Exploiting Adjoints in Property Directed Reachability Analysis

Mayuko Kori[1,2](✉), Flavio Ascari[3], Filippo Bonchi[3], Roberto Bruni[3], Roberta Gori[3], and Ichiro Hasuo[1,2]

[1] National Institute of Informatics, Tokyo, Japan
{mkori,hasuo}@nii.ac.jp
[2] The Graduate University for Advanced Studies (SOKENDAI), Hayama, Japan
[3] Dipartimento di Informatica, Università di Pisa, Pisa, Italy
flavio.ascari@phd.unipi.it,
{filippo.bonchi,roberto.bruni,roberta.gori}@unipi.it

Abstract. We formulate, in lattice-theoretic terms, two novel algorithms inspired by Bradley's property directed reachability algorithm. For finding safe invariants or counterexamples, the first algorithm exploits over-approximations of both forward and backward transition relations, expressed abstractly by the notion of adjoints. In the absence of adjoints, one can use the second algorithm, which exploits lower sets and their principals. As a notable example of application, we consider quantitative reachability problems for Markov Decision Processes.

Keywords: PDR · Lattice theory · Adjoints · MDPs · Over-approximation

1 Introduction

Property directed reachability analysis (PDR) refers to a class of verification algorithms for solving safety problems of transition systems [5,12]. Its essence consists of 1) interleaving the construction of an *inductive invariant* (a *positive chain*) with that of a *counterexample* (a *negative sequence*), and 2) making the two sequences *interact*, with one narrowing down the search space for the other.

PDR algorithms have shown impressive performance both in hardware and software verification, leading to active research [15,18,28,29] going far beyond its original scope. For instance, an abstract domain [8] capturing the over-approximation exploited by PDR has been recently introduced in [13], while PrIC3 [3] extended PDR for quantitative verification of probabilistic systems.

Research supported by MIUR PRIN Project 201784YSZ5 *ASPRA*, by JST ERATO HASUO Metamathematics for Systems Design Project JPMJER1603, by JST CREST Grant JPMJCR2012, by JSPS DC KAKENHI Grant 22J21742 and by EU Next-GenerationEU (NRRP) SPOKE 10, Mission 4, Component 2, Investment N. 1.4, CUP N. I53C22000690001.

C. Enea and A. Lal (Eds.): CAV 2023, LNCS 13965, pp. 41–63, 2023.
https://doi.org/10.1007/978-3-031-37703-7_3

To uncover the abstract principles behind PDR and its extensions, Kori et al. proposed LT-PDR [19], a generalisation of PDR in terms of lattice/category theory. LT-PDR can be instantiated using domain-specific *heuristics* to create effective algorithms for different kinds of systems such as Kripke structures, Markov Decision Processes (MDPs), and Markov reward models. However, the theory in [19] does not offer guidance on devising concrete heuristics.

Adjoints in PDR. Our approach shares the same vision of LT-PDR, but we identify different principles: *adjunctions* are the core of our toolset.

An adjunction $f \dashv g$ is one of the central concepts in category theory [23]. It is prevalent in various fields of computer science, too, such as abstract interpretation [8] and functional programming [22]. Our use of adjoints in this work comes in the following two flavours.

$$\begin{tikzcd} A \arrow[r, bend right, "g"'] \arrow[r, phantom, "\perp"] & C \arrow[l, bend right, "f"'] \end{tikzcd}$$

- (forward-backward adjoint) f describes the *forward semantics* of a transition system, while g is the *backward* one, where we typically have $A = C$.
- (abstraction-concretization adjoint) C is a concrete semantic domain, and A is an abstract one, much like in abstract interpretation. An adjoint enables us to convert a fixed-point problem in C to that in A.

Our Algorithms. The problem we address is the standard lattice theoretical formulation of safety problems, namely whether the least fixed point of a continuous map b over a complete lattice $(L, \sqsubseteq)$ is below a given element $p \in L$. In symbols $\mu b \sqsubseteq_? p$. We present two algorithms.

The first one, named `AdjointPDR`, assumes to have an element $i \in L$ and two adjoints $f \dashv g \colon L \to L$, representing respectively initial states, forward semantics and backward semantics (see right) such that $b(x) = f(x) \sqcup i$ for all $x \in L$. Under this assumption, we have the following equivalences (they follow from the Knaster-Tarski theorem, see §2):

$$\begin{tikzcd} L \arrow[r, bend right, "g"'] \arrow[r, phantom, "\perp"] & L \arrow[l, bend right, "f"'] \end{tikzcd}$$

$$\mu b \sqsubseteq p \quad \Leftrightarrow \quad \mu(f \sqcup i) \sqsubseteq p \quad \Leftrightarrow \quad i \sqsubseteq \nu(g \sqcap p),$$

where $\mu(f \sqcup i)$ and $\nu(g \sqcap p)$ are, by the Kleene theorem, the limits of the *initial* and *final* chains illustrated below.

$$\bot \sqsubseteq i \sqsubseteq f(i) \sqcup i \sqsubseteq \cdots \qquad\qquad \cdots \sqsubseteq g(p) \sqcap p \sqsubseteq p \sqsubseteq \top$$

As positive chain, PDR exploits an over-approximation of the initial chain: it is made greater to accelerate convergence; still it has to be below p.

The distinguishing feature of `AdjointPDR` is to take as a negative sequence (that is a sequential construction of potential counterexamples) an over-approximation of the final chain. This crucially differs from the negative sequence of LT-PDR, namely an under-approximation of the computed positive chain.

We prove that `AdjointPDR` is sound (Theorem 5) and does not loop (Proposition 7) but since, the problem $\mu b \sqsubseteq_? p$ is not always decidable, we cannot prove termination. Nevertheless, `AdjointPDR` allows for a formal theory of heuristics that are essential when instantiating the algorithm to concrete problems.

The theory prescribes the choices to obtain the boundary executions, using initial and final chains (Proposition 10); it thus identifies a class of heuristics guaranteeing termination when answers are negative (Theorem 12).

`AdjointPDR`'s assumption of a forward-backward adjoint $f \dashv g$, however, does not hold very often, especially in probabilistic settings. Our second algorithm `AdjointPDR`$^{\downarrow}$ circumvents this problem by extending the lattice for the negative sequence, from L to the lattice $L^{\downarrow}$ of *lower sets* in L.

Specifically, by using the second form of adjoints, namely an abstraction-concretization pair, the problem $\mu b \sqsubseteq_? p$ in L can be translated to an equivalent problem on $b^{\downarrow}$ in $L^{\downarrow}$, for which an adjoint $b^{\downarrow} \dashv b^{\downarrow}_r$ is guaranteed. This allows one to run `AdjointPDR` in the lattice $L^{\downarrow}$. We then notice that the search for a positive chain can be conveniently restricted to principals in $L^{\downarrow}$, which have representatives in L. The resulting algorithm, using L for positive chains and $L^{\downarrow}$ for negative sequences, is `AdjointPDR`$^{\downarrow}$.

$$\begin{tikzcd} L \arrow[loop left, "b"] \arrow[r, bend right, "(-)^{\downarrow}"'] \arrow[r, phantom, "\perp"] & L^{\downarrow} \arrow[l, bend right, "\bigsqcup"'] \arrow[loop right, "b^{\downarrow} \dashv b^{\downarrow}_r"] \end{tikzcd}$$

The use of lower sets for the negative sequence is a key advantage. It not only avoids the restrictive assumption on forward-backward adjoints $f \dashv g$, but also enables a more thorough search for counterexamples. `AdjointPDR`$^{\downarrow}$ can simulate step-by-step LT-PDR (Theorem 17), while the reverse is not possible due to a single negative sequence in `AdjointPDR`$^{\downarrow}$ potentially representing multiple (Proposition 18) or even all (Proposition 19) negative sequences in LT-PDR.

Concrete Instances. Our lattice-theoretic algorithms yield many concrete instances: the original IC3/PDR [5,12] as well as Reverse PDR [27] are instances of `AdjointPDR` with L being the powerset of the state space; since LT-PDR can be simulated by `AdjointPDR`$^{\downarrow}$, the latter generalizes all instances in [19].

As a notable instance, we apply `AdjointPDR`$^{\downarrow}$ to MDPs, specifically to decide if the maximum reachability probability [1] is below a given threshold. Here the lattice $L = [0,1]^S$ is that of fuzzy predicates over the state space S. Our theory provides guidance to devise two heuristics, for which we prove negative termination (Corollary 20). We present its implementation in Haskell, and its experimental evaluation, where comparison is made against existing probabilistic PDR algorithms (PrIC3 [3], LT-PDR [19]) and a non-PDR one (Storm [11]). The performance of `AdjointPDR`$^{\downarrow}$ is encouraging—it supports the potential of PDR algorithms in probabilistic model checking. The experiments also indicate the importance of having a variety of heuristics, and thus the value of our adjoint framework that helps coming up with those.

Additionally, we found that abstraction features of Haskell allows us to code lattice-theoretic algorithms almost literally ($\sim$100 lines). Implementing a few heuristics takes another $\sim$240 lines. This way, we found that mathematical abstraction can directly help easing implementation effort.

Related Work. Reverse PDR [27] applies PDR from unsafe states using a backward transition relation $\mathbf{T}$ and tries to prove that initial states are unreachable. Our right adjoint g is also backward, but it differs from $\mathbf{T}$ in the presence of nondeterminism: roughly, $\mathbf{T}(X)$ is the set of states which *can* reach X in one step, while $g(X)$ are states which *only* reach X in one step. *fb*PDR [28,29] runs PDR and Reverse PDR in parallel with shared information. Our work uses both forward and backward directions (the pair $f \dashv g$), too, but approximate differently: Reverse PDR over-approximates the set of states that can reach an unsafe state, while we over-approximate the set of states that only reach safe states.

The comparison with LT-PDR [19] is extensively discussed in Sect. 4.2. PrIC3 [3] extended PDR to MDPs, which are our main experimental ground: Sect. 6 compares the performances of PrIC3, LT-PDR and $\texttt{AdjointPDR}^{\downarrow}$.

We remark that PDR has been applied to other settings, such as software model checking using theories and SMT-solvers [6,21] or automated planning [30]. Most of them (e.g., software model checking) fall already in the generality of LT-PDR and thus they can be embedded in our framework.

It is also worth to mention that, in the context of abstract interpretation, the use of adjoints to construct initial and final chains and exploit the interaction between their approximations has been investigated in several works, e.g., [7].

Structure of the Paper. After recalling some preliminaries in Sect. 2, we present `AdjointPDR` in Sect. 3 and $\texttt{AdjointPDR}^{\downarrow}$ in Sect. 4. In Sect. 5 we introduce the heuristics for the max reachability problems of MDPs, that are experimentally tested in Sect. 6.

