



PrIC3: Property Directed Reachability for MDPs

Kevin Batz¹ , Sebastian Junges² , Benjamin Lucien Kaminski³ ,
Joost-Pieter Katoen¹ , Christoph Matheja⁴ , and Philipp Schröder¹ 

¹ RWTH Aachen University, Aachen, Germany
kevin.batz@cs.rwth-aachen.de
² University of California, Berkeley, USA
³ University College London, London, UK
⁴ ETH Zürich, Zürich, Switzerland



Abstract. IC3 has been a leap forward in symbolic model checking. This paper proposes PrIC3 (pronounced pricy-three), a conservative extension of IC3 to symbolic model checking of MDPs. Our main focus is to develop the theory underlying PrIC3. Alongside, we present a first implementation of PrIC3 including the key ingredients from IC3 such as generalization, repushing, and propagation.

1 Introduction

IC3. Also known as property-directed reachability (PDR) [23], IC3 [13] is a symbolic approach for verifying finite transition systems (TSs) against safety properties like “*bad states are unreachable*”. It combines bounded model checking (BMC) [12] and inductive invariant generation. Put shortly, IC3 either proves that a set B of bad states is *unreachable* by finding a set of non- B states closed under reachability—called an *inductive invariant*—or refutes reachability of B by a *counterexample* path reaching B . Rather than unrolling the transition relation (as in BMC), IC3 attempts to incrementally strengthen the invariant “no state in B is reachable” into an inductive one. In addition, it applies aggressive abstraction to the explored state space, so-called generalization [36]. These aspects together with the enormous advances in modern SAT solvers have led to IC3’s success. IC3 has been extended [27, 38] and adapted to software verification [19, 44]. This paper develops a *quantitative* IC3 framework for probabilistic models.

MDPs. Markov decision processes (MDPs) extend TSs with discrete probabilistic choices. They are central in planning, AI as well as in modeling randomized distributed algorithms. A key question in verifying MDPs is *quantitative* reachability: “*is the (maximal) probability to reach B at most λ ?*”. Quantitative reachability [5, 6]

This work has been supported by the ERC Advanced Grant 787914 (FRAPPANT), NSF grants 1545126 (VeHiCaL) and 1646208, the DARPA Assured Autonomy program, Berkeley Deep Drive, and by Toyota under the iCyPhy center.

© The Author(s) 2020

S. K. Lahiri and C. Wang (Eds.): CAV 2020, LNCS 12225, pp. 512–538, 2020.

https://doi.org/10.1007/978-3-030-53291-8_27

reduces to solving linear programs (LPs). Various tools support MDP model checking, e.g., Prism [43], Storm [22], modest [34], and EPMC [31]. The LPs are mostly solved using (variants of) value iteration [8, 28, 35, 51]. Symbolic BDD-based MDP model checking originated two decades ago [4] and is rather successful.

Towards IC3 for MDPs. Despite the success of BDD-based symbolic methods in tools like Prism, IC3 has not penetrated probabilistic model checking yet. The success of IC3 and the importance of quantitative reachability in probabilistic model checking raises the question *whether and how IC3 can be adapted—not just utilized—to reason about quantitative reachability in MDPs*. This paper addresses the challenges of answering this question. It extends IC3 in several dimensions to overcome these hurdles, making PrIC3—to our knowledge—the *first IC3 framework for quantitative reachability in MDPs*¹. Notably, PrIC3 is conservative: For a threshold $\lambda = 0$, PrIC3 solves the same qualitative problem *and behaves (almost) the same as standard IC3*. Our main contribution is developing the theory underlying PrIC3, which is accompanied by a proof-of-concept implementation.

Challenge 1 (Leaving the Boolean domain). IC3 iteratively computes *frames*, which are over-approximations of sets of states that can reach B in a bounded number of steps. For MDPs, Boolean reachability becomes a *quantitative reachability probability*. This requires a shift: frames become real-valued functions rather than sets of states. Thus, there are infinitely many possible frames—even for finite-state MDPs—just as for infinite-state software [19, 44] and hybrid systems [54]. Additionally, whereas in TSs a state reachable within k steps remains reachable on increasing k , the reachability probability in MDPs may increase. This complicates ensuring termination of an IC3 algorithm for MDPs. \triangle

Challenge 2 (Counterexamples \neq single paths). For TSs, a single cycle-free path² to B suffices to refute that “ B is not reachable”. This is not true in the probabilistic setting [32]. Instead, proving that the probability of reaching B exceeds the threshold λ requires a *set of possibly cyclic paths*—e.g., represented as a sub-MDP [15]—whose probability mass exceeds λ . Handling sets of paths as counterexamples in the context of IC3 is new. \triangle

Challenge 3 (Strengthening). This key IC3 technique intuitively turns a proof obligation of type (I) “state s is unreachable from the initial state s_I ” into type (II) “ s ’s predecessors are unreachable from s_I ”. A first issue is that in the quantitative setting, the standard characterization of reachability probabilities in MDPs (the Bellman equations) inherently *reverses* the direction of reasoning (cf. “reverse” IC3 [53]): Hence, strengthening turns (i) “ s cannot reach B ” into (II) “ s ’s successors cannot reach B ”.

A much more challenging issue, however, is that in the quantitative setting obligations of type (i) read “ s is reachable *with at most probability* δ ”. However,

¹ Recently, (standard) IC3 for TSs was *utilized* in model checking Markov chains [49] to on-the-fly compute the states that cannot reach B .

² In [38], tree-like counterexamples are used for non-linear predicate transformers in IC3.

the strengthened type (II) obligation must then read: “*the weighted sum over the reachability probabilities of the successors of s is at most δ* ”. In general, there are infinitely many possible choices of subobligations for the successors of s in order to satisfy the original obligation, because—grossly simplified—there are infinitely many possibilities for a and b to satisfy weighted sums such as $\frac{1}{3}a + \frac{2}{3}b \leq \delta$. While we only need one choice of subobligations, picking a *good* one is approximately as hard as solving the entire problem altogether. We hence require a heuristic, which is guided by a *user-provided oracle*. \triangle

Challenge 4 (Generalization). “One of the key components of IC3 is [inductive] generalization” [13]. Generalization [36] abstracts single states. It makes IC3 scale, but is *not* essential for correctness. To facilitate generalization, systems should be encoded symbolically, i.e., integer-valued program variables describe states. Frames thus map variables to probabilities. A first aspect is how to effectively present them to an SMT-solver. Conceptually, we use uninterpreted functions and universal quantifiers (encoding program behavior) together with linear real arithmetic to encode the weighted sums occurring when reasoning about probabilities. A second aspect is more fundamental: Abstractly, IC3’s generalization guesses an unreachable set of states. We, however, need to guess this set *and* a probability for each state. To be effective, these guesses should moreover eventually yield an inductive frame, which is often highly nonlinear. We propose three SMT-guided interpolation variants for guessing these maps. \triangle

Structure of this Paper. We develop PrIC3 gradually: We explain the underlying rationale in Sect. 3. We also describe the core of PrIC3—called PrIC3 \mathcal{H} —which resembles closely the main loop of standard IC3, but uses adapted frames and termination criteria (Challenge 1). In line with Challenge 3, PrIC3 \mathcal{H} is parameterized by a heuristic \mathcal{H} which is applied whenever we need to select one out of infinitely many probabilities. No requirements on the quality of \mathcal{H} are imposed. PrIC3 \mathcal{H} is *sound* and always terminates: If it returns `true`, then the maximal reachability probability is bounded by λ . Without additional assumptions about \mathcal{H} , PrIC3 \mathcal{H} is *incomplete*: on returning `false`, it is unknown whether the returned sub-MDP is indeed a counterexample (Challenge 2). Section 4 details strengthening (Challenge 3). Section 5 presents a sound *and* complete algorithm PrIC3 on top of PrIC3 \mathcal{H} . Section 6 presents a prototype, discusses our chosen heuristics, and addresses Challenge 4. Section 7 shows some encouraging experiments, but also illustrates need for further progress.

Related Work. Just like IC3 has been a symbiosis of different approaches, PrIC3 has been inspired by several existing techniques from the verification of probabilistic systems.

BMC. Adaptions of BMC to Markov chains (MCs) with a dedicated treatment of cycles have been pursued in [57]. The encoding in [24] annotates sub-formulae with probabilities. The integrated SAT solving process implicitly unrolls all paths leading to an exponential blow-up. In [52], this is circumvented by grouping paths, discretizing them, and using an encoding with quantifiers and bit-vectors,

but without numerical values. Recently, [56] extends this idea to a PAC algorithm by purely propositional encodings and (approximate) model counting [17]. These approaches focus on MCs and are not mature yet.

Invariant Synthesis. Quantitative loop invariants are key in analyzing *probabilistic programs* whose operational semantics are (possibly infinite) MDPs [26]. A quantitative invariant I maps states to probabilities. I is shown to be an invariant by comparing I to the result of applying the MDP’s Bellman operator to I . Existing approaches for invariant synthesis are, e.g., based on weakest pre-expectations [33, 39, 40, 42, 46], template-based constraint solving [25], notions of martingales [3, 9, 16, 55], and solving recurrence relations [10]. All but the last technique require user guidance.

Abstraction. To combat state-space explosion, abstraction is often employed. CEGAR for MDPs [37] deals with explicit sets of paths as counterexamples. Game-based abstraction [30, 41] and partial exploration [14] exploit that not all paths have to be explored to prove bounds on reachability probabilities.

Statistical Methods and (deep) Reinforcement Learning. Finally, an avenue that avoids storing a (complete) model are simulation-based approaches (statistical model checking [2]) and variants of reinforcement learning, possibly with neural networks. For MDPs, these approaches yield weak statistical guarantees [20], but may provide good oracles.

2 Problem Statement

Our aim is to prove that the *maximal probability of reaching* a set B of *bad states* from the initial state s_I of a *Markov decision process* \mathfrak{M} is at most some *threshold* λ . Below, we give a formal description of our problem. We refer to [7, 50] for a thorough introduction.

Definition 1 (MDPs). A Markov decision process (MDP) is a tuple $\mathfrak{M} = (S, s_I, \text{Act}, P)$, where S is a finite set of states, $s_I \in S$ is the initial state, Act is a finite set of actions, and $P: S \times \text{Act} \times S \rightarrow [0, 1]$ is a transition probability function. For state s , let $\text{Act}(s) = \{a \in \text{Act} \mid \exists s' \in S: P(s, a, s') > 0\}$ be the enabled actions at s . For all states $s \in S$, we require $|\text{Act}(s)| \geq 1$ and $\sum_{s' \in S} P(s, a, s') = 1$. \triangle

For this paper, we fix an MDP $\mathfrak{M} = (S, s_I, \text{Act}, P)$, a set of *bad states* $B \subseteq S$, and a threshold $\lambda \in [0, 1]$. The *maximal*³ (*unbounded*) *reachability probability* to eventually reach a state in B from a state s is denoted by $\Pr^{\max}(s \models \diamond B)$. We characterize $\Pr^{\max}(s \models \diamond B)$ using the so-called *Bellman operator*. Let M^N denote the set of functions from N to M . Anticipating IC3 terminology, we call a function $F \in [0, 1]^S$ a *frame*. We denote by $F[s]$ the evaluation of frame F for state s .

