Formal Methods in System Design (2020) 56:90–126
https://doi.org/10.1007/s10703-020-00348-y

Exact quantitative probabilistic model checking through rational search

Umang Mathur[1] · Matthew S. Bauer[2] · Rohit Chadha[3] · A. Prasad Sistla[4] · Mahesh Viswanathan[1]

Published online: 29 July 2020

Abstract

Model checking systems formalized using probabilistic models such as discrete time Markov chains (DTMCs) and Markov decision processes (MDPs) can be reduced to computing constrained reachability properties. Linear programming methods to compute reachability probabilities for DTMCs and MDPs do not scale to large models. Thus, model checking tools often employ iterative methods to approximate reachability probabilities. These approximations can be far from the actual probabilities, leading to inaccurate model checking results. On the other hand, specialized techniques employed in existing state-of-the-art exact quantitative model checkers, don't scale as well as their iterative counterparts. In this work, we present a new model checking algorithm that improves the approximate results obtained by scalable iterative techniques to compute exact reachability probabilities. Our techniques are implemented as an extension of the PRISM model checker and are evaluated against other exact quantitative model checking engines.

Keywords Exact quantitative model checking · Markov decision processes · Markov chains · Probabilistic systems

1 Introduction

Probabilistic models such as discrete time Markov chains (DTMCs) and Markov decision processes (MDPs) are often used to describe systems in many application areas such as distributed systems [25,50], hardware communication protocols [26], reliability engineering in circuits [15,35,46,47], dynamic power management [14,49], networking [41,42] and security [20]. Probabilistic transitions in these models are used to capture random faults, the

We gratefully acknowledge the support of the following Grants—Umang Mathur was partially supported by a Google PhD Fellowship; Rohit Chadha was partially supported by NSF CNS-1553548 and NSF CCF-1900924; A. Prasad Sistla was partially supported by NSF CNS-1314485, NSF CCF-1319754, NSF CCF-1564296 and NSF CCF-1901069; and Mahesh Viswanathan was partially supported by NSF CCF-1901069.

✉ Umang Mathur
umathur3@illinois.edu

Extended author information available on the last page of the article

uncertainty of the environment, and explicit randomization used in algorithms. Analyzing properties of these probabilistic models is typically achieved through Probabilistic Computation Tree Logic (PCTL) model checking [51], wherein, a desired property of the model is specified as a PCTL formula, and the validity of such a formula is evaluated against the system in question.

PCTL is a quantitative extension of the temporal logic Computation Tree Logic (CTL) used to describe how a system evolves over time. For example, a PCTL formula ψ can be used to specify the property that almost surely no execution of a probabilistic program leads to a state with a deadlock. Given $\bowtie \in \{\leq, <, \geq, >\}$, the formula $\mathcal{P}_{\bowtie p}[\psi]$ expresses the property that the measure of computation paths satisfying ψ is $\bowtie p$. For a DTMC or MDP $\mathcal{M}$ and a PCTL formula ϕ, the PCTL model checking procedure recursively computes the set of states of $\mathcal{M}$ that satisfy subformulas of ϕ. Each recursive step, in turn, reduces to *constrained quantitative reachability*, wherein, given a set of good states G and a set of target states T, the goal is to compute the measure of the paths that reach T while remaining in G. If the model is decorated with *costs* or *rewards*, one may also be interested in computing the expected cost/reward of reaching T. It is well known that the constrained quantitative reachability problem for DTMCs and MDPs can be solved in polynomial time by a reduction to linear programming [10,51].

Despite low asymptotic complexity, linear programming, unfortunately, doesn't scale to large models and is rarely used to solve the constrained quantitative reachability problem in practice. Instead, probabilistic model checkers [22,23,32,38,39,44], typically compute *approximations* to the exact reachability probabilities through an iterative process. The most prevalent iterative technique is *value iteration*, where exact reachability probabilities may only be approached in the limit. For completion in a finite number of steps, it is common practice for model checking tools to terminate value iteration based on various heuristics, for example, when the difference between the computed reachability probabilities of successive iterations is "small". This approximation step may lead to unsound results [11,31,54], particularly in systems where high magnitude changes in value iteration are preceded by periods of stability that cause iteration to terminate prematurely.