2 Preliminaries and Notation

We assume that the reader is familiar with lattice theory, see, e.g., [10]. We use $(L, \sqsubseteq)$, $(L_1, \sqsubseteq_1)$, $(L_2, \sqsubseteq_2)$ to range over complete lattices and x, y, z to range over their elements. We omit subscripts and order relations whenever clear from the context. As usual, $\bigsqcup$ and $\bigsqcap$ denote least upper bound and greatest lower bound, $\sqcup$ and $\sqcap$ denote join and meet, $\top$ and $\bot$ top and bottom. Hereafter we will tacitly assume that all maps are monotone. Obviously, the identity map $id\colon L \to L$ and the composition $f \circ g\colon L_1 \to L_3$ of two monotone maps $g\colon L_1 \to L_2$ and $f\colon L_2 \to L_3$ are monotone. For a map $f\colon L \to L$, we inductively define $f^0 = id$ and $f^{n+1} = f \circ f^n$. Given $l\colon L_1 \to L_2$ and $r\colon L_2 \to L_1$, we say that l is the *left adjoint* of r, or equivalently that r is the *right adjoint* of l, written $l \dashv r$, when it holds that $l(x) \sqsubseteq_2 y$ iff $x \sqsubseteq_1 r(y)$ for all $x \in L_1$ and $y \in L_2$. Given a map $f\colon L \to L$, the element $x \in L$ is a *post-fixed point* iff $x \sqsubseteq f(x)$, a *pre-fixed point* iff $f(x) \sqsubseteq x$ and a *fixed point* iff $x = f(x)$. Pre, post and fixed points form complete lattices: we write μf and νf for the least and greatest fixed point.

Several problems relevant to computer science can be reduced to check if $\mu b \sqsubseteq p$ for a monotone map $b\colon L \to L$ on a complete lattice L. The Knaster-Tarski fixed-point theorem characterises μb as the least upper bound of all pre-fixed points of b and νb as the greatest lower bound of all its post-fixed points:

$$\mu b = \bigsqcap\{x \mid b(x) \sqsubseteq x\} \qquad\qquad \nu b = \bigsqcup\{x \mid x \sqsubseteq b(x)\} \ .$$

This immediately leads to two proof principles, illustrated below:

$$\frac{\exists x,\ b(x) \sqsubseteq x \sqsubseteq p}{\mu b \sqsubseteq p} \qquad\qquad \frac{\exists x,\ i \sqsubseteq x \sqsubseteq b(x)}{i \sqsubseteq \nu b} \tag{KT}$$

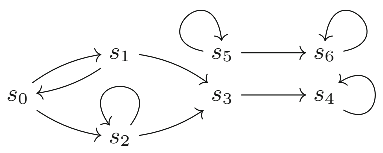


Fig. 1. The transition system of Example 1, with $S = \{s_0, \dots s_6\}$ and $I = \{s_0\}$.

By means of (KT), one can prove $\mu b \sqsubseteq p$ by finding some pre-fixed point x, often called *invariant*, such that $x \sqsubseteq p$. However, automatically finding invariants might be rather complicated, so most of the algorithms rely on another fixed-point theorem, usually attributed to Kleene. It characterises μb and νb as the least upper bound and the greatest lower bound, of the *initial* and *final chains*:

$$\bot \sqsubseteq b(\bot) \sqsubseteq b^2(\bot) \sqsubseteq \cdots \quad \text{and} \quad \cdots \sqsubseteq b^2(\top) \sqsubseteq b(\top) \sqsubseteq \top. \quad \text{That is,} \qquad \text{(Kl)}$$
$$\mu b = \bigsqcup_{n \in \mathbb{N}} b^n(\bot), \qquad \nu b = \bigsqcap_{n \in \mathbb{N}} b^n(\top).$$

The assumptions are stronger than for Knaster-Tarski: for the leftmost statement, it requires the map b to be ω-*continuous* (i.e., it preserves $\bigsqcup$ of ω-chains) and, for the rightmost ω-*co-continuous* (similar but for $\bigsqcap$). Observe that every left adjoint is continuous and every right adjoint is co-continuous (see e.g. [23]).

As explained in [19], property directed reachability (PDR) algorithms [5] exploits (KT) to try to prove the inequation and (Kl) to refute it. In the algorithm we introduce in the next section, we further assume that b is of the form $f \sqcup i$ for some element $i \in L$ and map $f \colon L \to L$, namely $b(x) = f(x) \sqcup i$ for all $x \in L$. Moreover we require f to have a right adjoint $g \colon L \to L$. In this case

$$\mu(f \sqcup i) \sqsubseteq p \qquad \text{iff} \qquad i \sqsubseteq \nu(g \sqcap p) \tag{1}$$

(which is easily shown using the Knaster-Tarski theorem) and $(f \sqcup i)$ and $(g \sqcap p)$ are guaranteed to be (co)continuous. Since $f \dashv g$ and left and right adjoints preserve, resp., arbitrary joins and meets, then for all $n \in \mathbb{N}$

$$(f \sqcup i)^n(\bot) = \bigsqcup_{j<n} f^j(i) \qquad (g \sqcap p)^n(\top) = \bigsqcap_{j<n} g^j(p) \tag{2}$$

which by (Kl) provide useful characterisations of least and greatest fixed points.

$$\mu(f \sqcup i) = \bigsqcup_{n \in \mathbb{N}} f^n(i) \qquad \nu(g \sqcap p) = \bigsqcap_{n \in \mathbb{N}} g^n(p) \qquad \text{(Kl}\dashv\text{)}$$

We conclude this section with an example that we will often revisit. It also provides a justification for the intuitive terminology that we sporadically use.

Example 1 (Safety problem for transition systems). A *transition system* consists of a triple (S, I, δ) where S is a set of states, $I \subseteq S$ is a set of initial states, and $\delta \colon S \to \mathcal{P}S$ is a transition relation. Here $\mathcal{P}S$ denotes the powerset of S, which

$$x_0 = \bot \quad \text{(I0)}$$
$$1 \leq k \leq n \quad \text{(I1)}$$
$$\forall j \in [0, n-2],\ x_j \sqsubseteq x_{j+1} \quad \text{(I2)}$$

$$i \sqsubseteq x_1 \quad \text{(P1)}$$
$$x_{n-2} \sqsubseteq p \quad \text{(P2)}$$
$$\forall j \in [0, n-2],\ f(x_j) \sqsubseteq x_{j+1} \quad \text{(P3)}$$
$$\forall j \in [0, n-2],\ x_j \sqsubseteq g(x_{j+1}) \quad \text{(P3a)}$$

$$\text{If } \boldsymbol{y} \neq \varepsilon \text{ then } p \sqsubseteq y_{n-1} \quad \text{(N1)}$$
$$\forall j \in [k, n-2],\ g(y_{j+1}) \sqsubseteq y_j \quad \text{(N2)}$$

$$\forall j \in [k, n-1],\ x_j \not\sqsubseteq y_j \quad \text{(PN)}$$
$$\forall j \in [0, n-1],\ (f \sqcup i)^j(\bot) \sqsubseteq x_j \sqsubseteq (g \sqcap p)^{n-1-j}(\top) \quad \text{(A1)}$$
$$\forall j \in [1, n-1],\ x_{j-1} \sqsubseteq g^{n-1-j}(p) \quad \text{(A2)}$$
$$\forall j \in [k, n-1],\ g^{n-1-j}(p) \sqsubseteq y_j \quad \text{(A3)}$$

Fig. 2. Invariants of `AdjointPDR`.

forms a complete lattice ordered by inclusion $\subseteq$. By defining $F\colon \mathcal{P}S \to \mathcal{P}S$ as $F(X) \overset{\text{def}}{=} \bigcup_{s \in X} \delta(s)$ for all $X \in \mathcal{P}S$, one has that $\mu(F \cup I)$ is the set of all states reachable from I. Therefore, for any $P \in \mathcal{P}S$, representing some safety property, $\mu(F \cup I) \subseteq P$ holds iff all reachable states are safe. It is worth to remark that F has a right adjoint $G\colon \mathcal{P}S \to \mathcal{P}S$ defined for all $X \in \mathcal{P}S$ as $G(X) \overset{\text{def}}{=} \{s \mid \delta(s) \subseteq X\}$. Thus by (1), $\mu(F \cup I) \subseteq P$ iff $I \subseteq \nu(G \cap P)$.

Consider the transition system in Fig. 1. Hereafter we write S_j for the set of states $\{s_0, s_1, \dots, s_j\}$ and we fix the set of safe states to be $P = S_5$. It is immediate to see that $\mu(F \cup I) = S_4 \subseteq P$. Automatically, this can be checked with the initial chains of $(F \cup I)$ or with the final chain of $(G \cap P)$ displayed below on the left and on the right, respectively.

$$\emptyset \subseteq I \subseteq S_2 \subseteq S_3 \subseteq S_4 \subseteq S_4 \subseteq \cdots \qquad\qquad \cdots \subseteq S_4 \subseteq S_4 \subseteq P \subseteq S$$

The $(j+1)$-th element of the initial chain contains all the states that can be reached by I in at most j transitions, while $(j+1)$-th element of the final chain contains all the states that in at most j transitions reach safe states only.

3 Adjoint PDR

In this section we present `AdjointPDR`, an algorithm that takes in input a tuple (i, f, g, p) with $i, p \in L$ and $f \dashv g\colon L \to L$ and, if it terminates, it returns true whenever $\mu(f \sqcup i) \sqsubseteq p$ and false otherwise.

The algorithm manipulates two sequences of elements of L: $\boldsymbol{x} \overset{\text{def}}{=} x_0, \dots, x_{n-1}$ of length n and $\boldsymbol{y} \overset{\text{def}}{=} y_k, \dots y_{n-1}$ of length $n-k$. These satisfy, through the executions of `AdjointPDR`, the invariants in Fig. 2. Observe that, by (A1), x_j over-approximates the j-th element of the initial chain, namely $(f \sqcup i)^j(\bot) \sqsubseteq x_j$, while, by (A3), the j-indexed element y_j of $\boldsymbol{y}$ over-approximates $g^{n-j-1}(p)$ that, borrowing the terminology of Example 1, is the set of states which are safe in $n-j-1$ transitions. Moreover, by (PN), the element y_j witnesses that x_j is unsafe, i.e., that $x_j \not\sqsubseteq g^{n-1-j}(p)$ or equivalently $f^{n-j-1}(x_j) \not\sqsubseteq p$. Notably, $\boldsymbol{x}$ is a positive chain and $\boldsymbol{y}$ a negative sequence, according to the definitions below.

```
                         AdjointPDR $(i, f, g, p)$

<INITIALISATION>
  $(\boldsymbol{x}\|\boldsymbol{y})_{n,k}$ := $(\bot, \top\|\varepsilon)_{2,2}$
<ITERATION>                                        % $\boldsymbol{x}, \boldsymbol{y}$ not conclusive
  case $(\boldsymbol{x}\|\boldsymbol{y})_{n,k}$ of
      $\boldsymbol{y} = \varepsilon$ and $x_{n-1} \sqsubseteq p$ :                    %(Unfold)
          $(\boldsymbol{x}\|\boldsymbol{y})_{n,k}$ := $(\boldsymbol{x}, \top\|\varepsilon)_{n+1,n+1}$
      $\boldsymbol{y} = \varepsilon$ and $x_{n-1} \not\sqsubseteq p$ :                    %(Candidate)
          choose $z \in L$ such that $x_{n-1} \not\sqsubseteq z$ and $p \sqsubseteq z$;
          $(\boldsymbol{x}\|\boldsymbol{y})_{n,k}$ := $(\boldsymbol{x}\|z)_{n,n-1}$
      $\boldsymbol{y} \neq \varepsilon$ and $f(x_{k-1}) \not\sqsubseteq y_k$ :              %(Decide)
          choose $z \in L$ such that $x_{k-1} \not\sqsubseteq z$ and $g(y_k) \sqsubseteq z$;
          $(\boldsymbol{x}\|\boldsymbol{y})_{n,k}$ := $(\boldsymbol{x}\|z, \boldsymbol{y})_{n,k-1}$
      $\boldsymbol{y} \neq \varepsilon$ and $f(x_{k-1}) \sqsubseteq y_k$ :              %(Conflict)
          choose $z \in L$ such that $z \sqsubseteq y_k$ and $(f \sqcup i)(x_{k-1} \sqcap z) \sqsubseteq z$;
          $(\boldsymbol{x}\|\boldsymbol{y})_{n,k}$ := $(\boldsymbol{x} \sqcap_k z\|\mathsf{tail}(\boldsymbol{y}))_{n,k+1}$
  endcase
<TERMINATION>
   if $\exists j \in [0, n-2] \,.\, x_{j+1} \sqsubseteq x_j$ then return true  % $\boldsymbol{x}$ conclusive
   if $i \not\sqsubseteq y_1$ then return false                  % $\boldsymbol{y}$ conclusive
```

Fig. 3. `AdjointPDR` algorithm checking $\mu(f \sqcup i) \sqsubseteq p$.