³ Maximal with respect to all possible resolutions of nondeterminism in the MDP.

Definition 2 (Bellman Operator). For a set of actions $A \subseteq \text{Act}$, we define the Bellman operator for A as a frame transformer $\Phi_A: [0, 1]^S \rightarrow [0, 1]^S$ with

$$\Phi_A(F)[s] = \begin{cases} 1, & \text{if } s \in B \\ \max_{a \in A} \sum_{s' \in S} P(s, a, s') \cdot F[s'], & \text{if } s \notin B. \end{cases}$$

We write Φ_a for $\Phi_{\{a\}}$, Φ for Φ_{Act} , and call Φ simply the Bellman operator. \triangle

For every state s , the maximal reachability probability $\text{Pr}^{\max}(s \models \diamond B)$ is then given by the least fixed point of the Bellman operator Φ . That is,

$$\forall s: \text{Pr}^{\max}(s \models \diamond B) = (\text{lfp } \Phi)[s],$$

where the underlying partial order on frames is a complete lattice with ordering

$$F_1 \leq F_2 \quad \text{iff} \quad \forall s \in S: F_1[s] \leq F_2[s].$$

In terms of the Bellman operator, our formal problem statement reads as follows:

Given an MDP \mathfrak{M} with initial state s_I , a set B of bad states, and a *threshold* $\lambda \in [0, 1]$,

prove or refute that $\text{Pr}^{\max}(s_I \models \diamond B) = (\text{lfp } \Phi)[s_I] \leq \lambda$.

Whenever $\text{Pr}^{\max}(s_I \models \diamond B) \leq \lambda$ indeed holds, we say that the MDP \mathfrak{M} is *safe* (with respect to the set of bad states B and threshold λ); otherwise, we call it *unsafe*.

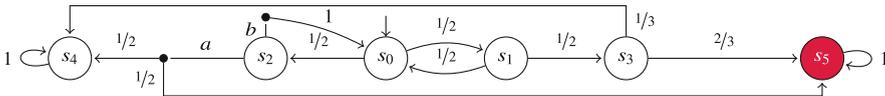


Fig. 1. The MDP \mathfrak{M} serving as a running example.

Recovery Statement 1. For $\lambda = 0$, our problem statement is equivalent to the qualitative reachability problem solved by (reverse) standard IC3, i.e., prove or refute that all bad states in B are unreachable from the initial state s_I .

Example 1. The MDP \mathfrak{M} in Fig. 1 consists of 6 states with initial state s_0 and bad states $B = \{s_5\}$. In s_2 , actions a and b are enabled; in all other states, one unlabeled action is enabled. We have $\text{Pr}^{\max}(s_0 \models \diamond B) = 2/3$. Hence, \mathfrak{M} is safe for all thresholds $\lambda \geq 2/3$ and unsafe for $\lambda < 2/3$. In particular, \mathfrak{M} is unsafe for $\lambda = 0$ as s_5 is *reachable* from s_0 . \triangle

3 The Core PrIC3 Algorithm

The purpose of PrIC3 is to prove or refute that the maximal probability to reach a bad state in B from the initial state s_I of the MDP \mathfrak{M} is at most λ . In this section, we explain the rationale underlying PrIC3. Moreover, we describe the core of PrIC3—called PrIC3_{IT}—which bears close resemblance to the main loop of standard IC3 for TSs.

Because of the inherent direction of the Bellman operator, we build PrIC3 on *reverse* IC3 [53], cf. Challenge 3. Reversing constitutes a shift from reasoning along the direction *initial-to-bad* to *bad-to-initial*. While this shift is mostly *inessential* to the fundamentals underlying IC3, the reverse direction is unswayable in the probabilistic setting. Whenever we draw a connection to standard IC3, we thus generally mean *reverse* IC3.

3.1 Inductive Frames

IC3 for TSs operates on (*qualitative*) frames representing sets of states of the TS at hand. A frame F can hence be thought of as a mapping⁴ from states to $\{0, 1\}$. In PrIC3 for MDPs, we need to move from a Boolean to a quantitative regime. Hence, a (*quantitative*) frame is a mapping from states to probabilities in $[0, 1]$.

For a given TS, consider the frame transformer T that adds to a given input frame F' all bad states in B and all predecessors of the states contained in F' . The rationale of standard (reverse) IC3 is to find a frame $F \in \{0, 1\}^S$ such that (I) the initial state s_I does not belong to F and (II) applying T takes us down in the partial order on frames, i.e.,

$$(I) \quad F[s_I] = 0 \quad \text{and} \quad (II) \quad T(F) \leq F .$$

Intuitively, (I) postulates the *hypothesis* that s_I cannot reach B and (II) expresses that F is closed under adding bad states and taking predecessors, thus affirming the hypothesis.

Analogously, the rationale of PrIC3 is to find a frame $F \in [0, 1]^S$ such that (I) F postulates that the probability of s_I to reach B is at most the threshold λ and (II) applying the Bellman operator Φ to F takes us down in the partial order on frames, i.e.,

$$(I) \quad F[s_I] \leq \lambda \quad \text{and} \quad (II) \quad \Phi(F) \leq F .$$

Frames satisfying the above conditions are called *inductive invariants* in IC3. We adopt this terminology. By *Park's Lemma* [48], which in our setting reads

$$\Phi(F) \leq F \quad \text{implies} \quad \text{lfp } \Phi \leq F ,$$

⁴ In IC3, frames are typically characterized by logical formulae. To understand IC3's fundamental principle, however, we prefer to think of frames as functions in $\{0, 1\}^S$ partially ordered by \leq .

an inductive invariant F would indeed *witness* that $\text{Pr}^{\max}(s_I \models \diamond B) \leq \lambda$, because

$$\text{Pr}^{\max}(s_I \models \diamond B) = (\text{lfp } \Phi)[s_I] \leq F[s_I] \leq \lambda .$$

If no inductive invariant exists, then standard IC3 will find a counterexample: a *path* from the initial state s_I to a bad state in B , which serves as a witness to refute. Analogously, PrIC3 will find a counterexample, but of a different kind: Since single paths are insufficient as counterexamples in the probabilistic realm (Challenge 2), PrIC3 will instead find a *subsystem* of states of the MDP witnessing $\text{Pr}^{\max}(s_I \models \diamond B) > \lambda$.

3.2 The PrIC3 Invariants

Analogously to standard IC3, PrIC3 aims to find the inductive invariant by maintaining a *sequence of frames* $F_0 \leq F_1 \leq F_2 \leq \dots$ such that $F_i[s]$ overapproximates the maximal probability of reaching B from s within *at most i steps*. This *i -step-bounded reachability probability* $\text{Pr}^{\max}(s \models \diamond^{\leq i} B)$ can be characterized using the Bellman operator: $\Phi(\mathbf{0})$ is the 0-step probability; it is 1 for every $s \in B$ and 0 otherwise. For any $i \geq 0$, we have

$$\text{Pr}^{\max}(s \models \diamond^{\leq i} B) = \left(\Phi^i(\Phi(\mathbf{0})) \right)[s] = \left(\Phi^{i+1}(\mathbf{0}) \right)[s] ,$$

where $\mathbf{0}$, the frame that maps every state to 0, is the least frame of the underlying complete lattice. For a finite MDP, the *unbounded* reachability probability is then given by the limit

$$\text{Pr}^{\max}(s \models \diamond B) = (\text{lfp } \Phi)[s] \stackrel{(*)}{=} \left(\lim_{n \rightarrow \infty} \Phi^n(\mathbf{0}) \right)[s] = \lim_{n \rightarrow \infty} \text{Pr}^{\max}(s \models \diamond^{\leq n} B) ,$$

where $(*)$ is a consequence of the well-known Kleene fixed point theorem [45].

The sequence $F_0 \leq F_1 \leq F_2 \leq \dots$ maintained by PrIC3 should frame-wise overapproximate the increasing sequence $\Phi(\mathbf{0}) \leq \Phi^2(\mathbf{0}) \leq \Phi^3(\mathbf{0}) \dots$. Pictorially:

$$\begin{array}{ccccccccccc} F_0 & \leq & F_1 & \leq & F_2 & \leq & \dots & \leq & F_k \\ \forall I & & \forall I & & \forall I & & & & \forall I \\ \mathbf{0} & \leq & \Phi(\mathbf{0}) & \leq & \Phi^2(\mathbf{0}) & \leq & \Phi^3(\mathbf{0}) & \leq & \dots & \leq & \Phi^{k+1}(\mathbf{0}) \end{array}$$

However, the sequence $\Phi(\mathbf{0}), \Phi^2(\mathbf{0}), \Phi^3(\mathbf{0}), \dots$ will never explicitly be known to PrIC3. Instead, PrIC3 will ensure the above frame-wise overapproximation property implicitly by enforcing the so-called PrIC3 *invariants* on the frame sequence F_0, F_1, F_2, \dots . Apart from allowing for a threshold $0 \leq \lambda \leq 1$ on the maximal reachability probability, these invariants coincide with the standard IC3 invariants (where $\lambda = 0$ is fixed). Formally:

Definition 3 (PrIC3 Invariants). *Frames F_0, \dots, F_k , for $k \geq 0$, satisfy the PrIC3 invariants, a fact we will denote by $\text{PrIC3Inv}(F_0, \dots, F_k)$, if all of the following hold:*

1. **Initiality :** $F_0 = \Phi(\mathbf{0})$
2. **Chain Property :** $\forall 0 \leq i < k: F_i \leq F_{i+1}$
3. **Frame-safety :** $\forall 0 \leq i \leq k: F_i[s_I] \leq \lambda$
4. **Relative Inductivity :** $\forall 0 \leq i < k: \Phi(F_i) \leq F_{i+1} \quad \triangle$

The PrIC3 invariants enforce the above picture: The *chain property* ensures $F_0 \leq F_1 \leq \dots \leq F_k$. We have $\Phi(\mathbf{0}) = F_0 \leq F_0$ by *initiality*. Assuming $\Phi^{i+1}(\mathbf{0}) \leq F_i$ as induction hypothesis, monotonicity of Φ and *relative inductivity* imply $\Phi^{i+2}(\mathbf{0}) \leq \Phi(F_i) \leq F_{i+1}$.

By overapproximating $\Phi(\mathbf{0}), \Phi^2(\mathbf{0}), \dots, \Phi^{k+1}(\mathbf{0})$, the frames F_0, \dots, F_k in effect bound the maximal step-bounded reachability probability of every state:

Lemma 1. *Let frames F_0, \dots, F_k satisfy the PrIC3 invariants. Then*

$$\forall s \ \forall i \leq k: \ Pr^{max}(s \models \diamond^{\leq i} B) \leq F_i[s].$$

In particular, Lemma 1 together with *frame-safety* ensures that the maximal step-bounded reachability probability of the *initial state* s_I to reach B is at most the threshold λ .