Another iterative technique for computing constrained quantitative reachability is *interval iteration* [11,17,31,53]. Aimed at addressing the shortcomings of value iteration, interval iteration utilizes two simultaneous value iteration procedures converging to the exact probability values from above and below. While, this allows one to bound the error present in the approximation, the exact solution cannot be obtained from such an interval bound. Further, state-of-the-art model checkers typically implement these iterative procedures using floating-point numbers and finite-precision arithmetic. As a result, both iterative techniques are susceptible to overflows in floating-point calculations. The inherent imprecision in the approximate answers, combined with the errors introduced from finite precision arithmetic can be further compounded by the presence of nested probability operators in PCTL formulas when the sets of good states G and target states T are not correctly computed in the recursive step (see Example 3 in Sect. 3).

1.1 Contributions

In this article, we present a new algorithm and its implementation that *sharpens* approximate solutions computed by fast iterative techniques, to obtain the *exact* constrained reachability probabilities. The starting point of our approach is the observation that when the transition probabilities in the model are rational numbers, an exact solution is also a rational number of polynomially many bits. The second ingredient in our technique is an algorithm due to Kwek and Mehlhorn [40], which, given a "close enough" approximation to a rational number, finds the rational number efficiently. The rough outline of our algorithm is as follows. We use an iterative technique (value iteration or interval iteration) to compute an approximate solution and then apply the Kwek–Mehlhorn algorithm to find a close candidate rational solution. Since the approximate solution that we start with is of unknown quality, the candidate rational solution obtained may not be the exact answer. Therefore, we check if the candidate satisfies certain necessary and sufficient conditions that characterize the actual solution. This allows one to confirm the correctness of the candidate rational solution. If it is not correct, the process is repeated, starting with an approximate solution of improved precision. Precise details of the algorithm are given in Sect. 5.

We have implemented this approach as an extension of the PRISM model checker, called RationalSearch. Our tool computes exact constrained reachability probabilities and exact expected rewards when model checking DTMCs against PCTL specifications. Our implementation also computes min reachability probabilities and max expected rewards when model checking MDPs against PCTL specifications. For max reachability probabilities, we currently support only the Explicit engine of PRISM. Evaluation of our implementation against a broad set of examples from the PRISM benchmark suite [2] and case studies [3] shows that our technique can be applied to a wide array of examples. In many cases, our tool is orders of magnitude faster than the exact model checking engines implemented in state-of-the-art tools like PRISM [44] and STORM [22].

1.2 Related work

The work closest in spirit to ours is [30], which presents an approach to obtain exact solutions for reachability properties for MDPs and discounted MDPs. The underlying idea in [30] is to interpret the scheduler obtained for an approximate solution, as a *basis* for the linear program corresponding to the verification question. By examining the optimality of the solution associated with this basis, the exact solution can be obtained by improving the scheduler using the Simplex algorithm. This is significantly different from our approach. In particular, for the case of DTMCs (where there is no scheduler), the approach of [30] reduces to solving a linear program, which is known to be not scalable. Since the implementation from [30] is not available, we could not experimentally compare it with our approach.

Several existing tools [22,44] implement algorithms for exact quantitative model checking. Essentially these tools work by creating a model representation using rational numbers and performing a state elimination computation similar to Gauss elimination. Much of the infrastructure of this computation can be derived from *parametric model checking* techniques [21,23,33,34] that analyze systems in which portions of the model are left unspecified. These computations are intrinsically more complicated than those performed by approximation engines. Our techniques avoid these expensive computations while still producing exact solutions for a large class of examples.