Definition 2 (positive chain). *A* positive chain *for $\mu(f \sqcup i) \sqsubseteq p$ is a finite chain $x_0 \sqsubseteq \cdots \sqsubseteq x_{n-1}$ in L of length $n \geq 2$ which satisfies* (P1)*,* (P2)*,* (P3) *in Fig. 2. It is* conclusive *if $x_{j+1} \sqsubseteq x_j$ for some $j \leq n-2$.*

In a conclusive positive chain, x_{j+1} provides an invariant for $f \sqcup i$ and thus, by (KT), $\mu(f \sqcup i) \sqsubseteq p$ holds. So, when $\boldsymbol{x}$ is conclusive, `AdjointPDR` returns true.

Definition 3 (negative sequence). *A* negative sequence *for $\mu(f \sqcup i) \sqsubseteq p$ is a finite sequence $y_k, \ldots, y_{n-1}$ in L with $1 \leq k \leq n$ which satisfies* (N1) *and* (N2) *in Fig. 2. It is* conclusive *if $k = 1$ and $i \not\sqsubseteq y_1$.*

When $\boldsymbol{y}$ is conclusive, `AdjointPDR` returns false as y_1 provides a counterexample: (N1) and (N2) entail (A3) and thus $i \not\sqsubseteq y_1 \sqsupseteq g^{n-2}(p)$. By (Kl⊣), $g^{n-2}(p) \sqsupseteq \nu(g \sqcap p)$ and thus $i \not\sqsubseteq \nu(g \sqcap p)$. By (1), $\mu(f \sqcup i) \not\sqsubseteq p$.

The pseudocode of the algorithm is displayed in Fig. 3, where we write $(\boldsymbol{x}\|\boldsymbol{y})_{n,k}$ to compactly represents the state of the algorithm: the pair (n, k) is called the *index* of the state, with $\boldsymbol{x}$ of length n and $\boldsymbol{y}$ of length $n - k$. When $k = n$, $\boldsymbol{y}$ is the empty sequence ε. For any $z \in L$, we write $\boldsymbol{x}, z$ for the chain $x_0, \ldots, x_{n-1}, z$ of length $n + 1$ and $z, \boldsymbol{y}$ for the sequence $z, y_k, \ldots y_{n-1}$ of length $n-(k-1)$. Moreover, we write $\boldsymbol{x} \sqcap_j z$ for the chain $x_0 \sqcap z, \ldots, x_j \sqcap z, x_{j+1}, \ldots, x_{n-1}$. Finally, $\mathsf{tail}(\boldsymbol{y})$ stands for the tail of $\boldsymbol{y}$, namely $y_{k+1}, \ldots y_{n-1}$ of length $n-(k+1)$.

The algorithm starts in the initial state $s_0 \overset{\text{def}}{=} (\bot, \top\|\varepsilon)_{2,2}$ and, unless one of $\boldsymbol{x}$ and $\boldsymbol{y}$ is conclusive, iteratively applies one of the four mutually exclusive rules: (Unfold), (Candidate), (Decide) and (Conflict). The rule (Unfold) extends the positive chain by one element when the negative sequence is empty and the positive chain is under p; since the element introduced by (Unfold) is $\top$, its application typically triggers rule (Candidate) that starts the negative sequence with an over-approximation of p. Recall that the role of y_j is to witness that x_j is unsafe. After (Candidate) either (Decide) or (Conflict) are possible: if y_k witnesses that, besides x_k, also $f(x_{k-1})$ is unsafe, then (Decide) is used to further extend the negative sequence to witness that x_{k-1} is unsafe; otherwise, the rule (Conflict) improves the precision of the positive chain in such a way that y_k no longer witnesses $x_k \sqcap z$ unsafe and, thus, the negative sequence is shortened.

Note that, in (Candidate), (Decide) and (Conflict), the element $z \in L$ is chosen among a set of possibilities, thus `AdjointPDR` is nondeterministic.

To illustrate the executions of the algorithm, we adopt a labeled transition system notation. Let $\mathcal{S} \overset{\text{def}}{=} \{(\boldsymbol{x}\|\boldsymbol{y})_{n,k} \mid n \geq 2,\, k \leq n,\, \boldsymbol{x} \in L^n \text{ and } \boldsymbol{y} \in L^{n-k}\}$ be the set of all possible states of `AdjointPDR`. We call $(\boldsymbol{x}\|\boldsymbol{y})_{n,k} \in \mathcal{S}$ *conclusive* if $\boldsymbol{x}$ or $\boldsymbol{y}$ are such. When $s \in \mathcal{S}$ is not conclusive, we write $s\overset{D}{\to}$ to mean that s satisfies the guards in the rule (Decide), and $s\overset{D}{\to}_z s'$ to mean that, being (Decide) applicable, `AdjointPDR` moves from state s to s' by choosing z. Similarly for the other rules: the labels Ca, Co and U stands for (Candidate), (Conflict) and (Unfold), respectively. When irrelevant we omit to specify labels and choices and we just write $s \to s'$. As usual $\to^+$ stands for the transitive closure of $\to$ while $\to^*$ stands for the reflexive and transitive closure of $\to$.

Example 4. Consider the safety problem in Example 1. Below we illustrate two possible computations of `AdjointPDR` that differ for the choice of z in (Conflict). The first run is conveniently represented as the following series of transitions.

$$\begin{gathered}(\emptyset, S\|\varepsilon)_{2,2} \overset{Ca}{\to}_P (\emptyset, S\|P)_{2,1} \overset{Co}{\to}_I (\emptyset, I\|\varepsilon)_{2,2} \overset{U}{\to} (\emptyset, I, S\|\varepsilon)_{3,3} \overset{Ca}{\to}_P (\emptyset, I, S\|P)_{3,2} \\ \overset{Co}{\to}_{S_2} (\emptyset, I, S_2\|\varepsilon)_{3,3} \overset{U}{\to}\overset{Ca}{\to}_P (\emptyset, I, S_2, S\|P)_{4,3} \overset{Co}{\to}_{S_3} (\emptyset, I, S_2, S_3\|\varepsilon)_{4,4} \overset{U}{\to}\overset{Ca}{\to}_P (\emptyset, I, S_2, S_3, S\|P)_{5,4} \\ \overset{Co}{\to}_{S_4} (\emptyset, I, S_2, S_3, S_4\|\varepsilon)_{5,5} \overset{U}{\to}\overset{Ca}{\to}_P (\emptyset, I, S_2, S_3, S_4, S\|P)_{6,5} \overset{Co}{\to}_{S_4} (\emptyset, I, S_2, S_3, S_4, S_4\|\varepsilon)_{6,6}\end{gathered}$$

The last state returns true since $x_4 = x_5 = S_4$. Observe that the elements of $\boldsymbol{x}$, with the exception of the last element x_{n-1}, are those of the initial chain of $(F \cup I)$, namely, x_j is the set of states reachable in at most $j-1$ steps. In the second computation, the elements of $\boldsymbol{x}$ are roughly those of the final chain of $(G \cap P)$. More precisely, after (Unfold) or (Candidate), x_{n-j} for $j < n-1$ is the set of states which only reach safe states within j steps.

$$\begin{gathered}(\emptyset, S\|\varepsilon)_{2,2} \overset{Ca}{\to}_P (\emptyset, S\|P)_{2,1} \overset{Co}{\to}_P (\emptyset, P\|\varepsilon)_{2,2} \\ \overset{U}{\to}\overset{Ca}{\to}_P (\emptyset, P, S\|P)_{3,2} \overset{D}{\to}_{S_4} (\emptyset, P, S\|S_4, P)_{3,1} \overset{Co}{\to}_{S_4} (\emptyset, S_4, S\|P)_{3,2} \overset{Co}{\to}_P (\emptyset, S_4, P\|\varepsilon)_{3,3} \\ \overset{U}{\to}\overset{Ca}{\to}_P (\emptyset, S_4, P, S\|P)_{4,3} \overset{D}{\to}_{S_4} (\emptyset, S_4, P, S\|S_4, P)_{4,2} \overset{Co}{\to}_{S_4} (\emptyset, S_4, S_4, S\|P)_{4,3}\end{gathered}$$

Observe that, by invariant (A1), the values of $\boldsymbol{x}$ in the two runs are, respectively, the least and the greatest values for all possible computations of `AdjointPDR`.

Theorem 5.1 follows by invariants (I2), (P1), (P3) and (KT); Theorem 5.2 by (N1), (N2) and (Kl⊣). Note that both results hold for any choice of z.

Theorem 5 (Soundness). `AdjointPDR` *is sound. Namely,*

1. *If* `AdjointPDR` *returns true then* $\mu(f \sqcup i) \sqsubseteq p$.
2. *If* `AdjointPDR` *returns false then* $\mu(f \sqcup i) \not\sqsubseteq p$.

3.1 Progression

It is necessary to prove that in any step of the execution, if the algorithm does not return true or false, then it can progress to a new state, not yet visited. To this aim we must deal with the subtleties of the non-deterministic choice of the element z in (Candidate), (Decide) and (Conflict). The following proposition ensures that, for any of these three rules, there is always a possible choice.

Proposition 6 (Canonical choices). *The following are always possible:*
1. in (Candidate) $z = p$;
2. in (Decide) $z = g(y_k)$;
3. in (Conflict) $z = y_k$;
4. in (Conflict) $z = (f \sqcup i)(x_{k-1})$.
Thus, for all non-conclusive $s \in \mathcal{S}$, if $s_0 \to^* s$ then $s \to$.

Then, Proposition 7 ensures that `AdjointPDR` always traverses new states.

Proposition 7 (Impossibility of loops). *If $s_0 \to^* s \to^+ s'$, then $s \neq s'$.*

Observe that the above propositions entail that `AdjointPDR` terminates whenever the lattice L is finite, since the set of reachable states is finite in this case.

Example 8. For (I, F, G, P) as in Example 1, `AdjointPDR` behaves essentially as IC3/PDR [5], solving reachability problems for transition systems with finite state space S. Since the lattice $\mathcal{P}S$ is also finite, `AdjointPDR` always terminates.

3.2 Heuristics

The nondeterministic choices of the algorithm can be resolved by using heuristics. Intuitively, a heuristic chooses for any states $s \in \mathcal{S}$ an element $z \in L$ to be possibly used in (Candidate), (Decide) or (Conflict), so it is just a function $h\colon \mathcal{S} \to L$. When defining a heuristic, we will avoid to specify its values on conclusive states or in those performing (Unfold), as they are clearly irrelevant.