As for proving that the *unbounded* reachability probability is also at most λ , it suffices to find two consecutive frames, say F_i and F_{i+1} , that coincide:

Lemma 2. *Let frames F_0, \dots, F_k satisfy the PrIC3 invariants. Then*

$$\exists i < k: \ F_i = F_{i+1} \quad \text{implies} \quad Pr^{max}(s_I \models \diamond B) \leq \lambda.$$

Proof. $F_i = F_{i+1}$ and *relative inductivity* yield $\Phi(F_i) \leq F_{i+1} = F_i$, rendering F_i *inductive*. By Park’s lemma (cf. Sect. 3.1), we obtain $\text{lfp } \Phi \leq F_i$ and—by *frame-safety*—conclude

$$\text{Pr}^{max}(s_I \models \diamond B) = (\text{lfp } \Phi)[s_I] \leq F_i[s_I] \leq \lambda. \quad \square$$

3.3 Operationalizing the PrIC3 Invariants for Proving Safety

Lemma 2 gives us a clear angle of attack for *proving* an MDP safe: Repeatedly add and refine frames approximating step-bounded reachability probabilities for more and more steps while enforcing the PrIC3 invariants (cf. Definition 3.2) until two consecutive frames coincide.

Analogously to standard IC3, this approach is taken by the core loop $\text{PrIC3}_{\mathcal{H}}$ depicted in Algorithm 1; differences to the main loop of IC3 (cf. [23, Fig. 5]) are highlighted in red. A particular difference is that $\text{PrIC3}_{\mathcal{H}}$ is parameterized by a heuristic \mathcal{H} for finding suitable probabilities (see Challenge 3). Since the precise choice of \mathcal{H} is irrelevant for the soundness of $\text{PrIC3}_{\mathcal{H}}$, we defer a detailed discussion of suitable heuristics to Sect. 4.

Data: MDP \mathfrak{M} , set of bad states B , threshold λ
Result: true or false and a subset of the states of \mathfrak{M}

```

1  $F_0 \leftarrow \Phi(\mathbf{0}); F_1 \leftarrow \mathbf{1}; k \leftarrow 1; \text{oldSubsystem} \leftarrow \emptyset;$ 
2 while true do
3    $\text{success}, F_0, \dots, F_k, \text{subsystem} \leftarrow \text{Strengthen}_{\mathcal{H}}(F_0, \dots, F_k);$ 
4   if  $\neg \text{success}$  then return false, subsystem;
5    $F_{k+1} \leftarrow \mathbf{1};$ 
6    $F_0, \dots, F_{k+1} \leftarrow \text{Propagate}(F_0, \dots, F_{k+1});$ 
7   if  $\exists 1 \leq i \leq k: F_i = F_{i+1}$  then return true,  $\_;$ ;
8   if  $\text{oldSubsystem} = \text{subsystem}$  then return false, subsystem;
9    $k \leftarrow k + 1; \text{oldSubsystem} \leftarrow \text{subsystem};$ 
10 end

```

Algorithm 1: PrIC3 $_{\mathcal{H}}$ (\mathfrak{M} , B , λ)

As input, PrIC3 $_{\mathcal{H}}$ takes an MDP $\mathfrak{M} = (S, s_I, \text{Act}, P)$, a set $B \subseteq S$ of bad states, and a threshold $\lambda \in [0, 1]$. Since the input is never changed, we assume it to be *globally available*, also to subroutines. As output, PrIC3 $_{\mathcal{H}}$ returns true if two consecutive frames become equal. We hence say that PrIC3 $_{\mathcal{H}}$ is *sound* if it only returns true if \mathfrak{M} is safe.

We will formalize soundness using Hoare triples. For precondition ϕ , postcondition ψ , and program P , the triple $\{\phi\} P \{\psi\}$ is *valid* (for partial correctness) if, whenever program P starts in a state satisfying precondition ϕ and terminates in some state s' , then s' satisfies postcondition ψ . Soundness of PrIC3 $_{\mathcal{H}}$ then means validity of the triple

$$\{\text{true}\} \text{safe}, _ \leftarrow \text{PrIC3}_{\mathcal{H}}(\mathfrak{M}, B, \lambda) \{ \text{safe} \Rightarrow \text{Pr}^{\max}(s_I \models \diamond B) \leq \lambda \}.$$

Let us briefly go through the individual steps of PrIC3 $_{\mathcal{H}}$ in Algorithm 1 and convince ourselves that it is indeed sound. After that, we discuss why PrIC3 $_{\mathcal{H}}$ terminates and what happens if it is unable to prove safety by finding two equal consecutive frames.

How PrIC3 $_{\mathcal{H}}$ works. Recall that PrIC3 $_{\mathcal{H}}$ maintains a sequence of frames F_0, \dots, F_k which is initialized in l. 1 with $k = 1$, $F_0 = \Phi(\mathbf{0})$, and $F_1 = \mathbf{1}$, where the frame $\mathbf{1}$ maps every state to 1. Every time upon entering the **while**-loop in terms l. 2, the initial segment F_0, \dots, F_{k-1} satisfies all PrIC3 invariants (cf. Definition 3), whereas the full sequence F_0, \dots, F_k potentially violates frame-safety as it is possible that $F_k[s_I] > \lambda$.

In l. 3, procedure Strengthen $_{\mathcal{H}}$ —detailed in Sect. 4—is called to restore *all* PrIC3 invariants on the *entire* frame sequence: It either returns true if successful or returns false and a counterexample (in our case a subsystem of the MDP) if it was unable to do so. To ensure soundness of PrIC3 $_{\mathcal{H}}$, it suffices that Strengthen $_{\mathcal{H}}$ restores the PrIC3 invariants whenever it returns true. Formally, Strengthen $_{\mathcal{H}}$ must meet the following specification:

Definition 4. Procedure $\text{Strengthen}_{\mathcal{H}}$ is sound if the following Hoare triple is valid:

$$\begin{aligned} & \{ \text{PrIC3Inv}(F_0, \dots, F_{k-1}) \wedge F_{k-1} \leq F_k \wedge \Phi(F_{k-1}) \leq F_k \} \\ & \quad \text{success}, F_0, \dots, F_k, _ \leftarrow \text{Strengthen}_{\mathcal{H}}(F_0, \dots, F_k) \\ & \{ \text{success} \Rightarrow \text{PrIC3Inv}(F_0, \dots, F_k) \}. \end{aligned}$$

If $\text{Strengthen}_{\mathcal{H}}$ returns `true`, then a new frame $F_{k+1} = \mathbf{1}$ is created in l. 5. After that, the (now initial) segment F_0, \dots, F_k again satisfies all PrIC3 invariants, whereas the full sequence F_0, \dots, F_{k+1} potentially violates frame-safety at F_{k+1} . *Propagation* (l. 6) aims to speed up termination by updating $F_{i+1}[s]$ by $F_i[s]$ iff this does not violate relative inductivity. Consequently, the previously mentioned properties remain unchanged.

If $\text{Strengthen}_{\mathcal{H}}$ returns `false`, the PrIC3 invariants—premises to Lemma 2 for witnessing safety—cannot be restored and PrIC3 $_{\mathcal{H}}$ terminates returning `false` (l. 4). Returning `false` (also possible in l. 8) has by specification no affect on soundness of PrIC3 $_{\mathcal{H}}$.

In l. 7, we check whether there exist two identical consecutive frames. If so, Lemma 2 yields that the MDP is safe; consequently, PrIC3 $_{\mathcal{H}}$ returns `true`. Otherwise, we increment k and are in the same setting as upon entering the loop, now with an increased frame sequence; PrIC3 $_{\mathcal{H}}$ then performs another iteration. In summary, we obtain:

Theorem 1 (Soundness of PrIC3 $_{\mathcal{H}}$). *If $\text{Strengthen}_{\mathcal{H}}$ is sound and Propagate does not affect the PrIC3 invariants, then PrIC3 $_{\mathcal{H}}$ is sound, i.e., the following triple is valid:*

$$\{ \text{true} \} \text{safe}, _ \leftarrow \text{PrIC3}_{\mathcal{H}}(\mathfrak{M}, B, \lambda) \{ \text{safe} \implies P_{\gamma}^{\max}(s_I \models \diamond B) \leq \lambda \}$$

PrIC3 $_{\mathcal{H}}$ Terminates for Unsafe MDPs. If the MDP is unsafe, then there exists a step-bound n , such that $\text{Pr}^{\max}(s_I \models \diamond^{\leq n} B) > \lambda$. Furthermore, any sound implementation of $\text{Strengthen}_{\mathcal{H}}$ (cf. Definition 4) either immediately terminates PrIC3 $_{\mathcal{H}}$ by returning `false` or restores the PrIC3 invariants for F_0, \dots, F_k . If the former case never arises, then $\text{Strengthen}_{\mathcal{H}}$ will eventually restore the PrIC3 invariants for a frame sequence of length $k = n$. By Lemma 1, we have $F_n[s_I] \geq \text{Pr}^{\max}(s_I \models \diamond^{\leq n} B) > \lambda$ contradicting frame-safety.

PrIC3 $_{\mathcal{H}}$ Terminates for Safe MDPs. Standard IC3 terminates on safe finite TSs as there are only finitely many different frames, making every ascending chain of frames eventually stabilize. For us, frames map states to probabilities (Challenge 1), yielding *infinitely many possible frames* even for finite MDPs. Hence, $\text{Strengthen}_{\mathcal{H}}$ need not ever yield a stabilizing chain of frames. If it continuously fails to stabilize while repeatedly reasoning about the same set of states, we give up. PrIC3 $_{\mathcal{H}}$ checks this by comparing the subsystem $\text{Strengthen}_{\mathcal{H}}$ operates on with the one it operated on in the previous loop iteration (l. 8).

Theorem 2. *If $\text{Strengthen}_{\mathcal{H}}$ and Propagate terminate, then PrIC3 $_{\mathcal{H}}$ terminates.*

Recovery Statement 2. For qual. reachability ($\lambda = 0$), $\text{PrIC3}_{\mathcal{H}}$ never terminates in l. 8.

PrIC3 $_{\mathcal{H}}$ is Incomplete. Standard IC3 either proves safety or returns false and a counterexample—a single path from the initial to a bad state. As single paths are insufficient as counterexamples in MDPs (Challenge 2), $\text{PrIC3}_{\mathcal{H}}$ instead returns a *subsystem* of the MDP \mathfrak{M} provided by $\text{Strengthen}_{\mathcal{H}}$. However, as argued above, we cannot trust $\text{Strengthen}_{\mathcal{H}}$ to provide a stabilizing chain of frames. Reporting false thus only means that the given MDP *may* be unsafe; the returned subsystem has to be analyzed further.

The full PrIC3 algorithm presented in Sect. 5 addresses this issue. Exploiting the subsystem returned by $\text{PrIC3}_{\mathcal{H}}$, PrIC3 returns true if the MDP is safe; otherwise, it returns false and provides a true counterexample witnessing that the MDP is unsafe.