1.3 History and organization

An extended abstract of this article appeared in [13]. The main difference from [13] is that in [13], we had claimed that in order to check whether a candidate solution vector represents the actual exact solution of max/min reachability probabilities or that of max/min expected costs for MDPs, it suffices to only check that the candidate vector is a solution to a linear program. This happens to be incorrect for the case of max reachability probabilities and min expected costs (see Sect. 4), and additional checks are required to claim that the candidate solution vector is indeed correct (Lemmas 1, 3). We have modified our algorithm to reflect this. We have also updated our prototype implementation for computing max reachability probabilities and evaluated the new version on our benchmarks. We do not currently support the computation of min expected costs. We have also computed the asymptotic complexity of the algorithm (see Theorem 2). Further, the version of RationalSearch evaluated in this work extends our original prototype by integrating with interval iteration and including several performance enhancements. Additionally, we describe the full details of our implementation and provide a more comprehensive evaluation of the tool.

The paper is organized as follows. Section 2 discusses preliminary notations, definitions and algorithms concerning PCTL model checking of DTMCs and MDPs. Section 3 describes iterative model checking techniques and their shortcomings. In Sect. 4, we discuss fixpoint characterizations for solutions to PCTL model checking questions of MDPs. In Sect. 5 we present our exact model checking algorithm. Sects. 6 and 7 describe the implementation and evaluation of our techniques and we conclude with Sect. 8.

2 Preliminaries

A common technique in the analysis of systems is to model them as *state transitions systems* where states describe information about the system at a point in time and transitions describe how the system evolves from one state to another. When this evolution is governed by random phenomena, such state transition systems can then be enriched to capture probabilistic behavior. The resulting model is known as a DTMC, in which every state is mapped to a distribution over the successor states. MDPs generalize DTMCs, in that, the distribution over the successor states is non-deterministically chosen. Our presentation of DTMCs and MDPs follows [52]. We begin by formalizing DTMCs and introducing the logic Probabilistic computation tree logic (PCTL), which is used to specify properties of DTMCs. We then discuss the model checking algorithm for DTMCs. We next formally describe MDPs and then present PCTL semantics and model checking for MDPs. Unless otherwise stated, all the transition probabilities in the paper are assumed to be rational numbers. The set of rational numbers shall be denoted as $\mathbb{Q}$ and the set of non-negative rational numbers as $\mathbb{Q}^{\geq 0}$.

2.1 Discrete time Markov chains (DTMCs)

2.1.1 Syntax and semantics

A DTMC is a tuple $\mathcal{M} = (Z, \Delta, \mathbf{C}, L)$ where Z is a finite set of states, $\Delta : Z \to \mathsf{Dist}(Z)$ is the *probabilistic transition function* that maps every state to a probability distribution over Z, $\mathbf{C} : Z \times Z \to \mathbb{Q}^{\geq 0}$ is a cost (or reward) structure and $L : Z \to 2^{\mathsf{AP}}$ is a labeling function that maps states to subsets of AP, the set of atomic propositions. For each $z \in Z$, $\Delta(z) : Z \to \mathbb{Q} \cap [0, 1]$ defines a discrete probability distribution over Z, that is, $\Delta(z)(z') \geq 0$ for all $z' \in Z$, and $\sum_{z' \in Z} \Delta(z)(z') = 1$. We will henceforth denote $\Delta(z)(z')$ by $\Delta(z, z')$.

Intuitively, a DTMC $\mathcal{M}$ evolves as follows. If $\mathcal{M}$ is in state z, it transitions to state z' with probability $\Delta(z, z')$. Formally, a finite (resp. infinite) path ρ of $\mathcal{M}$ is a finite (resp. infinite) sequence of states $z_0 \to z_1 \to \cdots$ such that $\Delta(z_i, z_{i+1}) > 0$. We write $\rho(i)$ to denote the ith state z_i in ρ. For a DTMC $\mathcal{M}$, the set of all infinite paths starting from state z will be denoted by $\mathsf{Paths}_z(\mathcal{M})$. For a finite path $\rho_{\text{fin}} = z_0 \to \cdots \to z_m$ starting at state z_0, we associate a measure $\mathsf{prob}_{z_0}(\rho_{\text{fin}}) = \prod_{i=0}^{m-1} \Delta(z_i, z_{i+1})$. The cylinder set of ρ_{fin} is $\mathsf{Cyl}(\rho_{\text{fin}}) = \{\rho \in \mathsf{Paths}_{z_0}(\mathcal{M}) \mid \rho_{\text{fin}}$ is a prefix of $\rho\}$ and its associated measure is $\mathsf{prob}_{z_0}(\mathsf{Cyl}(\rho_{\text{fin}})) = \mathsf{prob}_{z_0}(\rho_{\text{fin}})$. This measure prob_{z_0} can be extended to a unique probability measure over the smallest σ-algebra on $\mathsf{Paths}_{z_0}(\mathcal{M})$ that contain all cylinder sets; the resulting probability measure will also be denoted by prob_{z_0}.