With a heuristic, one can instantiate `AdjointPDR` by making the choice of z as prescribed by h. Syntactically, this means to erase from the code of Fig. 3 the three lines of `choose` and replace them by z`:=` $h(\,(\boldsymbol{x}\|\boldsymbol{c})_{n,k}\,)$. We call `AdjointPDR`$_h$ the resulting deterministic algorithm and write $s \to_h s'$ to mean that `AdjointPDR`$_h$ moves from state s to s'. We let $\mathcal{S}^h \overset{\text{def}}{=} \{s \in \mathcal{S} \mid s_0 \to_h^* s\}$ be the sets of all states reachable by `AdjointPDR`$_h$.

Definition 9 (legit heuristic). *A heuristic $h\colon \mathcal{S} \to L$ is called* legit *whenever for all $s, s' \in \mathcal{S}^h$, if $s \to_h s'$ then $s \to s'$.*

When h is legit, the only execution of the deterministic algorithm `AdjointPDR`$_h$ is one of the possible executions of the non-deterministic algorithm `AdjointPDR`.

The canonical choices provide two legit heuristics: first, we call *simple* any legit heuristic h that chooses z in (Candidate) and (Decide) as in Proposition 6:

$$(\boldsymbol{x}\|\boldsymbol{y})_{n,k} \mapsto \begin{cases} p & \text{if } (\boldsymbol{x}\|\boldsymbol{y})_{n,k} \overset{Ca}{\to} \\ g(y_k) & \text{if } (\boldsymbol{x}\|\boldsymbol{y})_{n,k} \overset{D}{\to} \end{cases} \tag{3}$$

Then, if the choice in (Conflict) is like in Proposition 6.4, we call h *initial*; if it is like in Proposition 6.3, we call h *final*. Shortly, the two legit heuristics are:

simple initial	(3) and $(\boldsymbol{x}\|\boldsymbol{y})_{n,k} \mapsto (f \sqcup i)(x_{k-1})$	if $(\boldsymbol{x}\|\boldsymbol{y})_{n,k} \in Co$
simple final	(3) and $(\boldsymbol{x}\|\boldsymbol{y})_{n,k} \mapsto y_k$	if $(\boldsymbol{x}\|\boldsymbol{y})_{n,k} \in Co$

Interestingly, with any simple heuristic, the sequence $\boldsymbol{y}$ takes a familiar shape:

Proposition 10. *Let $h\colon \mathcal{S} \to L$ be any simple heuristic. For all $(\boldsymbol{x}\|\boldsymbol{y})_{n,k} \in \mathcal{S}^h$, invariant* (A3) *holds as an equality, namely for all $j \in [k, n-1]$, $y_j = g^{n-1-j}(p)$.*

By the above proposition and (A3), the negative sequence $\boldsymbol{y}$ occurring in the execution of $\texttt{AdjointPDR}_h$, for a simple heuristic h, is the least amongst all the negative sequences occurring in any execution of `AdjointPDR`.

Instead, invariant (A1) informs us that the positive chain $\boldsymbol{x}$ is always in between the initial chain of $f \sqcup i$ and the final chain of $g \sqcap p$. Such values of $\boldsymbol{x}$ are obtained by, respectively, simple initial and simple final heuristic.

Example 11. Consider the two runs of `AdjointPDR` in Example 4. The first one exploits the simple initial heuristic and indeed, the positive chain $\boldsymbol{x}$ coincides with the initial chain. Analogously, the second run uses the simple final heuristic.

3.3 Negative Termination

When the lattice L is not finite, `AdjointPDR` may not terminate, since checking $\mu(f \sqcup i) \sqsubseteq p$ is not always decidable. In this section, we show that the use of certain heuristics can guarantee termination whenever $\mu(f \sqcup i) \not\sqsubseteq p$.

The key insight is the following: if $\mu(f \sqcup i) \not\sqsubseteq p$ then by (Kl), there should exist some $\tilde{n} \in \mathbb{N}$ such that $(f \sqcup i)^{\tilde{n}}(\bot) \not\sqsubseteq p$. By (A1), the rule (Unfold) can be applied only when $(f \sqcup i)^{n-1}(\bot) \sqsubseteq x_{n-1} \sqsubseteq p$. Since (Unfold) increases n and n is never decreased by other rules, then (Unfold) can be applied at most $\tilde{n}$ times.

The elements of negative sequences are introduced by rules (Candidate) and (Decide). If we guarantee that for any index (n, k) the heuristic in such cases returns a finite number of values for z, then one can prove termination. To make this formal, we fix $CaD^h_{n,k} \overset{\text{def}}{=} \{(\boldsymbol{x}\|\boldsymbol{y})_{n,k} \in \mathcal{S}^h \mid (\boldsymbol{x}\|\boldsymbol{y})_{n,k} \overset{Ca}{\to} \text{ or } (\boldsymbol{x}\|\boldsymbol{y})_{n,k} \overset{D}{\to}\}$, i.e., the set of all (n, k)-indexed states reachable by $\texttt{AdjointPDR}_h$ that trigger (Candidate) or (Decide), and $h(CaD^h_{n,k}) \overset{\text{def}}{=} \{h(s) \mid s \in CaD^h_{n,k}\}$, i.e., the set of all possible values returned by h in such states.

Theorem 12 (Negative termination). *Let h be a legit heuristic. If $h(CaD^h_{n,k})$ is finite for all n, k and $\mu(f \sqcup i) \not\sqsubseteq p$, then $\texttt{AdjointPDR}_h$ terminates.*

Corollary 13. *Let h be a simple heuristic. If $\mu(f \sqcup i) \not\sqsubseteq p$, then $\texttt{AdjointPDR}_h$ terminates.*

Note that this corollary ensures negative termination whenever we use the canonical choices in (Candidate) and (Decide) *irrespective of the choice for* (Conflict), therefore it holds for both simple initial and simple final heuristics.

4 Recovering Adjoints with Lower Sets

In the previous section, we have introduced an algorithm for checking $\mu b \sqsubseteq p$ whenever b is of the form $f \sqcup i$ for an element $i \in L$ and a left-adjoint $f\colon L \to L$. This, unfortunately, is not the case for several interesting problems, like the max reachability problem [1] that we will illustrate in Sect. 5.

The next result informs us that, under standard assumptions, one can transfer the problem of checking $\mu b \sqsubseteq p$ to lower sets, where adjoints can always be defined. Recall that, for a lattice $(L, \sqsubseteq)$, a *lower set* is a subset $X \subseteq L$ such that if $x \in X$ and $x' \sqsubseteq x$ then $x' \in X$; the set of lower sets of L forms a complete lattice $(L^{\downarrow}, \subseteq)$ with joins and meets given by union and intersection; as expected $\bot$ is $\emptyset$ and $\top$ is L. Given $b\colon L \to L$, one can define two functions $b^{\downarrow}, b^{\downarrow}_r\colon L^{\downarrow} \to L^{\downarrow}$ as $b^{\downarrow}(X) \overset{\text{def}}{=} b(X)^{\downarrow}$ and $b^{\downarrow}_r(X) \overset{\text{def}}{=} \{x \mid b(x) \in X\}$. It holds that $b^{\downarrow} \dashv b^{\downarrow}_r$.

$$\begin{tikzcd} (L,\sqsubseteq) \arrow[loop left, "b"] \arrow[r, bend right, "(-)^{\downarrow}"'] \arrow[r, phantom, "\perp"] & (L^{\downarrow},\subseteq) \arrow[l, bend right, "\bigsqcup"'] \arrow[loop right, "b^{\downarrow} \dashv b^{\downarrow}_r"] \end{tikzcd} \tag{4}$$

In the diagram above, $(-)^{\downarrow}\colon x \mapsto \{x' \mid x' \sqsubseteq x\}$ and $\bigsqcup\colon L^{\downarrow} \to L$ maps a lower set X into $\bigsqcup\{x \mid x \in X\}$. The maps $\bigsqcup$ and $(-)^{\downarrow}$ form a *Galois insertion*, namely $\bigsqcup \dashv (-)^{\downarrow}$ and $\bigsqcup(-)^{\downarrow} = id$, and thus one can think of (4) in terms of *abstract interpretation* [8,9]: $L^{\downarrow}$ represents the concrete domain, L the abstract domain and b is a sound abstraction of $b^{\downarrow}$. Most importantly, it turns out that b is *forward-complete* [4,14] w.r.t. $b^{\downarrow}$, namely the following equation holds.

$$(-)^{\downarrow} \circ b = b^{\downarrow} \circ (-)^{\downarrow} \tag{5}$$

Proposition 14. *Let $(L, \sqsubseteq)$ be a complete lattice, $p \in L$ and $b\colon L \to L$ be a ω-continuous map. Then $\mu b \sqsubseteq p$ iff $\mu(b^{\downarrow} \cup \bot^{\downarrow}) \subseteq p^{\downarrow}$.*

By means of Proposition 14, we can thus solve $\mu b \sqsubseteq p$ in L by running `AdjointPDR` on $(\bot^{\downarrow}, b^{\downarrow}, b^{\downarrow}_r, p^{\downarrow})$. Hereafter, we tacitly assume that b is ω-continuous.

4.1 `AdjointPDR`$^{\downarrow}$: Positive Chain in L, Negative Sequence in $L^{\downarrow}$

While `AdjointPDR` on $(\bot^{\downarrow}, b^{\downarrow}, b^{\downarrow}_r, p^{\downarrow})$ might be computationally expensive, it is the first step toward the definition of an efficient algorithm that exploits a convenient form of the positive chain.

A lower set $X \in L^{\downarrow}$ is said to be a *principal* if $X = x^{\downarrow}$ for some $x \in L$. Observe that the top of the lattice $(L^{\downarrow}, \subseteq)$ is a principal, namely $\top^{\downarrow}$, and that the meet (intersection) of two principals $x^{\downarrow}$ and $y^{\downarrow}$ is the principal $(x \sqcap y)^{\downarrow}$.