Example 2. We conclude this section with two example executions of $\text{PrIC3}_{\mathcal{H}}$ on a simplified version of the MDP in Fig. 1. Assume that action b has been removed. Then, for every state, exactly one action is enabled, i.e., we consider a Markov chain. Figure 2 depicts the frame sequences computed by $\text{PrIC3}_{\mathcal{H}}$ (for a reasonable \mathcal{H}) on that Markov chain for two thresholds: $5/9 = \text{Pr}^{\max}(s_0 \models \diamond B)$ and $9/10$. In particular, notice that *proving the coarser bound of $9/10$ requires fewer frames than proving the exact bound of $5/9$.* \triangle

| It. | 1 | | 2 | | | 3 | | | 4 | | | | 5 | | | | |
|-------|-------|---------|-------|---------|---------|---------|---------|---------|-------|---------|---------|---------|---------|-------|--|--|--|
| | F_1 | F_2 | F_1 | F_2 | F_3 | F_1 | F_2 | F_3 | F_4 | F_1 | F_2 | F_3 | F_4 | F_5 | | | |
| s_0 | $5/9$ | $5/9$ | $5/9$ | $5/9$ | $5/9$ | $5/9$ | $5/9$ | $5/9$ | $5/9$ | $5/9$ | $5/9$ | $5/9$ | $5/9$ | $5/9$ | | | |
| s_1 | 1 | $11/18$ | 1 | $11/18$ | $11/18$ | $11/18$ | $11/18$ | $11/18$ | 1 | $11/18$ | $11/18$ | $11/18$ | $11/18$ | 1 | | | |
| s_2 | 1 | $1/2$ | 1 | $1/2$ | $1/2$ | 1 | $1/2$ | $1/2$ | $1/2$ | 1 | $1/2$ | $1/2$ | $1/2$ | $1/2$ | | | |
| s_3 | 1 | 1 | 1 | $2/3$ | 1 | 1 | $2/3$ | $2/3$ | 1 | 1 | $2/3$ | $2/3$ | $2/3$ | 1 | | | |
| s_4 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | | | |
| s_5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | |

(a) Threshold $\lambda = 5/9$

| It. | 1 | | 2 | | | 3 | | | 4 | | | | |
|-------|--------|----------|--------|----------|----------|--------|----------|----------|----------|--------|----------|----------|----------|
| | F_1 | F_2 | F_1 | F_2 | F_3 | F_1 | F_2 | F_3 | F_4 | F_1 | F_2 | F_3 | F_4 |
| s_0 | $9/10$ | $9/10$ | $9/10$ | $9/10$ | $9/10$ | $9/10$ | $9/10$ | $9/10$ | $9/10$ | $9/10$ | $9/10$ | $9/10$ | $9/10$ |
| s_1 | 1 | $99/100$ | 1 | $99/100$ | $99/100$ | 1 | $99/100$ | $99/100$ | $99/100$ | 1 | $99/100$ | $99/100$ | $99/100$ |
| s_2 | 1 | $81/100$ | 1 | $81/100$ | $81/100$ | 1 | $81/100$ | $81/100$ | $81/100$ | 1 | $81/100$ | $81/100$ | $81/100$ |
| s_3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| s_4 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| s_5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

(b) Threshold $\lambda = 9/10$

Fig. 2. Two runs of $\text{PrIC3}_{\mathcal{H}}$ on the Markov chain induced by selecting action a in Fig. 1. For every iteration, frames are recorded after invocation of $\text{Strengthen}_{\mathcal{H}}$.

4 Strengthening in $\text{PrIC3}_{\mathcal{H}}$

When the main loop of $\text{PrIC3}_{\mathcal{H}}$ has created a new frame $F_k = \mathbf{1}$ in its previous iteration, this frame may violate frame-safety (Definition 3.3) because of $F_k[s_I] = 1 \not\leq \lambda$. The task of $\text{Strengthen}_{\mathcal{H}}$ is to restore the PrIC3 invariants on *all* frames F_0, \dots, F_k . To this end, our first *obligation* is to lower the value in frame $i = k$ for state $s = s_I$ to $\delta = \lambda \in [0, 1]$. We denote such an obligation by (i, s, δ) . Observe that implicitly $\delta = 0$ in the qualitative case, i.e., when proving unreachability. An obligation (i, s, δ) is *resolved* by updating the values assigned to state s in *all* frames F_1, \dots, F_i to at most δ . That is, for all $j \leq i$, we set $F_j[s]$ to the minimum

```

1  $Q \leftarrow \{(k, s_I, \lambda)\}$ ;
2 while  $Q$  not empty do
3    $(i, s, \delta) \leftarrow Q.\text{popMin}()$ ; /* pop obligation with minimal frame
   index */
4   if  $i = 0 \vee (s \in B \wedge \delta < 1)$  then
5     /* possible counterexample given by subsystem
       consisting of states popped from  $Q$  at some point */
6     return false, __,  $Q.\text{touched}()$ ;
7     /* check whether  $F_i[s] \leftarrow \delta$  violates relative inductivity */
8     if  $\exists a \in \text{Act}(s) : \Phi_a(F_{i-1})[s] > \delta$  then for such an  $a$ 
9        $\delta_1, \dots, \delta_n \leftarrow \mathcal{H}(s, a, \delta)$ ;
10       $\{s_1, \dots, s_n\} \leftarrow \text{Succs}(s, a)$ ;
11       $Q.\text{push}((i-1, s_1, \delta_1), \dots, (i-1, s_n, \delta_n), (i, s, \delta))$ ;
12    else /* resolve  $(i, s, \delta)$  without violating relative
       inductivity */
13       $F_1[s] \leftarrow \min(F_1[s], \delta); \dots; F_i[s] \leftarrow \min(F_i[s], \delta)$ ;
14    end
15  /*  $Q$  empty; all obligations have been resolved */ return
   true,  $F_0, \dots, F_k, Q.\text{touched}()$ ;

```

Algorithm 2: $\text{Strengthen}_{\mathcal{H}}(F_0, \dots, F_k)$

of δ and the original value $F_j[s]$. Such an update affects neither initiality nor the chain property (Definitions 3.1, 3.2). It may, however, violate relative inductivity (Definition 3.4), i.e., $\Phi(F_{i-1}) \leq F_i$. Before resolving obligation (i, s, δ) , we may thus have to further decrease some entries in F_{i-1} as well. Hence, *resolving obligations may spawn additional obligations* which have to be resolved first to maintain relative inductivity. In this section, we present a generic instance of $\text{Strengthen}_{\mathcal{H}}$ meeting its specification (Definition 4) and discuss its correctness.

Strengthen $_{\mathcal{H}}$ by Example. $\text{Strengthen}_{\mathcal{H}}$ is given by the pseudo code in Algorithm 2; differences to standard IC3 (cf. [23, Fig. 6]) are highlighted in red. Intuitively, $\text{Strengthen}_{\mathcal{H}}$ attempts to recursively resolve all obligations until either both frame-safety and relative inductivity are restored for *all* frames or it detects a *potential counterexample* justifying why it is unable to do so. We first consider an execution where the latter does not arise:

Example 3. We zoom in on Example 2: Prior to the second iteration, we have created the following three frames assigning values to the states s_0, s_5 :

$$F_0 = (0, 0, 0, 0, 1), \quad F_1 = (5/9, 1, 1, 1, 1), \quad \text{and} \quad F_2 = \mathbf{1}.$$

To keep track of unresolved obligations (i, s, δ) , $\text{Strengthen}_{\mathcal{H}}$ employs a priority queue Q which pops obligations with minimal frame index i first. Our first step is to ensure frame-safety of F_2 , i.e., alter F_2 so that $F_2[s_0] \leq 5/9$; we thus initialize the queue Q with the initial obligation $(2, s_0, 5/9)$ (l. 1). To do so, we check whether updating $F_2[s_0]$ to $5/9$ would invalidate relative inductivity (l. 6). This is indeed the case:

$$\Phi(F_1)[s_0] = 1/2 \cdot F_1[s_1] + 1/2 \cdot F_1[s_2] = 1 \not\leq 5/9.$$

To restore relative inductivity, **Strengthen _{\mathcal{H}}** spawns one new obligation for each relevant successor of s_0 . These have to be resolved before retrying to resolve the old obligation.⁵

In contrast to standard IC3, spawning obligations involves finding suitable probabilities δ (l. 7). In our example this means we have to spawn two obligations $(1, s_1, \delta_1)$ and $(1, s_2, \delta_2)$ such that $1/2 \cdot \delta_1 + 1/2 \cdot \delta_2 \leq 5/9$. There are *infinitely many choices* for δ_1 and δ_2 satisfying this inequality. Assume some heuristic \mathcal{H} chooses $\delta_1 = 11/18$ and $\delta_2 = 1/2$; we push obligations $(1, s_1, 11/18)$, $(1, s_2, 1/2)$, and $(2, s_0, 5/9)$ (ll. 8, 9). In the next iteration, we first pop obligation $(1, s_1, 11/18)$ (l. 3) and find that it can be resolved without violating relative inductivity (l. 6). Hence, we set $F_1[s_1]$ to $11/18$ (l. 11); no new obligation is spawned. Obligation $(1, s_2, 1/2)$ is resolved analogously; the updated frame is $F_1 = (5/9, 11/18, 1/2, 1)$. Thereafter, our initial obligation $(2, s_0, 5/9)$ can be resolved; relative inductivity is restored for F_0, F_1, F_2 . Hence, **Strengthen _{\mathcal{H}}** returns `true` together with the updated frames. \triangle

Strengthen _{\mathcal{H}} is Sound. Let us briefly discuss why Algorithm 2 meets the specification of a sound implementation of **Strengthen _{\mathcal{H}}** (Definition 4): First, we observe that Algorithm 2 alters the frames—and thus potentially invalidates the PrIC3 invariants—only in l. 11 by resolving an obligation (i, s, δ) with $\Phi(F_{i-1})[s] \leq \delta$ (due to the check in l. 6).

Let $F \langle s \mapsto \delta \rangle$ denote the frame F in which $F[s]$ is set to δ , i.e.,

$$F \langle s \mapsto \delta \rangle [s'] = \begin{cases} \delta, & \text{if } s' = s, \\ F[s'], & \text{otherwise.} \end{cases}$$

Indeed, resolving obligation (i, s, δ) in l. 11 lowers the values assigned to state s to at most δ *without* invalidating the PrIC3 invariants:

Lemma 3. *Let (i, s, δ) be an obligation and F_0, \dots, F_i , for $i > 0$, be frames with $\Phi(F_{i-1})[s] \leq \delta$. Then $\text{PrIC3Inv}(F_0, \dots, F_i)$ implies*

$$\text{PrIC3Inv} \left(F_0 \langle s \mapsto \min(F_0[s], \delta) \rangle, \dots, F_i \langle s \mapsto \min(F_i[s], \delta) \rangle \right).$$

Crucially, the precondition of Definition 4 guarantees that all PrIC3 invariants except frame safety hold initially. Since these invariants are never invalidated due to Lemma 3, Algorithm 2 is a sound implementation of **Strengthen _{\mathcal{H}}** if it restores frame safety whenever it returns `true`, i.e., once it leaves the loop with an empty obligation queue Q (ll. 12–13). Now, an obligation (i, s, δ) is only popped from Q in l. 3. As (i, s, δ) is added to Q upon reaching l. 9, the size of Q can only ever be reduced (without returning `false`) by resolving (i, s, δ) in l. 11. Hence, Algorithm 2 does not return `true` unless it restored frame safety by resolving, amongst all other obligations, the initial obligation (k, s_I, λ) . Consequently:

⁵ We assume that the set $\text{Succs}(s, a) = \{s' \in S \mid P(s, a, s') > 0\}$ of *relevant a-successors* of state s is returned in some arbitrary, but fixed order.

