Thus, the generalized Decide rule is:

$$\begin{aligned}\textsf{G-Decide}\ \ \langle \mathcal{M}, i+1\rangle M \parallel A \implies\ & \langle \tilde{\boldsymbol{x}} = \boldsymbol{c_0} \wedge Pre[\tilde{\boldsymbol{x}}], i\rangle \langle \mathcal{M}, i+1\rangle M \parallel A \\ & \textbf{if } \tilde{\boldsymbol{x}}_{\boldsymbol{0}} = \boldsymbol{c_0} \ \rightarrow\ Pre[\tilde{\boldsymbol{x}}_{\boldsymbol{0}}], \quad Pre[\tilde{\boldsymbol{x}}_{\boldsymbol{0}}] \ \rightarrow \exists \boldsymbol{x}\ .\ \mathcal{M}[\boldsymbol{x}] \wedge \mathcal{T}[R_i(\boldsymbol{x_0})]\end{aligned}$$

A crucial insight in [6] is to use ternary simulation for computing the relevant subset $\tilde{\boldsymbol{x}}_{\boldsymbol{0}}$ of $\boldsymbol{x}$. This reduces the set of literals in $\tilde{\boldsymbol{x}}_{\boldsymbol{0}} = \boldsymbol{c_0}$. We are not aware of a canonical approach to lifting model generalization to the first-order case. The following is a heuristic. For the first-order case we also leverage ternary simulation to minimize $\tilde{\boldsymbol{x}}_{\boldsymbol{0}} = \boldsymbol{c_0}$, and select the literals in $\mathcal{T}[R_i(\boldsymbol{x_0})]$ that contribute to making the formula true under $\tilde{\boldsymbol{x}}_{\boldsymbol{0}} = \boldsymbol{c_0}, \mathcal{M}[\boldsymbol{x}]$. The goal is to produce a conjunction $\tilde{\boldsymbol{x}}_{\boldsymbol{0}} = \boldsymbol{c_0} \wedge Pre[\tilde{\boldsymbol{x}}_{\boldsymbol{0}}]$ comprising of an assignment to $\tilde{\boldsymbol{x}}_{\boldsymbol{0}}$ and auxiliary literals over $\tilde{\boldsymbol{x}}_{\boldsymbol{0}}$ such that $\tilde{\boldsymbol{x}}_{\boldsymbol{0}} = \boldsymbol{c_0} \models Pre[\tilde{\boldsymbol{x}}_{\boldsymbol{0}}]$. So by induction assume $\mathcal{M}[\boldsymbol{x}]$ is also of this form: $\mathcal{M}^1 \wedge Pre^1$. When $\mathcal{F}$ is derived from guarded assignments, the variables $\boldsymbol{x}$ are typically given as a function of previous state variables, and the selected literals from $\mathcal{T}[R_i(\boldsymbol{x_0})]$ contains equalities of the form $x = t[\boldsymbol{x_0}]$. We collect these equalities as a substitution θ. The condition for $Pre[\tilde{\boldsymbol{x}}_{\boldsymbol{0}}]$ is then reduced to:

$$\tilde{\boldsymbol{x}}_{\boldsymbol{0}} = \boldsymbol{c_0} \ \rightarrow\ Pre[\tilde{\boldsymbol{x}}_{\boldsymbol{0}}], \quad Pre[\tilde{\boldsymbol{x}}_{\boldsymbol{0}}] \ \rightarrow \exists \boldsymbol{x}\ .\ \mathcal{M}^1 \wedge (Pre^1 \wedge \mathcal{T}[R_i(\boldsymbol{x_0})])\theta \tag{13}$$

Our current approach creates $Pre[\tilde{\boldsymbol{x}}_{\boldsymbol{0}}]$ as the conjunction of $\tilde{\boldsymbol{x}}_{\boldsymbol{0}} = \boldsymbol{c_0}$ and the selected literals from $(Pre^1 \wedge \mathcal{T}[R_i(\boldsymbol{x_0})])\theta$ that do not contain variables from $\boldsymbol{x}$ and that do not mix variables from different predecessor states.

Example 3. Assume a candidate counter example to (11) sets $x' = 3, y' = 1$. Then, $\exists x', y'\ .x' = 3 \wedge y' = 4 \wedge [(y' = y + 2 \wedge x' = x + 1 \wedge 2x' \neq y') \vee y' = x' = 0]$ yields the pre-condition $x = 2 \wedge y = 2 \wedge 2x \neq y$. ⊠

It is now also necessary to generalize Conflict so that it can produce the necessary conflict clauses from either $\mathcal{M}$ or the predicates from the weakest pre-condition.

Multi-Core Conflicts Each unsatisfiable core for $\mathcal{F}(R_i) \wedge \mathcal{M}$ gives rise to a different conflict that can enable a different proof. A proper generalization of G-Conflict is therefore to allow multiple conflicts

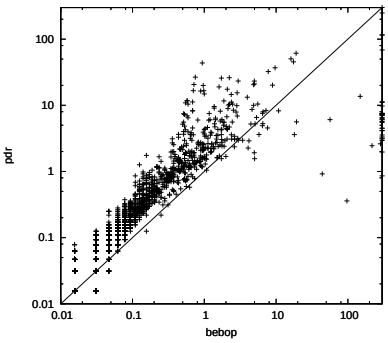
$$\text{MC-Conflict} \quad \langle \mathcal{M}, i+1 \rangle M \parallel A \implies M \parallel A[R_{i+1} \leftarrow R_{i+1} \wedge Post_1 \wedge \ldots \wedge Post_k] \\ \textbf{if } \models \mathcal{F}(R_i) \rightarrow Post_j, \quad Post_j \rightarrow \neg \mathcal{M}, \text{ for } j = 1..k.$$

Algorithms for unsatisfiable cores [10] and efficient integration of cores in PDR is beyond the scope of this paper.