2.1.2 Reachability probability and expected cost

Let $z \in Z$ and $F \subseteq Z$. The probability of reaching F from the state z is defined to be the measure $\mathsf{prob}_z(Reach)$ where $Reach$ is the set of all infinite paths ρ such that $\rho(0) = z$ and $\rho(i) \in F$ for some $i \geq 0$. For defining expected cost, we first define the function $\mathsf{cost}_z(F) : \mathsf{Paths}_z \to \mathbb{Q}^{\geq 0}$ such that for any $\rho \in \mathsf{Paths}_z(\mathcal{M})$, $\mathsf{cost}_z(F)(\rho) = \sum_{i=0}^{m-1} \mathbf{C}(z_i, z_{i+1})$ if $z_0 \to \cdots \to z_m$ is the shortest prefix of ρ such that $z_m \in F$ and $\mathsf{cost}_z(F)(\rho) = \infty$ if no such prefix exists. Let $\mathbb{E}_z$ be the usual expectation on $\mathsf{Paths}_z(\mathcal{M})$ with respect to the measure prob_z. Then $\mathbb{E}_z[\mathsf{cost}_z(F)]$ is defined to be the expected cost of reaching F. Observe that, following [52], the expected cost $\mathbb{E}_z[\mathsf{cost}_z(F)]$ is finite iff the set F can be reached from z with probability 1.

Example 1 Consider an embedded control system [43] comprised of an input processor, a main processor, an output processor and a bus. In each cycle of the system, the input processor collects data from a set of n sensors $S_1, S_2, \ldots, S_n$. The main processor polls the input processor and passes instructions to the output processor controlling a set of m actuators $A_1, A_2, \ldots A_m$. Communication between processors occurs over the bus. The system is designed to tolerate failures in a limited number of components. If the input processor reports that the number of sensor failures exceeds some threshold MAX_FAILURES, then the main processor shuts the system down. Otherwise, it activates the actuators, which again, are prone to failure. When the probabilities with which each of these components fail are known, one can model the system's reliability using a DTMC. In Fig. 1, we give a DTMC that models a single cycle of such a system with $n = 2$ sensors and $m = 1$ actuator. For simplicity, we assume that each sensor fails with probability E_s and each actuator fails with probability E_a. States of the model are labeled with $e_1^s, \ldots, e_n^s \in \{0, 1\}$ and $e_1^a, \ldots, e_m^a \in \{0, 1\}$, where $e_i^s = 1$ denotes the failure of sensor S_i and $e_i^a = 1$ denotes the failure of actuator A_i. In Fig. 1, we omit labels if they are not relevant in a particular state.

2.2 Probabilistic computation tree logic (PCTL)

Properties of DTMCs be expressed in the logic PCTL, which extends the temporal logic CTL with the ability to reason quantitatively. We start by describing the syntax and semantics of PCTL.

2.2.1 Syntax

Analogous to CTL, PCTL has state formulas that model properties of states and path formulas that model properties of paths.