Lemma 4. *Procedure $\text{Strengthen}_{\mathcal{H}}$ is sound, i.e., it satisfies the specification in Definition 4.*

Theorem 3. *Procedure $\text{PrIC3}_{\mathcal{H}}$ is sound, i.e., satisfies the specification in Theorem 1.*

We remark that, analogously to standard IC3, resolving an obligation in l. 11 may be accompanied by *generalization*. That is, we attempt to update the values of multiple states at once. Generalization is, however, highly non-trivial in a probabilistic setting. We discuss three possible approaches to generalization in Sect. 6.2.

Strengthen $_{\mathcal{H}}$ Terminates. We now show that $\text{Strengthen}_{\mathcal{H}}$ as in Algorithm 2 terminates. The only scenario in which $\text{Strengthen}_{\mathcal{H}}$ may not terminate is if it keeps spawning obligations in l. 9. Let us thus look closer at how obligations are spawned: Whenever we detect that resolving an obligation (i, s, δ) would violate relative inductivity for some action a (l. 6), we first need to update the values of the successor states $s_1, \dots, s_n \in \text{Succs}(s, a)$ in frame $i-1$, i.e., we push the obligations $(i-1, s_1, \delta_1), \dots, (i-1, s_n, \delta_n)$ which have to be resolved first (ll. 7–9). It is noteworthy that, for a TS, a single action leads to a single successor state s_1 . Algorithm 2 employs a heuristic \mathcal{H} to determine the probabilities required for pushing obligations (l. 7). Assume for an obligation (i, s, δ) that the check in l. 6 yields $\exists a \in \text{Act}(s) : \Phi_a(F_{i-1})[s] > \delta$. Then \mathcal{H} takes s, a, δ and reports some probability δ_j for every a -successor s_j of s . However, an arbitrary heuristic of type $\mathcal{H}: S \times \text{Act} \times [0, 1] \rightarrow [0, 1]^*$ may lead to non-terminating behavior: If $\delta_1, \dots, \delta_n = F_{i-1}[s_1], \dots, F_{i-1}[s_n]$, then the heuristic has no effect. It is thus natural to require that an *adequate* heuristic \mathcal{H} yields probabilities such that the check $\Phi_a(F_{i-1})[s] > \delta$ in l. 6 cannot succeed twice for the *same obligation* (i, s, δ) and *same action* a . Formally, this is guaranteed by the following:

Definition 5. *Heuristic \mathcal{H} is adequate if the following triple is valid (for any frame F):*

$$\begin{aligned} & \{ \text{Succs}(s, a) = s_1, \dots, s_n \} \\ & \quad \delta_1, \dots, \delta_n \leftarrow \mathcal{H}(s, a, \delta) \\ & \{ \Phi_a(F \langle s_1 \mapsto \delta_1 \rangle \dots \langle s_n \mapsto \delta_n \rangle) [s] \leq \delta \} \quad \triangle \end{aligned}$$

Details regarding our implementation of heuristic \mathcal{H} are found in Sect. 6.1.

For an adequate heuristic, attempting to resolve an obligation (i, s, δ) (ll. 3 – 11) either succeeds after spawning it at most $|\text{Act}(s)|$ times or $\text{Strengthen}_{\mathcal{H}}$ returns `false`. By a similar argument, attempting to resolve an obligation $(i > 0, s, _)$ leads to at most $\sum_{a \in \text{Act}(s)} |\{s' \in S \mid P(s, a, s') > 0\}|$ other obligations of the form $(i-1, s', _)$. Consequently, the total number of obligations spawned by Algorithm 2 is bounded. Since Algorithm 2 terminates if all obligations have been resolved (l. 12) and each of its loop iterations either returns `false`, spawns obligations, or resolves an obligation, we conclude:

Lemma 5. $\text{Strengthen}_{\mathcal{H}}(F_0, \dots, F_k)$ terminates for every adequate heuristic \mathcal{H} .

Recovery Statement 3. Let \mathcal{H} be adequate. Then for qualitative reachability ($\lambda = 0$), all obligations spawned by $\text{Strengthen}_{\mathcal{H}}$ as in Algorithm 2 are of the form $(i, s, 0)$.

Strengthen $_{\mathcal{H}}$ returns false. There are two cases in which $\text{Strengthen}_{\mathcal{H}}$ fails to restore the PrIC3 invariants and returns `false`. The first case (the left disjunct of l. 4) is that we encounter an obligation for frame F_0 . Resolving such an obligation would inevitably violate *initiality*; analogously to standard IC3, we thus return `false`.

The second case (the right disjunct of l. 4) is that we encounter an obligation (i, s, δ) for a bad state $s \in B$ with a probability $\delta < 1$ (though, obviously, all $s \in B$ have probability =1). Resolving such an obligation would inevitably prevent us from restoring *relative inductivity*: If we updated $F_i[s]$ to δ , we would have $\Phi(F_{i-1})[s] = 1 > \delta = F_i[s]$. Notice that, in contrast to standard IC3, this second case *can* occur in PrIC3:

Example 4. Assume we have to resolve an obligation $(i, s_3, 1/2)$ for the MDP in Fig. 1. This involves spawning obligations $(i-1, s_4, \delta_1)$ and $(i-1, s_5, \delta_2)$, where s_5 is a bad state, such that $1/3 \cdot \delta_1 + 2/3 \cdot \delta_2 \leq 1/2$. Even for $\delta_1 = 0$, this is only possible if $\delta_2 \leq 3/4 < 1$. △

Strengthen $_{\mathcal{H}}$ Cannot Prove Unsafety. If standard IC3 returns `false`, it proves unsafety by constructing a counterexample, i.e., *a single path from the initial state to a bad state*. If PrIC3 returns `false`, there are two possible reasons: *Either* the MDP is indeed unsafe, *or* the heuristic \mathcal{H} at some point selected probabilities in a way such that $\text{Strengthen}_{\mathcal{H}}$ is unable to restore the PrIC3 invariants (even though the MDP might in fact be safe). $\text{Strengthen}_{\mathcal{H}}$ thus only returns a *potential* counterexample which either proves unsafety or indicates that our heuristic was inappropriate.

Counterexamples in our case consist of subsystems rather than a single path (see Challenge 2 and Sect. 5). $\text{Strengthen}_{\mathcal{H}}$ hence returns the set $Q.\text{touched}()$ of all states that eventually appeared in the obligation queue. This set is a conservative approximation, and optimizations as in [1] may be beneficial. Furthermore, in the qualitative case, our potential counterexample subsumes the counterexamples constructed by standard IC3:

Recovery Statement 4. Let \mathcal{H}_0 be the adequate heuristic mapping every state to 0. For qual. reachability ($\lambda = 0$), if `success = false` is returned by $\text{Strengthen}_{\mathcal{H}_0}(F_0, \dots, F_k)$, then $Q.\text{touched}()$ contains a path from the initial to a bad state.⁶

⁶ $Q.\text{touched}()$ might be restricted to only contain this path by some simple adaptations.

Data: global MDP \mathfrak{M} , set of bad states B , threshold λ
Result: true iff $\Pr^{\max}(s_I \models \diamond B) \leq \lambda$

```

1  $\Omega \leftarrow \text{Initialize}(); \text{ touched} \leftarrow \{s_I\};$ 
2 do
3    $\mathcal{H} \leftarrow \text{CreateHeuristic}(\Omega); \text{ safe}, \text{ subsystem} \leftarrow \text{PrIC3}_{\mathcal{H}}();$ 
4   if  $\text{ safe}$  then return true ;
5   if  $\text{ CheckRefutation}(\text{ subsystem})$  then return false ;
6    $\text{ touched} \leftarrow \text{Enlarge}(\text{ touched}, \text{ subsystem});$ 
7    $\Omega \leftarrow \text{Refine}(\Omega, \text{ touched});$ 
8 while  $\text{ touched} \neq S;$ 
9 return  $\Omega(s_I) \leq \lambda$ 

```

Algorithm 3: PrIC3: The outermost loop dealing with possibly imprecise heuristics

5 Dealing with Potential Counterexamples

Recall that our core algorithm $\text{PrIC3}_{\mathcal{H}}$ is incomplete for a fixed heuristic \mathcal{H} : It cannot give a conclusive answer whenever it finds a potential counterexample for two possible reasons: Either the heuristic \mathcal{H} turned out to be inappropriate or the MDP is indeed unsafe. The idea to overcome the former is to call $\text{PrIC3}_{\mathcal{H}}$ finitely often in an outer loop that generates new heuristics until we find an appropriate one: If $\text{PrIC3}_{\mathcal{H}}$ still does not report safety of the MDP, then it is indeed unsafe. We do not blindly generate new heuristics, but use the potential counterexamples returned by $\text{PrIC3}_{\mathcal{H}}$ to refine the previous one.

Let consider the procedure PrIC3 in Algorithm 3 which wraps our core algorithm $\text{PrIC3}_{\mathcal{H}}$ in more detail: First, we create an *oracle* $\Omega: S \rightarrow [0, 1]$ which (roughly) *estimates* the probability of reaching B for every state. A *perfect oracle* would yield *precise* maximal reachability probabilities, i.e., $\Omega(s) = \Pr^{\max}(s \models \diamond B)$ for every state s . We construct oracles by **user-supplied methods** (highlighted in blue). Examples of implementations of all user-supplied methods in Algorithm 3 are discussed in Sect. 7.

Assuming the oracle is good, but not perfect, we construct an adequate heuristic \mathcal{H} selecting probabilities based on the oracle⁷ for all successors of a given state: There are various options. The simplest is to pass-through the oracle values. A version that is more robust against noise in the oracle is discussed in Sect. 6. We then invoke $\text{PrIC3}_{\mathcal{H}}$. If $\text{PrIC3}_{\mathcal{H}}$ reports safety, the MDP is indeed safe by the soundness of $\text{PrIC3}_{\mathcal{H}}$.

Check Refutation. If $\text{PrIC3}_{\mathcal{H}}$ does not report safety, it reports a subsystem that hints to a *potential* counterexample. Formally, this subsystem is a subMDP of states that were ‘visited’ during the invocation of $\text{Strengthen}_{\mathcal{H}}$.