Suppose now that, in (Conflict), `AdjointPDR`$(\bot^{\downarrow}, b^{\downarrow}, b^{\downarrow}_r, p^{\downarrow})$ always chooses principals rather than arbitrary lower sets. This suffices to guarantee that all the elements of $\boldsymbol{x}$ are principals (with the only exception of x_0 which is constantly the bottom element of $L^{\downarrow}$ that, note, is $\emptyset$ and not $\bot^{\downarrow}$). In fact, the elements of

```
                              AdjointPDR$^{\downarrow}$ $(b, p)$

<INITIALISATION>
  $(\boldsymbol{x}\|\boldsymbol{Y})_{n,k}$ := $(\emptyset, \bot, \top\|\varepsilon)_{3,3}$
<ITERATION>
  case $(\boldsymbol{x}\|\boldsymbol{Y})_{n,k}$ of                    % $\boldsymbol{x}, \boldsymbol{Y}$ not conclusive
      $\boldsymbol{Y} = \varepsilon$ and $x_{n-1} \sqsubseteq p$ :                  %(Unfold)
         $(\boldsymbol{x}\|\boldsymbol{Y})_{n,k}$ := $(\boldsymbol{x}, \top\|\varepsilon)_{n+1,n+1}$
      $\boldsymbol{Y} = \varepsilon$ and $x_{n-1} \not\sqsubseteq p$ :                  %(Candidate)
         choose $Z \in L^{\downarrow}$ such that $x_{n-1} \notin Z$ and $p \in Z$;
         $(\boldsymbol{x}\|\boldsymbol{Y})_{n,k}$ := $(\boldsymbol{x}\|Z)_{n,n-1}$
      $\boldsymbol{Y} \neq \varepsilon$ and $b(x_{k-1}) \notin Y_k$ :                %(Decide)
         choose $Z \in L^{\downarrow}$ such that $x_{k-1} \notin Z$ and $b_r^{\downarrow}(Y_k) \subseteq Z$;
         $(\boldsymbol{x}\|\boldsymbol{Y})_{n,k}$ := $(\boldsymbol{x}\|Z, \boldsymbol{Y})_{n,k-1}$
      $\boldsymbol{Y} \neq \varepsilon$ and $b(x_{k-1}) \in Y_k$ :                %(Conflict)
         choose $z \in L$ such that $z \in Y_k$ and $b(x_{k-1} \sqcap z) \sqsubseteq z$;
         $(\boldsymbol{x}\|\boldsymbol{Y})_{n,k}$ := $(\boldsymbol{x} \sqcap_k z\|\mathsf{tail}(\boldsymbol{Y}))_{n,k+1}$
  endcase
<TERMINATION>
   if $\exists j \in [0, n-2] \,.\, x_{j+1} \sqsubseteq x_j$ then return true  % $\boldsymbol{x}$ conclusive
   if $Y_1 = \emptyset$ then return false                    % $\boldsymbol{Y}$ conclusive
```

Fig. 4. The algorithm AdjointPDR$^{\downarrow}$ for checking $\mu b \sqsubseteq p$: the elements of negative sequence are in $L^{\downarrow}$, while those of the positive chain are in L, with the only exception of x_0 which is constantly the bottom lower set $\emptyset$. For x_0, we fix $b(x_0) = \bot$.

$\boldsymbol{x}$ are all obtained by (Unfold), that adds the principal $\top^{\downarrow}$, and by (Conflict), that takes their meets with the chosen principal.

Since principals are in bijective correspondence with the elements of L, by imposing to AdjointPDR$(\bot^{\downarrow}, b^{\downarrow}, b_r^{\downarrow}, p^{\downarrow})$ to choose a principal in (Conflict), we obtain an algorithm, named AdjointPDR$^{\downarrow}$, where the elements of the positive chain are drawn from L, while the negative sequence is taken in $L^{\downarrow}$. The algorithm is reported in Fig. 4 where we use the notation $(\boldsymbol{x}\|\boldsymbol{Y})_{n,k}$ to emphasize that the elements of the negative sequence are lower sets of elements in L.

All definitions and results illustrated in Sect. 3 for AdjointPDR are inherited[1] by AdjointPDR$^{\downarrow}$, with the only exception of Proposition 6.3. The latter does not hold, as it prescribes a choice for (Conflict) that may not be a principal. In contrast, the choice in Proposition 6.4 is, thanks to (5), a principal. This means in particular that the simple initial heuristic is always applicable.

Theorem 15. *All results in Sect. 3, but Proposition 6.3, hold for AdjointPDR*$^{\downarrow}$.

4.2 AdjointPDR$^{\downarrow}$ Simulates LT-PDR

The closest approach to AdjointPDR and AdjointPDR$^{\downarrow}$ is the lattice-theoretic extension of the original PDR, called LT-PDR [19]. While these algorithms exploit essentially the same positive chain to find an invariant, the main difference lies in the sequence used to witness the existence of some counterexamples.

[1] Up to a suitable renaming: the domain is $(L^{\downarrow}, \subseteq)$ instead of $(L, \sqsubseteq)$, the parameters are $\bot^{\downarrow}, b^{\downarrow}, b_r^{\downarrow}, p^{\downarrow}$ instead of i, f, g, p and the negative sequence is $\boldsymbol{Y}$ instead of $\boldsymbol{y}$.

Definition 16 (Kleene sequence, from [19]). *A sequence $\boldsymbol{c} = c_k, \dots, c_{n-1}$ of elements of L is a* Kleene sequence *if the conditions* (C1) *and* (C2) *below hold. It is* conclusive *if also condition* (C0) *holds.*

$$(\text{C0})\ c_1 \sqsubseteq b(\bot), \qquad (\text{C1})\ c_{n-1} \not\sqsubseteq p, \qquad (\text{C2})\ \forall j \in [k, n-2].\ c_{j+1} \sqsubseteq b(c_j).$$

LT-PDR tries to construct an under-approximation c_{n-1} of $b^{n-2}(\bot)$ that violates the property p. The Kleene sequence is constructed by trial and error, starting by some arbitrary choice of c_{n-1}.

AdjointPDR crucially differs from LT-PDR in the search for counterexamples: LT-PDR under-approximates the final chain while AdjointPDR over-approximates it. The algorithms are thus incomparable. However, we can draw a formal correspondence between AdjointPDR$^\downarrow$ and LT-PDR by showing that AdjointPDR$^\downarrow$ simulates LT-PDR, but cannot be simulated by LT-PDR. In fact, AdjointPDR$^\downarrow$ exploits the existence of the adjoint to start from an over-approximation Y_{n-1} of $p^\downarrow$ and computes backward an over-approximation of the set of safe states. Thus, the key difference comes from the strategy to look for a counterexample: to prove $\mu b \not\sqsubseteq p$, AdjointPDR$^\downarrow$ tries to find Y_{n-1} satisfying $p \in Y_{n-1}$ and $\mu b \notin Y_{n-1}$ while LT-PDR tries to find c_{n-1} s.t. $c_{n-1} \not\sqsubseteq p$ and $c_{n-1} \sqsubseteq \mu b$.

Theorem 17 below states that any execution of LT-PDR can be mimicked by AdjointPDR$^\downarrow$. The proof exploits a map from LT-PDR's Kleene sequences $\boldsymbol{c}$ to AdjointPDR$^\downarrow$'s negative sequences $\boldsymbol{neg(c)}$ of a particular form. Let $(L^\uparrow, \supseteq)$ be the complete lattice of upper sets, namely subsets $X \subseteq L$ such that $X = X^\uparrow \overset{\text{def}}{=} \{x' \in L \mid \exists x \in X \,.\, x \sqsubseteq x'\}$. There is an isomorphism $\neg \colon (L^\uparrow, \supseteq) \xleftrightarrow{\cong} (L^\downarrow, \subseteq)$ mapping each $X \subseteq S$ into its complement. For a Kleene sequence $\boldsymbol{c} = c_k, \dots, c_{n-1}$ of LT-PDR, the sequence $\boldsymbol{neg(c)} \overset{\text{def}}{=} \neg(\{c_k\}^\uparrow), \dots, \neg(\{c_{n-1}\}^\uparrow)$ is a negative sequence, in the sense of Definition 3, for AdjointPDR$^\downarrow$. Most importantly, the assignment $\boldsymbol{c} \mapsto \boldsymbol{neg(c)}$ extends to a function, from the states of LT-PDR to those of AdjointPDR$^\downarrow$, that is proved to be a *strong simulation* [24].

Theorem 17. AdjointPDR$^\downarrow$ *simulates LT-PDR.*

Remarkably, AdjointPDR$^\downarrow$'s negative sequences are not limited to the images of LT-PDR's Kleene sequences: they are more general than the complement of the upper closure of a singleton. In fact, a single negative sequence of AdjointPDR$^\downarrow$ can represent *multiple* Kleene sequences of LT-PDR at once. Intuitively, this means that a single execution of AdjointPDR$^\downarrow$ can correspond to multiple runs of LT-PDR. We can make this formal by means of the following result.

Proposition 18. *Let $\{\boldsymbol{c^m}\}_{m \in M}$ be a family of Kleene sequences. Then its pointwise intersection $\bigcap_{m \in M} \boldsymbol{neg(c^m)}$ is a negative sequence.*

The above intersection is pointwise in the sense that, for all $j \in [k, n-1]$, it holds $(\bigcap_{m \in M} \boldsymbol{neg(c^m)})_j \overset{\text{def}}{=} \bigcap_{m \in M} (\boldsymbol{neg(c^m)})_j = \neg(\{c^m_j \mid m \in M\}^\uparrow)$: intuitively, this is (up to $\boldsymbol{neg(\cdot)}$) a set containing all the M counterexamples. Note that, if the negative sequence of `AdjointPDR`$^{\downarrow}$ makes (A3) hold as an equality, as it is possible with any simple heuristic (see Proposition 10), then its complement contains *all* Kleene sequences possibly computed by LT-PDR.

Proposition 19. *Let $\boldsymbol{c}$ be a Kleene sequence and $\boldsymbol{Y}$ be the negative sequence s.t. $Y_j = (b_r^{\downarrow})^{n-1-j}(p^{\downarrow})$ for all $j \in [k, n-1]$. Then $c_j \in \neg(Y_j)$ for all $j \in [k, n-1]$.*

While the previous result suggests that simple heuristics are always the best in theory, as they can carry all counterexamples, this is often not the case in practice, since they might be computationally hard and outperformed by some smart over-approximations. An example is given by (6) in the next section.

5 Instantiating `AdjointPDR`$^{\downarrow}$ for MDPs

In this section we illustrate how to use `AdjointPDR`$^{\downarrow}$ to address the max reachability problem [1] for Markov Decision Processes.

A *Markov Decision Process* (MDP) is a tuple (A, S, s_ι, δ) where A is a set of labels, S is a set of states, $s_\iota \in S$ is an initial state, and $\delta\colon S \times A \to \mathcal{D}S + 1$ is a transition function. Here $\mathcal{D}S$ is the set of probability distributions over S, namely functions $d\colon S \to [0,1]$ such that $\sum_{s \in S} d(s) = 1$, and $\mathcal{D}S + 1$ is the disjoint union of $\mathcal{D}S$ and $1 = \{*\}$. The transition function δ assigns to every label $a \in A$ and to every state $s \in S$ either a distribution of states or $* \in 1$. We assume that both S and A are finite sets and that the set $Act(s) \overset{\text{def}}{=} \{a \in A \mid \delta(s,a) \neq *\}$ of actions enabled at s is non-empty for all states.

Intuitively, the *max reachability problem* requires to check whether the probability of reaching some bad states $\beta \subseteq S$ is less than or equal to a given threshold $\lambda \in [0,1]$. Formally, it can be expressed in lattice theoretic terms, by considering the lattice $([0,1]^S, \leq)$ of all functions $d\colon S \to [0,1]$, often called frames, ordered pointwise. The max reachability problem consists in checking $\mu b \leq p$ for $p \in [0,1]^S$ and $b\colon [0,1]^S \to [0,1]^S$, defined for all $d \in [0,1]^S$ and $s \in S$, as

$$p(s) \overset{\text{def}}{=} \begin{cases} \lambda & \text{if } s = s_\iota, \\ 1 & \text{if } s \neq s_\iota, \end{cases} \qquad b(d)(s) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } s \in \beta, \\ \max\limits_{a \in Act(s)} \sum\limits_{s' \in S} d(s') \cdot \delta(s,a)(s') & \text{if } s \notin \beta. \end{cases}$$

The reader is referred to [1] for all details.

Since b is not of the form $f \sqcup i$ for a left adjoint f (see e.g. [19]), rather than using `AdjointPDR`, one can exploit `AdjointPDR`$^{\downarrow}$. Beyond the simple initial heuristic, which is always applicable and enjoys negative termination, we illustrate now two additional heuristics that are experimentally tested in Sect. 6.