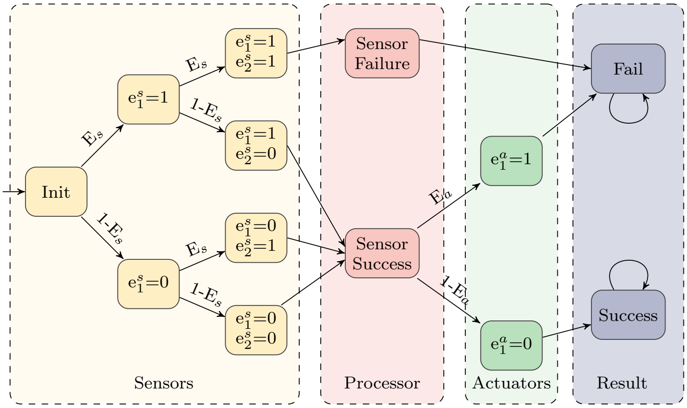

Fig. 1 Markov chain for a simple embedded control system with two sensors and one actuator tolerating a single sensor fault

Definition 1 Let $a \in \mathsf{AP}$ be an atomic proposition, $\bowtie \in \{\leq, <, \geq, >\}$, $p \in [0, 1]$, $c \in \mathbb{Q}^{\geq 0}$ and $k \in \mathbb{N}$. The syntax of PCTL is

$$\phi ::= \mathsf{true}\, a\, \neg\phi\, \phi \wedge \phi\, \mathcal{P}_{\bowtie p}[\psi]\, \mathcal{E}_{\bowtie c}[\phi]$$
$$\text{where } \psi ::= \mathcal{X}\phi \mid \phi\mathcal{U}\phi.$$

Here ϕ is a state formula and ψ a path formula.

2.2.2 Semantics

The state formulas are interpreted over states and path formulas over infinite paths.

Definition 2 Let $\mathcal{M} = (Z, \Delta, \mathbf{C}, L)$ be a DTMC, ϕ, ϕ_1, ϕ_2 be state formulas and ψ be a path formula. The satisfaction relation $\models$ for PCTL state formulas and for PCTL path formulas is defined by mutual induction:

$$\begin{array}{lcl}
\mathcal{M}, z \models \mathsf{true} & & \text{for all } z \in Z \\
\mathcal{M}, z \models a & \Leftrightarrow & a \in L(z) \\
\mathcal{M}, z \models \neg\phi & \Leftrightarrow & \mathcal{M}, z \not\models \phi \\
\mathcal{M}, z \models \phi_1 \wedge \phi_2 & \Leftrightarrow & \mathcal{M}, z \models \phi_1 \text{ and } \mathcal{M}, z \models \phi_2 \\
\mathcal{M}, z \models \mathcal{P}_{\bowtie p}[\psi] & \Leftrightarrow & p_z(\psi) \bowtie p \\
\mathcal{M}, z \models \mathcal{E}_{\bowtie c}[\phi] & \Leftrightarrow & e_z(\phi) \bowtie c \\
 & & \\
\mathcal{M}, \rho \models \mathcal{X}\phi & \Leftrightarrow & \mathcal{M}, \rho(1) \models \phi \\
\mathcal{M}, \rho \models \phi_1\mathcal{U}\phi_2 & \Leftrightarrow & \exists i \geq 0 : (\mathcal{M}, \rho(i) \models \phi_2 \text{ and } \forall j < i : \mathcal{M}, \rho(j) \models \phi_1)
\end{array}$$

where $p_z(\psi) = \mathsf{prob}_z(\{\rho \in \mathsf{Paths}_z(\mathcal{M}) \mid \mathcal{M}, \rho \models \psi\})$, $e_z(\phi) = \mathbb{E}_z[\mathsf{cost}_z(Z_\phi)]$ with $Z_\phi = \{z' \in Z \mid \mathcal{M}, z' \models \phi\}$.