Definition 6 (subMDP). Let $\mathfrak{M} = (S, s_I, \text{Act}, P)$ be an MDP and let $S' \subseteq S$ with $s_I \in S'$. We call $\mathfrak{M}_{S'} = (S', s_I, \text{Act}, P')$ the subMDP induced by \mathfrak{M} and S' , where for all $s, s' \in S'$ and all $a \in \text{Act}$, we have $P'(s, a, s') = P(s, a, s')$. \triangle

⁷ We thus assume that heuristic \mathcal{H} invokes the oracle whenever it needs to guess some probability.

A subMDP $\mathfrak{M}_{S'}$ may be substochastic where missing probability mass never reaches a bad state. Definition 1 is thus relaxed: For all states $s \in S'$ we require that $\sum_{s' \in S'} P(s, a, s') \leq 1$. If the subsystem is unsafe, we can conclude that the original MDP \mathfrak{M} is also safe.

Lemma 6. *If \mathfrak{M}' is a subMDP of \mathfrak{M} and \mathfrak{M}' is unsafe, then \mathfrak{M} is also unsafe.*

The role of CheckRefutation is to establish whether the subsystem is indeed a true counterexample or a spurious one. Formally, CheckRefutation should ensure:

$$\{\text{true}\} \text{res} \leftarrow \text{CheckRefutation}(\text{subsystem}) \{\text{res} = \text{true} \Leftrightarrow \mathfrak{M}_{\text{subsystem}} \text{unsafe}\}.$$

Again, PrIC3 is backward compatible in the sense that a single fixed heuristic is always sufficient when reasoning about reachability ($\lambda = 0$).

Recovery Statement 5. *For qualitative reachability ($\lambda = 0$) and the heuristic \mathcal{H}_0 from Recovery Statement 4, PrIC3 invokes its core PrIC3 \mathcal{H} exactly once.*

This statement is true, as PrIC3 \mathcal{H} returns either *safe* or a subsystem containing a path from the initial state to a bad state. In the latter case, CheckRefutation detects that the subsystem is indeed a counterexample which cannot be spurious in the qualitative setting.

We remark that the procedure CheckRefutation invoked in l. 5 is a classical fallback; it runs an (alternative) model checking algorithm, e.g., solving the set of Bellman equations, for the subsystem. In the worst case, i.e., for $S' = S$, we thus solve exactly our problem statement. Empirically (Table 1) we observe that for reasonable oracles the procedure CheckRefutation is invoked on significantly smaller subMDPs. However, in the worst case the subMDP must include *all* paths of the original MDP, and then thus coincides.

Refine Oracle. Whenever we have neither proven the MDP safe nor unsafe, we refine the oracle to prevent generating the same subsystem in the next invocation of PrIC3 \mathcal{H} . To ensure termination, oracles should only be refined finitely often. That is, we need some progress measure. The set *touched* overapproximates all counterexamples encountered in some invocation of PrIC3 \mathcal{H} and we propose to use its size as the progress measure. While there are several possibilities to update *touched* through the user-defined procedure Enlarge (l. 6), every implementation should hence satisfy $\{\text{true}\} \text{touched}' \leftarrow \text{Enlarge}(\text{touched}, _) \{|\text{touched}'| > |\text{touched}|\}$. Consequently, after finitely many iterations, the oracle is refined with respect to all states. In this case, we may as well rely on solving the characteristic LP problem:

Lemma 7. *The algorithm PrIC3 in Algorithm 3 is sound and complete if Refine(Ω, S) returns a perfect oracle Ω (with S is the set of all states).*

Weaker assumptions on Refine are possible, but are beyond the scope of this paper. Moreover, the above lemma does not rely on the abstract concept that heuristic \mathcal{H} provides suitable probabilities after finitely many refinements.⁸

⁸ One could of course now also create a heuristic that is trivial for a perfect oracle and invoke PrIC3 \mathcal{H} with the heuristic for the perfect oracle, but there really is no benefit in doing so.

6 Practical PrIC3

So far, we gave a conceptual view on PrIC3, but now take a more practical stance. We detail important features of effective implementations of PrIC3 (based on our empirical evaluation). We first describe an implementation without generalization, and then provide a prototypical extension that allows for three variants of generalization.

6.1 A Concrete PrIC3 Instance Without Generalization

Input. We describe MDPs using the Prism guarded command language⁹, exemplified in Fig. 3. States are described by valuations to m (integer-valued) program variables `vars`, and outgoing actions are described by commands of the form

$$[] \text{ guard} \rightarrow \text{prob}1 : \text{update}1 \ \& \ \dots \ \& \ \text{prob}k : \text{update}k$$

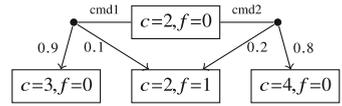
If a state satisfies `guard`, then the corresponding action with k branches exists; probabilities are given by `prob` i , the successor states are described by `update` i , see Fig. 3b.

```

module ex
  c : [0..20] init 0;   f : [0..1] init 0;
  [] c < 20 -> 0.1:(f'=1) + 0.9:(c'=c+1); // cmd 1
  [] c < 10 -> 0.2:(f'=1) + 0.8:(c'=c+2);
endmodule

```

(a) Prism code snippet



(b) Part of the corresponding MDP

Fig. 3. Illustrative Prism-style probabilistic guarded command language example

Encoding. We encode frames as logical formulae. Updating frames then corresponds to adding conjuncts, and checking for relative inductivity is a satisfiability call. Our encoding is as follows: States are assignments to the program variables, i.e., $\text{States} = \mathbb{Z}^m$. We use various uninterpreted functions, to whom we give semantics using appropriate constraints. Frames¹⁰ are represented by uninterpreted functions $\text{Frame} : \text{States} \rightarrow \mathbb{R}$ satisfying $\text{Frame}(s) = d$ implies $F[s] \geq d$. Likewise, the Bellman operator is an uninterpreted function $\text{Phi} : \text{States} \rightarrow \mathbb{R}$ such that $\text{Phi}(s) = d$ implies $\Phi(F)[s] \geq d$. Finally, we use $\text{Bad} : \text{States} \rightarrow \mathbb{B}$ with $\text{Bad}(s)$ iff $s \in B$.

Among the appropriate constraints, we ensure that variables are within their range, bound the values for the frames, and enforce $\text{Phi}(s) = 1$ for $s \in B$. We encode the guarded commands as exemplified by this encoding of the first command in Fig. 3:

$$\begin{aligned}
 & \forall s \in \text{States}: \neg \text{Bad}(s) \wedge s[c] < 20 \\
 & \implies \text{Phi}(s) = 0.1 \cdot \text{Frame}((s[c], 1)) + 0.9 \cdot \text{Frame}((s[c] + 1, s[f])).
 \end{aligned}$$

⁹ Preprocessing ensures a single thread (module) and no deadlocks.

¹⁰ In each operation, we only consider a single frame.

In our implementation, we optimize the encoding. We avoid the uninterpreted functions by applying an adapted Ackerman reduction. We avoid universal quantifiers, by first observing that we always ask whether a single state is not inductive, and then unfolding the guarded commands in the constraints that describe a frame. That encoding grows linear in the size of the maximal out-degree of the MDP, and is in the quantifier-free fragment of linear arithmetic (QFLRIA).

Heuristic. We select probabilities δ_i by solving the following optimization problem, with variables x_i , $\text{range}(x_i) \in [0, 1]$, for states $s_i \in \text{Succs}(s, a)$ and oracle Ω ¹¹.

$$\text{minimize } \sum_{\substack{i \\ s_i \notin B}}^k \left| \frac{x_i}{\sum_{j=1}^k x_j} - \frac{\Omega(s_i)}{\sum_{j=1}^n \Omega(s_j)} \right| \text{ s.t. } \delta = \sum_{i=1}^k P(s, a, s_i) \cdot \begin{cases} 1, & \text{if } s_i \in B, \\ x_i, & \text{else.} \end{cases}$$

The constraint ensures that, if the values x_i correspond to the actual reachability probabilities from s_i , then the reachability from state s is exactly δ . A constraint stating that $\delta \geq \dots$ would also be sound, but we choose equality as it preserves room between the actual probability and the threshold we want to show. Finally, the objective function aims to preserve the ratio between the suggested probabilities.

Repushing and Breaking Cycles. *Repushing* [23] is an essential ingredient of both standard IC3 and PrIC3. Intuitively, we avoid opening new frames and spawning obligations that can be deduced from current information. Since repushing generates further obligations in the current frame, its implementation requires that the detection of Zeno-behavior has to be moved from $\text{PrIC3}_{\mathcal{H}}$ into the $\text{Strengthen}_{\mathcal{H}}$ procedure. Therefore, we track the histories of the obligations in the queue. Furthermore, once we detect a cycle we first try to adapt the heuristic \mathcal{H} locally to overcome this cyclic behavior instead of immediately giving up. This local adaption reduces the number of $\text{PrIC3}_{\mathcal{H}}$ invocations.

Extended Queue. In contrast to standard IC3, the obligation queue might contain entries that vary only in their δ entry. In particular, if the MDP is not a tree, it may occur that the queue contains both (i, s, δ) and (i, s, δ') with $\delta > \delta'$. Then, (i, s, δ') can be safely pruned from the queue. Similarly, after handling (i, s, δ) , if some fresh obligation $(i, s, \delta'' > \delta)$ is pushed to the queue, it can be substituted with (i, s, δ) . To efficiently operationalize these observations, we keep an additional mapping which remains intact over multiple invocations of $\text{Strengthen}_{\mathcal{H}}$. We furthermore employed some optimizations for $Q.\text{touched}()$ aiming to track potential counterexamples better. After refining the heuristic, one may want to reuse frames or the obligation queue, but empirically this leads to performance degradation as the values in the frames are inconsistent with behavior suggested by the heuristic.

¹¹ If $\max \Omega(s_j) = 0$, we assume $\forall j. \Omega(s_j) = 0.5$. If $\delta = 0$, we omit rescaling to allow $\sum x_j = 0$.

6.2 Concrete PrIC3 with Generalization

So far, frames are updated by changing single entries whenever we resolve obligations (i, s, δ) , i.e., we add conjunctions of the form $F_i[s] \leq \delta$. Equivalently, we may add a constraint $\forall s' \in S : F_i[s'] \leq p_{\{s\}}(s')$ with $p_{\{s\}}(s) = \delta$ and $p_{\{s\}} = 1$ for all $s' \neq s$.

Generalization in IC3 aims to update a set G (including s) of states in a frame rather than a single one without invalidating relative inductivity. In our setting, we thus consider a function $p_G : G \rightarrow [0, 1]$ with $p_G(s) \leq \delta$ that assigns (possibly different) probabilities to all states in G . Updating a frame then amounts to adding the constraint

$$\forall s \in \text{States} : s \in G \implies \text{Frame}(s) \leq p_G(s).$$

Standard IC3 generalizes by iteratively “dropping” a variable, say v . The set G then consists of all states that do not differ from the fixed state s except for the value of v .¹² We take the same approach by iteratively dropping program variables. Hence, p_G effectively becomes a mapping from the value $s[v]$ to a probability. We experimented with four types of functions p_G that we describe for Markov chains. The ideas are briefly outlined below; details are beyond the scope of this paper.