The two novel heuristics make the same choices in (Candidate) and (Decide). They exploit functions $\alpha\colon S \to A$, also known as memoryless schedulers, and the function $b_\alpha\colon [0,1]^S \to [0,1]^S$ defined for all $d \in [0,1]^S$ and $s \in S$ as follows:

$$b_\alpha(d)(s) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } s \in \beta, \\ \sum_{s' \in S} d(s') \cdot \delta(s, \alpha(s))(s') & \text{otherwise.} \end{cases}$$

Since for all $D \in ([0,1]^S)^{\downarrow}$, $b_r^{\downarrow}(D) = \{d \mid b(d) \in D\} = \bigcap_{\alpha}\{d \mid b_{\alpha}(d) \in D\}$ and since $\texttt{AdjointPDR}^{\downarrow}$ executes (Decide) only when $b(x_{k-1}) \notin Y_k$, there should exist some α such that $b_{\alpha}(x_{k-1}) \notin Y_k$. One can thus fix

$$(\boldsymbol{x}\|\boldsymbol{Y})_{n,k} \mapsto \begin{cases} p^{\downarrow} & \text{if } (\boldsymbol{x}\|\boldsymbol{Y})_{n,k} \overset{Ca}{\rightarrow} \\ \{d \mid b_{\alpha}(d) \in Y_k\} & \text{if } (\boldsymbol{x}\|\boldsymbol{Y})_{n,k} \overset{D}{\rightarrow} \end{cases} \tag{6}$$

Intuitively, such choices are smart refinements of those in (3): for (Candidate) they are exactly the same; for (Decide) rather than taking $b_r^{\downarrow}(Y_k)$, we consider a larger lower-set determined by the labels chosen by α. This allows to represent each Y_j as a set of $d \in [0,1]^S$ satisfying a *single* linear inequality, while using $b_r^{\downarrow}(Y_k)$ would yield a systems of possibly exponentially many inequalities (see Example 21 below). Moreover, from Theorem 12, it follows that such choices ensures negative termination.

Corollary 20. *Let h be a legit heuristic defined for (Candidate) and (Decide) as in (6). If $\mu b \not\leq p$, then $\texttt{AdjointPDR}^{\downarrow}{}_h$ terminates.*

Example 21. Consider the maximum reachability problem with threshold $\lambda = \frac{1}{4}$ and $\beta = \{s_3\}$ for the following MDP on alphabet $A = \{a, b\}$ and $s_{\iota} = s_0$.

$$\begin{tikzcd}
s_2 \arrow[r, bend left, "{b,1}"] & s_0 \arrow[l, bend left, "{a,\frac{1}{2}\ b,\frac{2}{3}}"] \arrow[loop below, "{b,\frac{1}{3}}"] \arrow[r, bend right, "{a,\frac{1}{2}}"'] & s_1 \arrow[l, bend right, "{a,\frac{1}{2}}"'] \arrow[r, "{a,\frac{1}{2}}"] & s_3 \arrow[loop right, "{a,1}"]
\end{tikzcd},$$

Hereafter we write $d \in [0,1]^S$ as column vectors with four entries $v_0 \dots v_3$ and we will use $\cdot$ for the usual matrix multiplication. With this notation, the lower set $p^{\downarrow} \in ([0,1]^S)^{\downarrow}$ and $b\colon [0,1]^S \to [0,1]^S$ can be written as

$$p^{\downarrow} = \{\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \mid \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \leq \begin{bmatrix} \frac{1}{4} \end{bmatrix}\} \quad \text{and} \quad b(\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}) = \begin{bmatrix} \max(\frac{v_1+v_2}{2}, \frac{v_0+2v_2}{3}) \\ \frac{v_0+v_3}{2} \\ v_0 \\ 1 \end{bmatrix}.$$

Amongst the several memoryless schedulers, only two are relevant for us: $\zeta \overset{\text{def}}{=} (s_0 \mapsto a,\ s_1 \mapsto a,\ s_2 \mapsto b,\ s_3 \mapsto a)$ and $\xi \overset{\text{def}}{=} (s_0 \mapsto b,\ s_1 \mapsto a,\ s_2 \mapsto b,\ s_3 \mapsto a)$. By using the definition of $b_{\alpha}\colon [0,1]^S \to [0,1]^S$, we have that

$$b_{\zeta}(\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}) = \begin{bmatrix} \frac{v_1+v_2}{2} \\ \frac{v_0+v_3}{2} \\ v_0 \\ 1 \end{bmatrix} \quad \text{and} \quad b_{\xi}(\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}) = \begin{bmatrix} \frac{v_0+2v_2}{3} \\ \frac{v_0+v_3}{2} \\ v_0 \\ 1 \end{bmatrix}.$$

It is immediate to see that the problem has negative answer, since using ζ in 4 steps or less, s_0 can reach s_3 already with probability $\frac{1}{4} + \frac{1}{8}$.

To illustrate the advantages of (6), we run $\texttt{AdjointPDR}^{\downarrow}$ with the simple initial heuristic and with the heuristic that only differs for the choice in (Decide), taken as in (6). For both heuristics, the first iterations are the same: several

$$\begin{array}{l|c|c}
\mathcal{F}^0 \overset{\text{def}}{=} & \{\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \mid \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \le \begin{bmatrix} \frac{1}{4} \end{bmatrix}\} & \{\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \mid \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \le \begin{bmatrix} \frac{1}{4} \end{bmatrix}\} \\
\mathcal{F}^1 \overset{\text{def}}{=} & \{\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \mid \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 2 & 0 \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \le \begin{bmatrix} \frac{1}{2} \\ \frac{3}{4} \end{bmatrix}\} & \{\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \mid \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \le \begin{bmatrix} \frac{1}{4} \end{bmatrix}\} \\
\mathcal{F}^2 \overset{\text{def}}{=} & \{\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \mid \begin{bmatrix} 3 & 0 & 0 & 1 \\ 2 & 1 & 1 & 0 \\ 4 & 0 & 2 & 0 \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \le \begin{bmatrix} 1 \\ \frac{3}{2} \\ \frac{9}{4} \end{bmatrix}\} & \{\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \mid \begin{bmatrix} \frac{3}{4} & 0 & 0 & \frac{1}{4} \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \le \begin{bmatrix} \frac{1}{4} \end{bmatrix}\} \\
\mathcal{F}^3 \overset{\text{def}}{=} & \{\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \mid \begin{bmatrix} 0 & \frac{3}{2} & \frac{3}{2} & 0 \\ 1 & 0 & 2 & 0 \\ \frac{3}{2} & 1 & 1 & \frac{1}{2} \\ \frac{13}{6} & 0 & \frac{4}{3} & \frac{1}{2} \\ 2 & 2 & 2 & 0 \\ \frac{10}{3} & 0 & \frac{8}{3} & 0 \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \le \begin{bmatrix} 0 \\ 0 \\ \frac{3}{2} \\ \frac{3}{2} \\ \frac{9}{4} \\ \frac{9}{4} \end{bmatrix}\} & \{\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \mid \begin{bmatrix} 0 & \frac{3}{8} & \frac{3}{8} & 0 \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \le \begin{bmatrix} 0 \end{bmatrix}\} \\
\mathcal{F}^4 \overset{\text{def}}{=} & \{\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \mid \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \le \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}\} = \{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}\} & \{\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \mid \begin{bmatrix} \frac{9}{16} & 0 & 0 & \frac{3}{16} \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \le \begin{bmatrix} 0 \end{bmatrix}\} \\
\mathcal{F}^5 \overset{\text{def}}{=} & \emptyset & \emptyset
\end{array}$$

Fig. 5. The elements of the negative sequences computed by $\texttt{AdjointPDR}^{\downarrow}$ for the MDP in Example 21. In the central column, these elements are computed by means of the simple initial heuristics, that is $\mathcal{F}^i = (b_r^{\downarrow})^i(p^{\downarrow})$. In the rightmost column, these elements are computed using the heuristic in (6). In particular $\mathcal{F}^i = \{d \mid b_\zeta(d) \in \mathcal{F}^{i-1}\}$ for $i \le 3$, while for $i \ge 4$ these are computed as $\mathcal{F}^i = \{d \mid b_\xi(d) \in \mathcal{F}^{i-1}\}$.

repetitions of (Candidate), (Conflict) and (Unfold) exploiting elements of the positive chain that form the initial chain (except for the last element x_{n-1}).

$$(\emptyset \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \| \varepsilon)_{3,3} \xrightarrow{Ca} \xrightarrow{Co} (\emptyset \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \| \varepsilon)_{3,3} \xrightarrow{U} \xrightarrow{Ca} \xrightarrow{Co} \xrightarrow{U} \xrightarrow{Ca} \xrightarrow{Co} \xrightarrow{U} \xrightarrow{Ca} \xrightarrow{Co} \xrightarrow{U} \xrightarrow{Ca} (\emptyset \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ \frac{1}{2} \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \frac{1}{4} \\ \frac{1}{2} \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \frac{1}{4} \\ \frac{5}{8} \\ \frac{1}{4} \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \| p^{\downarrow})_{7,6}.$$

In the latter state the algorithm has to perform (Decide), since $b(x_5) \notin p^{\downarrow}$. Now the choice of z in (Decide) is different for the two heuristics: the former uses $b_r^{\downarrow}(p^{\downarrow}) = \{d \mid b(d) \in p^{\downarrow}\}$, the latter uses $\{d \mid b_\zeta(d) \in p^{\downarrow}\}$. Despite the different choices, both the heuristics proceed with 6 steps of (Decide):

$$(\emptyset \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ \frac{1}{2} \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \frac{1}{4} \\ \frac{1}{2} \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \frac{1}{4} \\ \frac{5}{8} \\ \frac{1}{4} \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \| \mathcal{F}^0)_{7,6} \xrightarrow{D} \xrightarrow{D} \xrightarrow{D} \xrightarrow{D} \xrightarrow{D} (\emptyset \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ \frac{1}{2} \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \frac{1}{4} \\ \frac{1}{2} \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \frac{1}{4} \\ \frac{5}{8} \\ \frac{1}{4} \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \| \mathcal{F}^5, \mathcal{F}^4, \mathcal{F}^3, \mathcal{F}^2, \mathcal{F}^1, \mathcal{F}^0)_{7,1}$$

The element of the negative sequence $\mathcal{F}^i$ are illustrated in Fig. 5 for both the heuristics. In both cases, $\mathcal{F}^5 = \emptyset$ and thus $\texttt{AdjointPDR}^{\downarrow}$ returns false.

To appreciate the advantages provided by (6), it is enough to compare the two columns for the $\mathcal{F}^i$ in Fig. 5: in the central column, the number of inequalities defining $\mathcal{F}^i$ significantly grows, while in the rightmost column is always 1.

Whenever Y_k is generated by a single linear inequality, we observe that $Y_k = \{d \in [0,1]^S \mid \sum_{s \in S}(r_s \cdot d(s)) \le r\}$ for suitable non-negative real numbers r and r_s for all $s \in S$. The convex set Y_k is generated by finitely many $d \in [0,1]^S$ enjoying a convenient property: $d(s)$ is different from 0 and 1 only for at most one $s \in S$. The set of its generators, denoted by $\mathcal{G}_k$, can thus be easily computed.