Example 2 Consider the DTMC modeling an embedded control system from Example 1. One can describe many important properties of this model using PCTL as follows ($\bowtie, \bowtie' \in \{\leq, \geq, <, >\}$ and $p \in [0, 1]$)

1. The probability of success is $\bowtie p$:

$$\mathcal{P}_{\bowtie p}\ [\ \mathsf{true}\ \mathcal{U}\ \text{“Sucess}''\ ]$$

2. The probability of reaching the set of states where there are no sensor failures is $\bowtie p$:

$$\mathcal{P}_{\bowtie p}\ [\ \mathsf{true}\ \mathcal{U}\ (e^s_1 + \cdots + e^s_n = 0)\ ]$$

3. Let G be the set of states from which the probability of reaching a state where sensor S_1 fails is $\bowtie \frac{1}{2}$. Let T be the set of states from which the probability of reaching a state in which actuator A_1 fails is 0. The probability of remaining in some state from the set G until reaching a state in T is $\bowtie' p$:

$$\mathcal{P}_{\bowtie' p}\ [\ \mathcal{P}_{\bowtie \frac{1}{2}}[\mathsf{true}\ \mathcal{U}\ (e^s_1{=}1)]\ \mathcal{U}\ \mathcal{P}_{\leq 0}[\mathsf{true}\ \mathcal{U}\ (e^a_1{=}1)]\ ]$$

2.3 PCTL model checking

The PCTL model checking question asks, given a state z_0 of a DTMC $\mathcal{M}$ and a PCTL formula ϕ, determine whether $\mathcal{M}, z_0 \models \phi$. Similar to the model checking algorithm for CTL, the PCTL model checking algorithm recursively computes the set of states satisfying a state sub-formula (see [10,52] for the complete details). We consider the special case when the formula ϕ is of the form $\mathcal{P}_{\bowtie p}[\phi_1\ \mathcal{U}\ \phi_2]$.

Let ϕ, ϕ' be state formulas. To check whether $\mathcal{M}, z_0 \models \mathcal{P}_{\bowtie p}[\phi\ \mathcal{U}\ \phi']$, one recursively computes the set of states Z_ϕ and $Z_{\phi'}$ satisfying the state formulas ϕ and ϕ', respectively. These can then be used to derive, for every $z{\in}Z$, the quantity $p_z(\phi\ \mathcal{U}\ \phi')$ which represents the probability of reaching the set $Z_{\phi'}$ while remaining in the set Z_ϕ, starting from the state z. Let $\lambda z.\ p_z(\phi\ \mathcal{U}\ \phi')$ denote the state-indexed vector (or the function) that maps $z \in Z$ to $p_z(\phi\ \mathcal{U}\ \phi')$. The state-indexed vector $\lambda z.\ p_z(\phi\ \mathcal{U}\ \phi')$ can be computed as the *unique* solution to following linear program [10,52]:

$$y_z = \begin{cases} 0 & \text{if } z \in \mathsf{Prob}_0[\phi\ \mathcal{U}\ \phi'] \\ 1 & \text{if } z \in \mathsf{Prob}_1[\phi\ \mathcal{U}\ \phi'] \\ \sum\limits_{z' \in Z} \Delta(z, z') \cdot y_{z'} & \text{otherwise} \end{cases} \tag{1}$$

In the equation above, $\mathsf{Prob}_0[\phi\ \mathcal{U}\ \phi']$ and $\mathsf{Prob}_1[\phi\ \mathcal{U}\ \phi']$ are the set of states of $\mathcal{M}$ that satisfy $\phi\ \mathcal{U}\phi'$ with probability 0 and 1, respectively. These sets can be determined via a pre-computation step that analyzes the underlying graph structure of the DTMC. The value of y_z in the solution is exactly the value $p_z(\phi\ \mathcal{U}\ \phi')$. To verify if $\mathcal{M}, z_0 \models \mathcal{P}_{\bowtie p}[\phi\ \mathcal{U}\ \phi']$, one computes $\lambda z.\ p_z(\phi\ \mathcal{U}\ \phi')$ and compares $p_{z_0}(z\ \mathcal{U}\ z') \bowtie p$. The model checking algorithm for $\neg\phi$, $\phi \wedge \phi'$, and $\mathcal{P}_{\bowtie p}[\mathcal{X}\phi]$ are as expected.