Constant p_G . Setting all $s \in G$ to δ is straightforward but empirically not helpful.

Linear Interpolation. We use a linear function p_G that interpolates two points. The first point $(s[v], \delta)$ is obtained from the obligation (i, s, δ) . For a second point, consider the following: Let Com be the unique¹³ command active at state s . Among all states in G that are enabled in the guard of Com, we take the state s' in which $s'[v]$ is maximal¹⁴. The second point for interpolation is then $(s'[v], \Phi(F_{i-1})[s'])$. If the relative inductivity fails for p_G we do not generalize with p_G , but may attempt to find other functions.

Polynomial Interpolation. Rather than linearly interpolating between two points, we may interpolate using more than two points. In order to properly fit these points, we can use a higher-degree polynomial. We select these points using counterexamples to generalization (CTGs): We start as above with linear interpolation. However, if p_G is not relative inductive, the SMT solver yields a model with state $s'' \in G$ and probability δ'' , with s'' violating relative inductivity, i.e., $\Phi(F_{i-1})[s''] > \delta''$. We call $(s'', \Phi(F_{i-1})[s''])$ a CTG, and $(s''[v], \Phi(F_{i-1})[s''])$ is then a further interpolation point, and we repeat.

Technically, when generalizing using nonlinear constraints, we use real-valued arithmetic with a branch-and-bound-style approach to ensure integer values.

¹² Formally, $G = \{s' \mid \text{for all } v' \in \text{vars} \setminus \{v\} : s'(v') = s(v')\}$.

¹³ Recall that we have a Markov chain consisting of a single module.

¹⁴ This implicitly assumes that v is increased. Adaptions are possible.

Hybrid Interpolation. In polynomial interpolation, we generate high-degree polynomials and add them to the encoding of the frame. In subsequent invocations, reasoning efficiency is drastically harmed by these high-degree polynomials. Instead, we soundly approximate p_G by a piecewise linear function, and use these constraints in the frame.

7 Experiments

We assess how PrIC3 may contribute to the state of the art in probabilistic model checking. We do some early empirical evaluation showing that PrIC3 is feasible. We see ample room for further improvements of the prototype.

Implementation. We implemented a prototype¹⁵ of PrIC3 based on Sect. 6.1 in Python. The input is represented using efficient data structures provided by the model checker Storm. We use an incremental instance of Z3 [47] for each frame, as suggested in [23]. A solver for each frame is important to reduce the time spent on pushing the large frame-encodings. The optimization problem in the heuristic is also solved using Z3. All previously discussed generalizations (none, linear, polynomial, hybrid) are supported.

Oracle and Refinement. We support the (pre)computation of four different types of oracles for the **initialization** step in Algorithm 3: (1) A perfect oracle solving *exactly* the Bellman equations. Such an oracle is unrealistic, but interesting from a conceptual point. (2) Relative frequencies by recording all visited states during simulation. This idea is a naïve simplification of Q-learning. (3) Model checking with decision diagrams (DDs) and few value iterations. Often, a DD representation of a model can be computed fast, and the challenge is in executing sufficient value iterations. We investigate whether doing few value iterations yields a valuable oracle (and covers states close to bad states). (4) Solving a (pessimistic) LP from BFS partial exploration. States that are not expanded are assumed bad. Roughly, this yields oracles covering states close to the initial states.

To implement **Refine** (cf. Algorithm 3, l. 7), we create an LP for the subMDP induced by the touched states. For states whose successors are not in the touched states, we add a transition to B labeled with the oracle value as probability. The solution of the resulting LP updates the entries corresponding to the touched states.

For **Enlarge** (cf. Algorithm 3, l. 6), we take the union of the subsystem and the touched states. If this does not change the set of touched states, we also add its successors.

Setup. We evaluate the run time and memory consumption of our prototype of PrIC3. We choose a combination of models from the literature (BRP [21], ZeroConf [18]) and some structurally straightforward variants of grids (chain, double chain; see [11, Appendix A]). Since our prototype lacks the sophisticated

¹⁵ The prototype is available open-source from <https://github.com/moves-rwth/PrIC3>.

preprocessing applied by many state-of-the-art model checkers, it is more sensitive to the precise encoding of a model, e.g., the number of commands. To account for this, we generated new encodings for all models. All experiments were conducted on a single core of an Intel® Xeon® Platinum 8160 processor. We use a 15 min time-limit and report TO otherwise. Memory is limited to 8GB; we report MO if it is exceeded. Apart from the oracle, all parameters of our prototype remain fixed over all experiments. To give an impression of the run times, we compare our prototype with both the explicit (Storm_{sparse}) and DD-based (Storm_{dd}) engine of the model checker Storm 1.4, which compared favourably in QComp [29].

Results. In Table 1, we present the run times for various invocations of our prototype and Oracle 4¹⁶. In particular, we give the model name and the number of (non-trivial) states in the particular instance, and the (estimated) actual probability to reach B . For each model, we consider multiple thresholds λ . The next 8 columns report on the four variants of PrIC3 with varying generalization schemes. Besides the scheme with the run times, we report for each scheme the number of states of the largest (last) subsystem that CheckRefutation in Algorithm 3, l. 5 was invoked upon (column $|sub|$). The last two columns report on the run times for Storm that we provide for comparison. In each row, we mark with purple MDPs that are unsafe, i.e., PrIC3 refutes these MDPs for the given threshold λ . We **highlight** the best configurations of PrIC3.

Discussion. Our experiments give a mixed picture on the performance of our implementation of PrIC3. On the one hand, Storm significantly outperforms PrIC3 on most models. On the other hand, PrIC3 is capable of reasoning about huge, yet simple, models with up to 10^{12} states that Storm is unable to analyze within the time and memory limits. There is more empirical evidence that PrIC3 may complement the state-of-the-art:

First, *the size of thresholds matters*. Our benchmarks show that—at least without generalization—more “wobble room” between the precise maximal reachability probability and the threshold generally leads to a better performance. PrIC3 may thus prove bounds for large models where a precise quantitative reachability analysis is out of scope.

Second, *PrIC3 enjoys the benefits of bounded model checking*. In some cases, e.g., ZeroConf for $\lambda = 0.45$, PrIC3 refutes very fast as it does not need to build the whole model.

Third, if PrIC3 proves the safety of the system, it does so without relying on checking large subsystems in the CheckRefutation step.

Fourth, *generalization is crucial*. Without generalization, PrIC3 is unable to prove safety for any of the considered models with more than 10^3 states. With generalization, however, it can prove safety for very large systems and thresholds close to the exact reachability probability. For example, it proved safety of the

¹⁶ We explore $\min\{|S|, 5000\}$ states using BFS and Storm.

Table 1. Empirical results. Run times are in seconds; time out = 15 min.

| | $ S $ | $\Pr^{\max}(s_I \models \diamond B)$ | λ | w/o | sub | lin | sub | pol | sub | hyb | sub | Storm _{sparse} | Storm _{dd} | |
|--------------|-----------|--------------------------------------|-------------|-------|------|-------|------|-------|-----|------|------|-------------------------|---------------------|------|
| BRP | 10^3 | 0.035 | 0.1 | TO | – | TO | – | TO | – | TO | – | <0.1 | 0.12 | |
| | | | 0.01 | 51.3 | 324 | 125.8 | 324 | TO | – | MO | – | <0.1 | 0.18 | |
| | | | 0.005 | 10.9 | 188 | 38.3 | 188 | TO | – | MO | – | <0.1 | 0.1 | |
| ZeroConf | 10^4 | 0.5 | 0.9 | TO | – | TO | – | 0.4 | 0 | 0.1 | 0 | <0.1 | 296.8 | |
| | | | 0.52 | TO | – | TO | – | 0.2 | 0 | 0.2 | 0 | <0.1 | 282.6 | |
| | | | 0.45 | <0.1 | 1 | <0.1 | 1 | <0.1 | 1 | <0.1 | 1 | <0.1 | 300.2 | |
| | 10^9 | ~ 0.55 | 0.9 | TO | – | TO | – | 3.7 | 0 | MO | – | MO | TO | |
| | | | 0.75 | TO | – | TO | – | 3.4 | 0 | MO | – | MO | TO | |
| | | | 0.52 | TO | – | TO | – | TO | – | TO | – | MO | TO | |
| | | | 0.45 | <0.1 | 1 | <0.1 | 1 | <0.1 | 1 | <0.1 | 1 | MO | TO | |
| Chain | 10^3 | 0.394 | 0.9 | 18.8 | 0 | 60.2 | 0 | 1.2 | 0 | 0.3 | 0 | <0.1 | <0.1 | |
| | | | 0.4 | 20.1 | 0 | 55.4 | 0 | 0.9 | 0 | TO | – | <0.1 | <0.1 | |
| | | | 0.35 | 91.8 | 431 | 119.5 | 431 | TO | – | TO | – | <0.1 | <0.1 | |
| | | | | 0.3 | 46.1 | 357 | 64.0 | 357 | TO | – | TO | – | <0.1 | <0.1 |
| | 10^4 | 0.394 | 0.9 | TO | – | TO | – | 1.6 | 0 | 0.3 | 0 | <0.1 | 4.5 | |
| | | | 0.4 | TO | – | TO | – | 1.4 | 0 | TO | – | <0.1 | 4.9 | |
| | | | 0.3 | TO | – | TO | – | TO | – | TO | – | <0.1 | 4.9 | |
| | 10^{12} | 0.394 | 0.9 | TO | – | TO | – | 6.4 | 0 | MO | – | MO | TO | |
| | | | 0.4 | TO | – | TO | – | 6.0 | 0 | MO | – | MO | TO | |
| Double chain | 10^3 | 0.215 | 0.9 | 528.1 | 0 | 828.8 | 0 | 203.3 | 0 | 0.6 | 0 | <0.1 | <0.1 | |
| | | | 0.3 | 588.4 | 0 | TO | – | 138.3 | 0 | 0.5 | 0 | <0.1 | <0.1 | |
| | | | 0.216 | 597.4 | 0 | TO | – | 765.8 | 0 | MO | – | <0.1 | <0.1 | |
| | | | | 0.15 | TO | – | TO | – | TO | – | <0.1 | <0.1 | | |
| | 10^4 | 0.22 | 0.3 | TO | – | TO | – | 17.5 | 0 | 0.5 | 0 | 0.2 | 2.6 | |
| | | | 0.24 | TO | – | TO | – | 16.8 | 0 | MO | – | 0.2 | 2.7 | |
| | 10^7 | $2.6E^{-4}$ | $4E^{-3}$ | TO | – | TO | – | TO | – | MO | – | TO | TO | |
| | | | $2.7E^{-4}$ | TO | – | TO | – | 281.2 | 0 | MO | – | TO | TO | |

Chain benchmark with 10^{12} states for a threshold of 0.4 which differs from the exact reachability probability by 0.006.