We exploit this property to resolve the choice for (Conflict). We consider its sub set $\mathcal{Z}_k \overset{\text{def}}{=} \{d \in \mathcal{G}_k \mid b(x_{k-1}) \leq d\}$ and define $z_B, z_{01} \in [0,1]^S$ for all $s \in S$ as

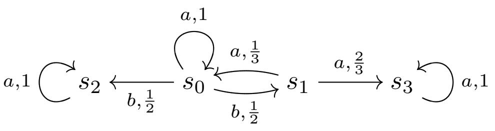
$$z_B(s) \overset{\text{def}}{=} \begin{cases} (\bigwedge \mathcal{Z}_k)(s) & \text{if } r_s \neq 0, \mathcal{Z}_k \neq \emptyset \\ b(x_{k-1})(s) & \text{otherwise} \end{cases} \quad z_{01}(s) \overset{\text{def}}{=} \begin{cases} \lceil z_B(s) \rceil & \text{if } r_s = 0, \mathcal{Z}_k \neq \emptyset \\ z_B(s) & \text{otherwise} \end{cases} \tag{7}$$

where, for $u \in [0,1]$, $\lceil u \rceil$ denotes 0 if $u = 0$ and 1 otherwise. We call `hCoB` and `hCo01` the heuristics defined as in (6) for (Candidate) and (Decide) and as z_B, respectively z_{01}, for (Conflict). The heuristics `hCo01` can be seen as a Boolean modification of `hCoB`, rounding up positive values to 1 to accelerate convergence.

Proposition 22. *The heuristics* `hCoB` *and* `hCo01` *are legit.*

By Corollary 20, `AdjointPDR`$^{\downarrow}$ terminates for negative answers with both `hCoB` and `hCo01`. We conclude this section with a last example.

Example 23. Consider the following MDP with alphabet $A = \{a, b\}$ and $s_\iota = s_0$



$$\begin{tikzcd}
s_2 \arrow[loop left, "{a,1}"] & s_0 \arrow[l, "{b,\frac{1}{2}}"'] \arrow[loop above, "{a,1}"] \arrow[r, bend right, "{b,\frac{1}{2}}"'] & s_1 \arrow[l, bend right, "{a,\frac{1}{3}}"'] \arrow[r, "{a,\frac{2}{3}}"] & s_3 \arrow[loop right, "{a,1}"]
\end{tikzcd}$$

and the max reachability problem with threshold $\lambda = \frac{2}{5}$ and $\beta = \{s_3\}$. The lower set $p^{\downarrow} \in ([0,1]^S)^{\downarrow}$ and $b\colon [0,1]^S \to [0,1]^S$ can be written as

$$p^{\downarrow} = \{\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \mid \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \leq \begin{bmatrix} \frac{2}{5} \end{bmatrix}\} \quad \text{and} \quad b(\begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}) = \begin{bmatrix} \max(v_0, \frac{v_1+v_2}{2}) \\ \frac{v_0+2\cdot v_3}{3} \\ v_2 \\ 1 \end{bmatrix}$$

With the simple initial heuristic, `AdjointPDR`$^{\downarrow}$ does not terminate. With the heuristic `hCo01`, it returns true in 14 steps, while with `hCoB` in 8. The first 4 steps, common to both `hCoB` and `hCo01`, are illustrated below.

$$(\emptyset \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \| \varepsilon)_{3,3} \overset{Ca}{\to} (\emptyset \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \| p^{\downarrow})_{3,2} \overset{Co}{\to} (\emptyset \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} \frac{2}{5} \\ 0 \\ 0 \\ 1 \end{bmatrix} \| \varepsilon)_{3,3} \qquad b(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \mathcal{Z}_2 = \{\begin{bmatrix} \frac{2}{5} \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} \frac{2}{5} \\ 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} \frac{2}{5} \\ 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} \frac{2}{5} \\ 1 \\ 1 \\ 1 \end{bmatrix}\}$$

$$\overset{U}{\to}\overset{Ca}{\to}(\emptyset \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} \frac{2}{5} \\ 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \| p^{\downarrow})_{4,3} \overset{Co}{\to} \left| (\emptyset \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} \frac{2}{5} \\ 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \frac{2}{5} \\ 1 \\ 0 \\ 1 \end{bmatrix} \| \varepsilon)_{4,4} \right| (\emptyset \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} \frac{2}{5} \\ 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \frac{2}{5} \\ \frac{4}{5} \\ 0 \\ 1 \end{bmatrix} \| \varepsilon)_{4,4} \qquad b(\begin{bmatrix} \frac{2}{5} \\ 0 \\ 0 \\ 1 \end{bmatrix}) = \begin{bmatrix} \frac{2}{5} \\ \frac{4}{5} \\ 0 \\ 1 \end{bmatrix} \quad \mathcal{Z}_3 = \{\begin{bmatrix} \frac{2}{5} \\ 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} \frac{2}{5} \\ 1 \\ 1 \\ 1 \end{bmatrix}\}$$

Observe that in the first (Conflict) $z_B = z_{01}$, while in the second $z_{01}(s_1) = 1$ and $z_B(s_1) = \frac{4}{5}$, leading to the two different states prefixed by vertical lines.

6 Implementation and Experiments

We first developed, using Haskell and exploiting its abstraction features, a common template that accommodates both `AdjointPDR` and `AdjointPDR`$^{\downarrow}$. It is a program parametrized by two lattices—used for positive chains and negative sequences, respectively—and by a heuristic.

For our experiments, we instantiated the template to AdjointPDR$^{\downarrow}$ for MDPs (letting $L = [0,1]^S$), with three different heuristics: hCoB and hCo01 from Proposition 22; and hCoS introduced below. Besides the template ($\sim$100 lines), we needed $\sim$140 lines to account for hCoB and hCo01, and additional $\sim$100 lines to further obtain hCoS. All this indicates a clear benefit of our abstract theory: a general template can itself be coded succinctly; instantiation to concrete problems is easy, too, thanks to an explicitly specified interface of heuristics.

Our implementation accepts MDPs expressed in a symbolic format inspired by Prism models [20], in which states are variable valuations and transitions are described by symbolic functions (they can be segmented with symbolic guards $\{\text{guard}_i\}_i$). We use rational arithmetic (Rational in Haskell) for probabilities to limit the impact of rounding errors.

Heuristics. The three heuristics (hCoB, hCo01, hCoS) use the same choices in (Candidate) and (Decide), as defined in (6), but different ones in (Conflict).

The third heuristics hCoS is a *symbolic* variant of hCoB; it relies on our symbolic model format. It uses z_S for z in (Conflict), where $z_S(s) = z_B(s)$ if $r_s \neq 0$ or $\mathscr{Z}_k = \emptyset$. The definition of $z_S(s)$ otherwise is notable: we use a piecewise affine function $(t_i \cdot s + u_i)_i$ for $z_S(s)$, where the affine functions $(t_i \cdot s + u_i)_i$ are guarded by the same guards $\{\text{guard}_i\}_i$ of the MDP's transition function. We let the SMT solver Z3 [25] search for the values of the coefficients t_i, u_i, so that z_S satisfies the requirements of (Conflict) (namely $b(x_{k-1})(s) \leq z_S(s) \leq 1$ for each $s \in S$ with $r_s = 0$), together with the condition $b(z_S) \leq z_S$ for faster convergence. If the search is unsuccessful, we give up hCoS and fall back on the heuristic hCoB.

As a task common to the three heuristics, we need to calculate $\mathscr{Z}_k = \{d \in \mathcal{G}_k \mid b(x_{k-1}) \leq d\}$ in (Conflict) (see (7)). Rather than computing the whole set $\mathcal{G}_k$ of generating points of the linear inequality that defines Y_k, we implemented an ad-hoc algorithm that crucially exploits the condition $b(x_{k-1}) \leq d$ for pruning.

Experiment Settings. We conducted the experiments on Ubuntu 18.04 and AWS t2.xlarge (4 CPUs, 16 GB memory, up to 3.0 GHz Intel Scalable Processor). We used several Markov chain (MC) benchmarks and a couple of MDP ones.

Research Questions. We wish to address the following questions.

RQ1 Does AdjointPDR$^{\downarrow}$ advance the state-of-the-art performance of *PDR* algorithms for probabilistic model checking?
RQ2 How does AdjointPDR$^{\downarrow}$'s performance compare against *non-PDR* algorithms for probabilistic model checking?
RQ3 Does the theoretical framework of AdjointPDR$^{\downarrow}$ successfully guide the discovery of various heuristics with practical performance?
RQ4 Does AdjointPDR$^{\downarrow}$ successfully manage nondeterminism in MDPs (that is absent in MCs)?

Experiments on MCs (Table 1). We used six benchmarks: Haddad-Monmege is from [17]; the others are from [3,19]. We compared AdjointPDR$^{\downarrow}$ (with three

Table 1. Experimental results on MC benchmarks. $|S|$ is the number of states, P is the reachability probability (calculated by manual inspection), λ is the threshold in the problem $P \leq_? \lambda$ (shaded if the answer is no). The other columns show the average execution time in seconds; TO is timeout (900 s); MO is out-of-memory. For `AdjointPDR`$^{\downarrow}$ and LT-PDR we used the `tasty-bench` Haskell package and repeated executions until std. dev. is $< 5\%$ (at least three execs). For PrIC3 and Storm, we made five executions. Storm's execution does not depend on λ: it seems to answer queries of the form $P \leq_? \lambda$ by calculating P. We observed a wrong answer for the entry with (†) (Storm, sp.-num., Haddad-Monmege); see the discussion of RQ2.

Benchmark	$\lvert S\rvert$	P	λ	`AdjointPDR`$^{\downarrow}$			LT-PDR	PrIC3				Storm		
				`hCoB`	`hCo01`	`hCoS`		none	lin.	pol.	hyb.	sp.-num.	sp.-rat.	sp.-sd.
Grid	10^2	0.033	0.3	0.013	0.022	0.659	0.343	1.383	23.301	MO	MO	0.010	0.010	0.010
			0.2	0.013	0.031	0.657	0.519	1.571	26.668	TO	MO			
	10^3	<0.001	0.3	1.156	2.187	5.633	126.441	TO	TO	TO	MO	0.010	0.017	0.011
			0.2	1.146	2.133	5.632	161.667	TO	TO	TO	MO			
BRP	10^3	0.035	0.1	12.909	7.969	55.788	TO	TO	TO	MO	MO	0.012	0.018	0.011
			0.01	1.977	8.111	5.645	21.078	60.738	626.052	524.373	823.082			
			0.005	0.604	2.261	2.709	1.429	12.171	254.000	197.940	318.840			
Zero-Conf	10^2	0.5	0.9	1.217	68.937	0.196	TO	19.765	136.491	0.630	0.468	0.010	0.018	0.011
			0.75	1.223	68.394	0.636	TO	19.782	132.780	0.602	0.467			
			0.52	1.228	60.024	0.739	TO	19.852	136.533	0.608	0.474			
			0.45	<0.001	0.001	0.001	<0.001	0.035	0.043	0.043	0.043			
	10^4	0.5	0.9	MO	TO	7.443	TO	TO	TO	0.602	0.465	0.037	262.193	0.031
			0.75	MO	TO	15.223	TO	TO	TO	0.599	0.470			
			0.52	MO	TO	TO	TO	TO	TO	0.488	0.475			
			0.45	0.108	0.119	0.169	0.016	0.035	0.040	0.040	0.040			
Chain	10^3	0.394	0.9	36.083	TO	0.478	TO	269.801	TO	0.938	0.686	0.010	0.014	0.011
			0.4	35.961	TO	394.955	TO	271.885	TO	0.920	TO			
			0.35	101.351	TO	454.892	435.199	238.613	TO	TO	TO			
			0.3	62.036	463.981	120.557	209.346	124.829	746.595	TO	TO			
Double-Chain	10^3	0.215	0.9	12.122	7.318	TO	TO	TO	TO	1.878	2.053	0.011	0.018	0.010
			0.3	12.120	20.424	TO	TO	TO	TO	1.953	2.058			
			0.216	12.096	19.540	TO	TO	TO	TO	172.170	TO			
			0.15	12.344	16.172	TO	16.963	TO	TO	TO	TO			
Haddad-Mon-mege	41	0.7	0.9	0.004	0.009	8.528	TO	1.188	31.915	TO	MO	0.011	0.011	1.560
			0.75	0.004	0.011	2.357	TO	1.209	32.143	TO	712.086			
	10^3	0.7	0.9	59.721	61.777	TO	TO	TO	TO	TO	TO	0.013 (†)	0.043	TO
			0.75	60.413	63.050	TO	TO	TO	TO	TO	TO			

heuristics) against LT-PDR [19], PrIC3 (with four heuristics *none*, *lin.*, *pol.*, *hyb.*, see [3]), and Storm 1.5 [11]. Storm is a recent comprehensive toolsuite that implements different algorithms and solvers. Among them, our comparison is against *sparse-numeric*, *sparse-rational*, and *sparse-sound*. The *sparse* engine uses explicit state space representation by sparse matrices; this is unlike another representative *dd* engine that uses symbolic BDDs. (We did not use *dd* since it often reported errors, and was overall slower than *sparse*.) *Sparse-numeric* is a value-iteration (VI) algorithm; *sparse-rational* solves linear (in)equations using rational arithmetic; *sparse-sound* is a sound VI algorithm [26].[2]