To check whether $\mathcal{E}_{\bowtie c}[\phi]$, one recursively computes Z_ϕ satisfying the state formula ϕ. The expected costs $\{e_z(\phi) \mid z \in Z\}$ can then be computed as the *unique* solution to the following linear program [52] (with the convention that $0 \cdot \infty = 0$):

$$y_z = \begin{cases} 0 & \text{if } z \in Z_\phi \\ \infty & \text{iff } z \in \mathsf{Cost}_\infty[\phi] \\ \sum\limits_{z' \in Z} \Delta(z, z') \cdot (\mathbf{C}(s, s') + y_{z'}) & \text{otherwise} \end{cases} \tag{2}$$

In the equation above, $\mathsf{Cost}_\infty[\phi]$ is the set of states for which the expected cost is ∞. The set Cost_∞ is exactly the set of states that satisfy ϕ with probability < 1, and can be determined via a pre-computation step that analyzes the underlying graph structure of the DTMC.

2.4 Markov decision processes (MDPs) and PCTL

2.4.1 Syntax

An MDP is a tuple $\mathcal{M} = (Z, \mathsf{Act}, \Delta, \mathbf{C}, L)$ where Z is a finite set of states, Act is a finite set of actions, the partial function $\Delta : Z \times \mathsf{Act} \hookrightarrow \mathsf{Dist}(Z)$, called *probabilistic transition function*, maps pairs of states and actions to probability distributions over Z, $\mathbf{C} : Z \times \mathsf{Act} \to \mathbb{Q}^{\geq 0}$ is a cost (or reward) structure and $L : Z \to 2^{\mathsf{AP}}$ is a labeling function. The set $\mathsf{enabled}(z) = \{\alpha \in \mathsf{Act} \mid \Delta(z, \alpha) \text{ is defined}\}$, describing the actions enabled from a state z, is assumed to be non-empty for every $z \in Z$. An MDP, therefore, differs from a DTMC, in that, at each state z, there is a choice among several possible distributions. The choice of which distribution to *trigger* is resolved by a *scheduler* (or an attacker). Informally, an MDP $\mathcal{M}$ evolves as follows. It starts from some state $z_0 \in Z$. After i execution steps, if $\mathcal{M}$ is in state z, the scheduler chooses an action $\alpha \in \mathsf{enabled}(z)$, which then defines a unique probability distribution μ given by $\Delta(z, \alpha)$. The process then moves to state z' in step $(i + 1)$ with probability $\Delta(z, \alpha)(z')$. We will write $\Delta(z, \alpha, z')$ to denote $\Delta(z, \alpha)(z')$ when $\alpha \in \mathsf{enabled}(z)$.

2.4.2 Reachability probability and expected cost

Formally, a path ρ of an MDP $\mathcal{M}$ is a sequence $z_0 \xrightarrow{\alpha_1} z_1 \xrightarrow{\alpha_2} \cdots$ such that for each $i \geq 0$, we have $\alpha_{i+1} \in \mathsf{enabled}(z_i)$ and $\Delta(z_i, \alpha_{i+1}, z_{i+1}) > 0$. As discussed above, the choice of which action to trigger in a given state is resolved by a scheduler, which is a function $\mathfrak{S}$ from finite paths to actions[1]. A path $z_0 \xrightarrow{\alpha_1} z_1 \xrightarrow{\alpha_2} \cdots$ is a $\mathfrak{S}$-path if $\mathfrak{S}(z_0 z_1 \ldots z_i) = \alpha_{i+1}$ for all $i \geq 0$. We will write $\mathsf{Paths}_z(\mathcal{M})$ for the set of infinite paths starting from z and $\mathsf{Paths}_z^{\mathfrak{S}}(\mathcal{M})$ for the set of infinite $\mathfrak{S}$-paths starting from z. The set of all schedulers will be denoted by $\mathcal{S}$. A scheduler $\mathfrak{S} \in \mathcal{S}$ for MDP $\mathcal{M}$ induces a (potentially infinite) DTMC $\mathcal{M}^{\mathfrak{S}}$ where the states of $\mathcal{M}^{\mathfrak{S}}$, denoted $Z^{\mathfrak{S}}$, are the set of finite paths of $\mathcal{M}$ and the transition function $\Delta^{\mathfrak{S}}$ is as follows. For any two finite paths $\rho, \rho' \in Z^{\mathfrak{S}}$ where $\rho = z_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_m} z_m$ let

$$\Delta^{\mathfrak{S}}(\rho, \rho') = \begin{cases} \mu(z') & \text{if } \rho' \text{ is of the form } \rho \xrightarrow{\mathfrak{S}(\rho)} z' \text{ and } \Delta(z_m, \mathfrak{S}(\rho)) = \mu \\ 0 & \text{otherwise.} \end{cases}$$