Fifth, *there is no best generalization*. There is no clear winner out of the considered generalization approaches. Linear generalization always performs worse than the other ones. In fact, it performs worse than no generalization at all. The hybrid approach, however, occasionally has the edge over the polynomial approach. This indicates that more research is required to find suitable generalizations.

In [11, Appendix A], we also compare the additional three types of oracles (1–3). We observed that only few oracle refinements are needed to prove *safety*; for small models at most one refinement was sufficient. However, this does not hold if the given MDP is unsafe. DoubleChain with $\lambda = 0.15$, for example, and Oracle 2 requires 25 refinements.

8 Conclusion

We have presented PrIC3—the first truly probabilistic, yet conservative, extension of IC3 to quantitative reachability in MDPs. Our theoretical development

is accompanied by a prototypical implementation and experiments. We believe there is ample space for improvements including an in-depth investigation of suitable oracles and generalizations.

References

1. Ábrahám, E., Becker, B., Dehnert, C., Jansen, N., Katoen, J.-P., Wimmer, R.: Counterexample generation for discrete-time Markov models: an introductory survey. In: Bernardo, M., Damiani, F., Hähnle, R., Johnsen, E.B., Schaefer, I. (eds.) SFM 2014. LNCS, vol. 8483, pp. 65–121. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07317-0_3
2. Agha, G., Palmiskog, K.: A survey of statistical model checking. *ACM Trans. Model. Comput. Simul.* **28**(1), 6:1–6:39 (2018)
3. Agrawal, S., Chatterjee, K., Novotný, P.: Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. In: PACMPL 2(POPL), pp. 34:1–34:32 (2018)
4. de Alfaro, L., Kwiatkowska, M., Norman, G., Parker, D., Segala, R.: Symbolic model checking of probabilistic processes using MTBDDs and the kronecker representation. In: Graf, S., Schwartzbach, M. (eds.) TACAS 2000. LNCS, vol. 1785, pp. 395–410. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-46419-0_27
5. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model checking probabilistic systems. *Handbook of Model Checking*, pp. 963–999. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_28
6. Baier, C., Hermanns, H., Katoen, J.-P.: The 10,000 facets of MDP model checking. In: Steffen, B., Woeginger, G. (eds.) *Computing and Software Science*. LNCS, vol. 10000, pp. 420–451. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-91908-9_21
7. Baier, C., Katoen, J.-P.: *Principles of Model Checking*. MIT Press, Cambridge (2008)
8. 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., Kunčák, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 160–180. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_8
9. Barthe, G., Espitau, T., Ferrer Fioriti, L.M., Hsu, J.: Synthesizing probabilistic invariants via Doob’s decomposition. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 43–61. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41528-4_3
10. Bartocci, E., Kovács, L., Stankovič, M.: Automatic generation of moment-based invariants for prob-solvable loops. In: Chen, Y.-F., Cheng, C.-H., Esparza, J. (eds.) ATVA 2019. LNCS, vol. 11781, pp. 255–276. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31784-3_15
11. Batz, K., Junges, S., Kaminski, B.L., Katoen, J.-P., Matheja, C., Schröer, P.: Pric3: Property directed reachability for MDPS. *ArXiv e-prints* (2020). <https://arxiv.org/abs/2004.14835>
12. Biere, A.: Bounded model checking, *Handbook of Satisfiability*. *Frontiers in Artificial Intelligence and Applications*, vol. 185, pp. 457–481. IOS Press (2009)
13. 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

14. Brázdil, T., Chatterjee, K., Chmelík, M., Forejt, V., Křetínský, J., Kwiatkowska, M., Parker, D., Ujma, M.: Verification of Markov decision processes using learning algorithms. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 98–114. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_8
15. Chadha, R., Viswanathan, M.: A counterexample-guided abstraction-refinement framework for Markov decision processes. *ACM Trans. Comput. Logist.* **12**(1), 1:1–1:49 (2010)
16. Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 511–526. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_34
17. Chakraborty, S., Fried, D., Meel, K.S., Vardi, M.Y.: From weighted to unweighted model counting. In: IJCAI, pp. 689–695. AAAI Press (2015)
18. Cheshire, S., Aboba, B., Guttman, E.: Dynamic configuration of ipv4 link-local addresses. RFC **3927**, 1–33 (2005)
19. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Infinite-state invariant checking with IC3 and predicate abstraction. *FMSD* **49**(3), 190–218 (2016)
20. D’Argenio, P.R., Hartmanns, A., Sedwards, S.: Lightweight statistical model checking in nondeterministic continuous time. In: Margaria, T., Steffen, B. (eds.) ISoLA 2018. LNCS, vol. 11245, pp. 336–353. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03421-4_22
21. D’Argenio, P.R., Jeannot, B., Jensen, H.E., Larsen, K.G.: Reachability analysis of probabilistic systems by successive refinements. In: de Alfaro, L., Gilmore, S. (eds.) PAM-PROBMIV 2001. LNCS, vol. 2165, pp. 39–56. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44804-7_3
22. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A STORM is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčák, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_31
23. Eén, N., Mishchenko, A., Brayton, R.K.: Efficient implementation of property directed reachability. In: FMCAD, pp. 125–134. FMCAD Inc. (2011)
24. Fränzle, M., Hermanns, H., Teige, T.: Stochastic satisfiability modulo theory: a novel technique for the analysis of probabilistic hybrid systems. In: Egerstedt, M., Mishra, B. (eds.) HSCC 2008. LNCS, vol. 4981, pp. 172–186. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78929-1_13
25. Gretz, F., Katoen, J.-P., McIver, A.: PRINSYS—On a Quest for Probabilistic Loop Invariants. In: Joshi, K., Siegle, M., Stoelinga, M., D’Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 193–208. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40196-1_17
26. Gretz, F., Katoen, J.-P., McIver, A.: Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. *Perform. Eval.* **73**, 110–132 (2014)
27. Gurfinkel, A., Ivrii, A.: Pushing to the top. In: FMCAD, pp. 65–72. IEEE (2015)
28. Haddad, S., Monmege, B.: Interval iteration algorithm for MDPs and IMDPs. *Theor. Comput. Sci.* **735**, 111–131 (2018)
29. Hahn, E.M., Hartmanns, A., Hensel, C., Klauck, M., Klein, J., Křetínský, J., Parker, D., Quatmann, T., Ruijters, E., Steinmetz, M.: The 2019 comparison of tools for the analysis of quantitative formal models. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) TACAS 2019. LNCS, vol. 11429, pp. 69–92. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17502-3_5

30. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: PASS: abstraction refinement for infinite probabilistic models. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 353–357. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12002-2_30
31. Hahn, E.M., Li, Y., Schewe, S., Turrini, A., Zhang, L.: ISCASMC: a web-based probabilistic model checker. In: Jones, C., Pihlajasaari, P., Sun, J. (eds.) FM 2014. LNCS, vol. 8442, pp. 312–317. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06410-9_22
32. Han, T., Katoen, J.-P., Damman, B.: Counterexample generation in probabilistic model checking. *IEEE Trans. Software Eng.* **35**(2), 241–257 (2009)
33. Hark, M., Kaminski, B.L., Giesl, J., Katoen, J.-P.: Aiming low is harder: Induction for lower bounds in probabilistic program verification. In: PACMPL 4(POPL), 37:1–37:28 (2020)
34. Hartmanns, A., Hermanns, H.: The modest toolset: an integrated environment for quantitative modelling and verification. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 593–598. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_51
35. Hartmanns, A., Kaminski, B.L.: Optimistic value iteration. CAV. LNCS, Springer (2020). [to appear]
36. Hassan, Z., Bradley, A.R., Somenzi, F.: Better generalization in IC3. In: FMCAD, pp. 157–164. IEEE (2013)
37. Hermanns, H., Wachter, B., Zhang, L.: Probabilistic CEGAR. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 162–175. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70545-1_16
38. 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
39. Kaminski, B.L.: Advanced Weakest Precondition Calculi for Probabilistic Programs. Ph.D. thesis, RWTH Aachen University, Germany (2019). <http://publications.rwth-aachen.de/record/755408/files/755408.pdf>
40. Kaminski, B.L., Katoen, J.-P., Matheja, C., Olmedo, F.: Weakest precondition reasoning for expected runtimes of randomized algorithms. *J. ACM* **65**(5), 30:1–30:68 (2018)
41. Kattenbelt, M., Kwiatkowska, M.Z., Norman, G., Parker, D.: A game-based abstraction-refinement framework for Markov decision processes. *FMSD* **36**(3), 246–280 (2010)
42. Kozen, D.: A probabilistic PDL. In: STOC, pp. 291–297. ACM (1983)
43. 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
44. Lange, T., Neuhäuser, M.R., Noll, T., Katoen, J.-P.: IC3 software model checking. In: STTT, vol. 22, pp. 135–161 (2020)
45. Lassez, J.L., Nguyen, V.L., Sonenberg, L.: Fixed point theorems and semantics: a folk tale. *Inf. Process. Lett.* **14**(3), 112–116 (1982)
46. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science. Springer, New York (2005). <https://doi.org/10.1007/b138392>
47. 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

48. Park, D.: Fixpoint induction and proofs of program properties. *Machine intelligence* **5**, 59–78 (1969)
49. Polgreen, E., Brain, M., Fränzle, M., Abate, A.: Verifying reachability properties in Markov chains via incremental induction. *CoRR* abs/1909.08017 (2019)
50. Puterman, M.L.: *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics, Wiley, Hoboken (1994)
51. 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
52. Rabe, M.N., Wintersteiger, C.M., Kugler, H., Yordanov, B., Hamadi, Y.: Symbolic approximation of the bounded reachability probability in large Markov chains. In: Norman, G., Sanders, W. (eds.) *QEST 2014*. LNCS, vol. 8657, pp. 388–403. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10696-0_30
53. Seufert, T., Scholl, C.: Sequential verification using reverse PDR. *MBMV*. pp. 79–90. Shaker Verlag (2017)
54. Suenaga, K., Ishizawa, T.: Generalized property-directed reachability for hybrid systems. In: Beyer, D., Zufferey, D. (eds.) *VMCAI 2020*. LNCS, vol. 11990, pp. 293–313. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-39322-9_14
55. Takisaka, T., Oyabu, Y., Urabe, N., Hasuo, I.: Ranking and repulsing supermartingales for reachability in probabilistic programs. In: Lahiri, S.K., Wang, C. (eds.) *ATVA 2018*. LNCS, vol. 11138, pp. 476–493. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4_28
56. Vazquez-Chanlatte, M., Rabe, M.N., Seshia, S.A.: A model counter’s guide to probabilistic systems. *CoRR* abs/1903.09354 (2019)
57. Wimmer, R., Braitling, B., Becker, B.: Counterexample generation for discrete-time markov chains using bounded model checking. In: Jones, N.D., Müller-Olm, M. (eds.) *VMCAI 2009*. LNCS, vol. 5403, pp. 366–380. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-93900-9_29

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