[2] There are another two sound algorithms in Storm: one that utilizes interval iteration [2] and the other does optimistic VI [16]. We have excluded them from the results since we observed that they returned incorrect answers.

Table 2. Experimental results on MDP benchmarks. The legend is the same as Table 1, except that P is now the maximum reachability probability.

Benchmark	$\lvert S\rvert$	P	λ	AdjointPDR$^{\downarrow}$			Storm		
				hCoB	hCo01	hCoS	sp.-num	sp.-rat.	sp.-sd.
CDrive2	38	0.865	0.9	MO	0.172	TO	0.019	0.019	0.018
			0.75	MO	0.058	TO			
			0.5	0.015	0.029	86.798			
TireWorld	8670	0.233	0.9	MO	3.346	TO	0.070	0.164	0.069
			0.75	MO	3.337	TO			
			0.5	MO	6.928	TO			
			0.2	4.246	24.538	TO			

Experiments on MDPs (Table 2). We used two benchmarks from [17]. We compared AdjointPDR$^{\downarrow}$ only against Storm, since RQ1 is already addressed using MCs (besides, PrIC3 did not run for MDPs).

Discussion. The experimental results suggest the following answers to the RQs.

RQ1. The performance advantage of AdjointPDR$^{\downarrow}$, over both LT-PDR and PrIC3, was clearly observed throughout the benchmarks. AdjointPDR$^{\downarrow}$ outperformed LT-PDR, thus confirming empirically the theoretical observation in Sect. 4.2. The profit is particularly evident in those instances whose answer is positive. AdjointPDR$^{\downarrow}$ generally outperformed PrIC3, too. Exceptions are in ZeroConf, Chain and DoubleChain, where PrIC3 with polynomial (pol.) and hybrid (hyb.) heuristics performs well. This seems to be thanks to the expressivity of the polynomial template in PrIC3, which is a possible enhancement we are yet to implement (currently our symbolic heuristic hCoS uses only the affine template).

RQ2. The comparison with Storm is interesting. Note first that Storm's *sparse-numeric* algorithm is a VI algorithm that gives a guaranteed lower bound *without guaranteed convergence*. Therefore its positive answer to $P \leq_{?} \lambda$ may not be correct. Indeed, for Haddad-Monmege with $\lvert S\rvert \sim 10^3$, it answered $P = 0.5$ which is wrong (($\dagger$) in Table 1). This is in contrast with PDR algorithms that discovers an explicit witness for $P \leq \lambda$ via their positive chain.

Storm's *sparse-rational* algorithm is precise. It was faster than PDR algorithms in many benchmarks, although AdjointPDR$^{\downarrow}$ was better or comparable in ZeroConf (10^4) and Haddad-Monmege (41), for λ such that $P \leq \lambda$ is true. We believe this suggests a general advantage of PDR algorithms, namely to accelerate the search for an invariant-like witness for safety.

Storm's *sparse-sound* algorithm is a sound VI algorithm that returns correct answers aside numerical errors. Its performance was similar to that of sparse-numeric, except for the two instances of Haddad-Monmege: sparse-sound returned correct answers but was much slower than sparse-numeric. For these two instances, `AdjointPDR`$^{\downarrow}$ outperformed sparse-sound.

It seems that a big part of Storm's good performance is attributed to the sparsity of state representation. This is notable in the comparison of the two instances of Haddad-Monmege (41 vs. 10^3): while Storm handles both of them easily, `AdjointPDR`$^{\downarrow}$ struggles a bit in the bigger instance. Our implementation can be extended to use sparse representation, too; this is future work.

RQ3. We derived the three heuristics (`hCoB`, `hCo01`, `hCoS`) exploiting the theory of `AdjointPDR`$^{\downarrow}$. The experiments show that each heuristic has its own strength. For example, `hCo01` is slower than `hCoB` for MCs, but it is much better for MDPs. In general, there is no silver bullet heuristic, so coming up with a variety of them is important. The experiments suggest that our theory of `AdjointPDR`$^{\downarrow}$ provides great help in doing so.

RQ4. Table 2 shows that `AdjointPDR`$^{\downarrow}$ can handle nondeterminism well: once a suitable heuristic is chosen, its performances on MDPs and on MCs of similar size are comparable. It is also interesting that better-performing heuristics vary, as we discussed above.

Summary. `AdjointPDR`$^{\downarrow}$ clearly outperforms existing probabilistic PDR algorithms in many benchmarks. It also compares well with Storm—a highly sophisticated toolsuite—in a couple of benchmarks. These are notable especially given that `AdjointPDR`$^{\downarrow}$ currently lacks enhancing features such as richer symbolic templates and sparse representation (adding which is future work). Overall, we believe that `AdjointPDR`$^{\downarrow}$ *confirms the potential of PDR algorithms in probabilistic model checking.* Through the three heuristics, we also observed the value of an abstract general theory in devising heuristics in PDR, which is probably true of verification algorithms in general besides PDR.

References

1. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press (2008)
2. Baier, C., Klein, J., Leuschner, L., Parker, D., Wunderlich, S.: Ensuring the reliability of your model checker: interval iteration for Markov decision processes. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 160–180. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_8
3. Batz, K., et al.: PrIC3: property directed reachability for MDPs. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12225, pp. 512–538. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53291-8_27
4. Bonchi, F., Ganty, P., Giacobazzi, R., Pavlovic, D.: Sound up-to techniques and complete abstract domains. In: Dawar, A., Grädel, E. (eds.) Proceedings of LICS 2018, pp. 175–184. ACM (2018). https://doi.org/10.1145/3209108.3209169
5. Bradley, A.R.: SAT-based model checking without unrolling. In: Jhala, R., Schmidt, D. (eds.) VMCAI 2011. LNCS, vol. 6538, pp. 70–87. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-18275-4_7
6. Cimatti, A., Griggio, A.: Software model checking via IC3. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 277–293. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_23

7. Cousot, P.: Partial completeness of abstract fixpoint checking. In: Choueiry, B.Y., Walsh, T. (eds.) SARA 2000. LNCS (LNAI), vol. 1864, pp. 1–25. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44914-0_1
8. Cousot, P.: Principles of Abstract Interpretation. MIT Press (2021)
9. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of POPL 1977, pp. 238–252. ACM (1977). https://doi.org/10.1145/512950.512973
10. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order, 2nd Edn. Cambridge University Press (2002)
11. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A storm is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_31
12. Eén, N., Mishchenko, A., Brayton, R.K.: Efficient implementation of property directed reachability. In: Bjesse, P., Slobodová, A. (eds.) Proc. of FMCAD 2011. pp. 125–134. FMCAD Inc. (2011). http://dl.acm.org/citation.cfm?id=2157675
13. Feldman, Y.M.Y., Sagiv, M., Shoham, S., Wilcox, J.R.: Property-directed reachability as abstract interpretation in the monotone theory. Proc. ACM Program. Lang. **6**(POPL), 1–31 (2022). https://doi.org/10.1145/3498676
14. Giacobazzi, R., Ranzato, F., Scozzari, F.: Making abstract interpretations complete. J. ACM **47**(2), 361–416 (2000). https://doi.org/10.1145/333979.333989
15. Gurfinkel, A.: IC3, PDR, and friends (2015). https://arieg.bitbucket.io/pdf/gurfinkel_ssft15.pdf
16. Hartmanns, A., Kaminski, B.L.: Optimistic value iteration. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12225, pp. 488–511. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53291-8_26
17. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 344–350. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17462-0_20
18. Hoder, K., Bjørner, N.: Generalized property directed reachability. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 157–171. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31612-8_13
19. Kori, M., Urabe, N., Katsumata, S., Suenaga, K., Hasuo, I.: The lattice-theoretic essence of property directed reachability analysis. In: Shoham, S., Vizel, Y. (eds.) Proceedings of CAV 2022, Part I. Lecture Notes in Computer Science, vol. 13371, pp. 235–256. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-13185-1_12
20. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
21. Lange, T., Neuhäußer, M.R., Noll, T., Katoen, J.-P.: IC3 software model checking. Int. J. Softw. Tools Technol. Trans. **22**(2), 135–161 (2019). https://doi.org/10.1007/s10009-019-00547-x
22. Levy, P.B.: Call-By-Push-Value: A Functional/Imperative Synthesis, Semantics Structures in Computation, vol. 2. Springer, Dordrecht (2004). https://doi.org/10.1007/978-94-007-0954-6
23. MacLane, S.: Categories for the Working Mathematician. Graduate Texts in Mathematics, vol. 5. Springer-Verlag, New York (1971)

24. Milner, R.: Communication and Concurrency. Prentice-Hall Inc, USA (1989)
25. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
26. Quatmann, T., Katoen, J.-P.: Sound value iteration. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 643–661. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_37
27. Seufert, T., Scholl, C.: Sequential verification using reverse PDR. In: Große, D., Drechsler, R. (eds.) Proceedings of MBMV 2017, pp. 79–90. Shaker Verlag (2017)
28. Seufert, T., Scholl, C.: Combining PDR and reverse PDR for hardware model checking. In: Madsen, J., Coskun, A.K. (eds.) Proceedings of DATE 2018, pp. 49–54. IEEE (2018). https://doi.org/10.23919/DATE.2018.8341978
29. Seufert, T., Scholl, C.: fbPDR: In-depth combination of forward and backward analysis in property directed reachability. In: Teich, J., Fummi, F. (eds.) Proceedings of DATE 2019, pp. 456–461. IEEE (2019). https://doi.org/10.23919/DATE.2019.8714819
30. Suda, M.: Property directed reachability for automated planning. In: Chien, S.A., Do, M.B., Fern, A., Ruml, W. (eds.) Proceedings of ICAPS 2014. AAAI (2014). https://doi.org/10.1613/jair.4231