6 Implementation and Experiments

We have implemented Generalized PDR in μZ [9] and we have performed a number of experiments to validate the generalizations to nonlinear PDR and linear real arithmetic. Additional material is online http://rise4fun.com/z3py/courses/fixedpoints.

We tested our implementation on a set of 2906 Boolean programs that come with the Windows Driver Research Platform.[2] Most programs are checked for safety violations within a second by both the Bebop tool and μZ. We wrote a basic converter from Boolean programs into Horn clauses. It associates a recursive predicate with each program statement and therefore sometimes requires much more space than the Boolean program. We are therefore not surprised that our prototype is generally 3 times slower than Bebop. Nevertheless it prevails where it matters: *it was able to solve 32 programs that Bebop could not solve within a 5 minute timeout.* μZ times out on just one program where Bebop also times out.



In other experiments we use μZ successfully on instantiations of timed transition systems, the examples in this paper, and a set of device drivers provided by Ken McMillan. They use arithmetic for reasoning about pointer offsets so Farkas-Conflict also suffices for verifying safety properties of these programs.

7 Conclusions, Related and Future Work

We generalized PDR in two directions. To solve general Horn clauses we first developed an abstract account of PDR and leveraged it for nonlinear predicate transformers. We also provided a solution to lifting PDR to linear real arithmetic. The solution uses a generalization of unsatisfiable cores for theories. The idea is to compute an interpolant based on the unit literals from a spurious counter example. We applied it to timed automata (with push-down capabilities). This is a new algorithm for timed automata, but not a new decidability result. Other

[2] http://research.microsoft.com/slam

extensions such as vector addition systems can be formulated in Datalog [14]. These extensions are not addressed here.

PDR can be seen as an instance of a Counter Example Guided Abstraction Refinement [5]. It refines state abstractions while avoiding approximating or unfolding the transition relation. Related approaches [2, 12, 8, 7, 1] also refine transition relations. In several cases (and in contrast to PDR), the abstraction refinement loop relies on unfolding the transition relation up to a certain depth. Of particular interest is [7], which explicates the connection between proof rules and solving Horn clauses.

Generalizing PDR to theories is an open-ended enterprise. The experiments so far indicate that Generalized PDR is attractive as a tool for satisfiability modulo fixed-points. Nevertheless, several extensions and optimizations should be pursued and there are several avenues for future work. A study of *weakest* T-*unsat cores* deserves attention from both an algorithmic point of view and from a point of view of commonly used theories. Our implementation in μZ works with algebraic data-types, but not yet with general uninterpreted functions. We believe uninterpreted functions can be handled by extending models to carry also a congruence class of terms. The corresponding version of Farkas-Conflict is then super-position on T-conflicts from congruence closure. We would also like to generalize other parts of PDR, in particular the crucial Induction rule. The implementations of PDR we are aware of use cheap strategies, they pick random literals in clauses and try to drop them one-by-one until a limit (of 4) failed strengthening attempts is reached. It is tempting to speculate of other generalizations for strengthening clauses. For example, $(\varphi \vee \neg(x \leq y+1) \vee \neg(z+2 \leq x)) \in R_i$ could be strengthened to $(\varphi \vee \neg(x+1 \leq y))$, and $(x \not\approx y \vee \varphi[x]) \in R_i$ could be strengthened to $\varphi[y]$.

Acknowledgments: Thanks to Natarajan Shankar, Josh Berdine, Bruno Dutertre, Sam Owre and the reviewers for significant constructive feedback. Also thanks to Andrey Rybalchenko and Ken McMillan for numerous discussions.

References

1. A. Albarghouthi, A. Gurfinkel, and M. Chechik. Whale: An interpolation-based algorithm for inter-procedural verification. In *VMCAI*, pages 39–55, 2012.
2. T. Ball and S. K. Rajamani. The SLAM project: debugging system software via static analysis. *SIGPLAN Not.*, 37(1):1–3, 2002.
3. A. Blass and Y. Gurevich. Existential fixed-point logic. In *Computation Theory and Logic*, pages 20–36, 1987.
4. A. R. Bradley. Sat-based model checking without unrolling. In *VMCAI*, 2011.
5. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. *Journal of the ACM*, 50(5):752–794, 2003.
6. N. Een, A. Mishchenko, and R. Brayton. Efficient implementation of property-directed reachability. In *FMCAD*, 2011.
7. S. Grebenshchikov, N. P. Lopes, C. Popeea, and A. Rybalchenko. Synthesizing software verifiers from proof rules. In *PLDI*, 2012.
8. A. Gupta, C. Popeea, and A. Rybalchenko. Solving Recursion-Free Horn Clauses over LI+UIF. In *APLAS*, pages 188–203, 2011.

9. K. Hoder, N. Bjørner, and L. M. de Moura. μZ- an efficient engine for fixed points with constraints. In *CAV*, pages 457–462, 2011.
10. M. H. Liffiton and K. A. Sakallah. Algorithms for computing minimal unsatisfiable subsets of constraints. *J. Autom. Reasoning*, 40(1):1–33, 2008.
11. Z. Manna and A. Pnueli. *Temporal verification of reactive systems - safety*. 1995.
12. K. L. McMillan. Interpolants from Z3 proofs. In *FMCAD*, 2011.
13. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an abstract DPLL procedure to DPLL(T). *J. ACM*, 53(6), 2006.
14. P. Z. Revesz. Safe datalog queries with linear constraints. In *CP*, 1998.