This allows one to use probability measure over DTMCs to define a probability measure $\mathsf{prob}_z^{\mathfrak{S}}$ over the set of paths $\mathsf{Paths}_z^{\mathfrak{S}}(\mathcal{M})$. One can also define the expected cost of reaching a target set of states F with respect to a scheduler $\mathfrak{S}$, denoted $\mathbb{E}_z^{\mathfrak{S}}[\mathsf{cost}_z(F)]$, in a fashion similar to the DTMC case. Interested readers should refer to standard texts such as [10,52] for more details.

[1] One can alternatively define a scheduler as a function from finite paths into probability distributions on actions. Both definitions are equivalent in the context of PCTL model checking.

2.4.3 Probabilistic computation tree logic (PCTL)

Like DTMCs, properties of MDPs can be expressed in the logic PCTL. The semantics of PCTL formulae stay the same, except for the semantics of $\mathcal{P}_{\bowtie p}[\psi]$ and $\mathcal{E}_{\bowtie c}[\phi]$, which now require a quantification over all schedulers.

Definition 3 Let $\mathcal{M}$ be an MDP, ϕ be a state formula, and ψ be a path formula. The satisfaction relation $\models$ for PCTL state formulae is defined identically to Definition 2, except for the following cases.

$$\begin{array}{ll} \mathcal{M}, z \models \mathcal{P}_{\bowtie p}[\psi] & \Leftrightarrow \forall \mathfrak{S} \in \mathcal{S},\ p_z^{\mathfrak{S}}(\psi) \bowtie p \\ \mathcal{M}, z \models \mathcal{E}_{\bowtie c}[\phi] & \Leftrightarrow \forall \mathfrak{S} \in \mathcal{S},\ e_z^{\mathfrak{S}}(\phi) \bowtie c \end{array}$$

where given an adversary $\mathfrak{S} \in \mathcal{S}$, $p_z^{\mathfrak{S}}(\psi) = \mathsf{prob}_z^{\mathfrak{S}}(\{\rho \in \mathsf{Paths}_z^{\mathfrak{S}}(\mathcal{M}) \mid \mathcal{M}, \rho \models \psi\})$ and $e_z^{\mathfrak{S}}(\phi) = \mathbb{E}_z^{\mathfrak{S}}[\mathsf{cost}_z(Z_\phi)]$ with $Z_\phi = \{z' \in Z \mid \mathcal{M}, z' \models \phi\}$.

2.5 PCTL model checking for MDPs

Similar to the PCTL model checking algorithm for DTMCs, the PCTL model checking algorithm for MDPs recursively computes the set of states satisfying a state sub-formula (see [10,52] for the complete details). We illustrate the differences when we model check the probability and expected cost operators.

$\mathcal{P}_{\bowtie p}[\phi\ \mathcal{U}\ \phi']$ *operator.* For checking whether a state z_0 satisfies $\mathcal{P}_{\bowtie p}[\phi\ \mathcal{U}\ \phi']$, we recursively compute the sets of states Z_ϕ and $Z_{\phi'}$ as in the case of DTMCs. Given a state z, let $p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi') = \max\limits_{\mathfrak{S} \in \mathcal{S}} p_z^{\mathfrak{S}}(\phi\ \mathcal{U}\ \phi')$ and $p_z^{\mathsf{min}}(\phi\ \mathcal{U}\ \phi') = \min\limits_{\mathfrak{S} \in \mathcal{S}} p_z^{\mathfrak{S}}(\phi\ \mathcal{U}\ \phi')$. Thus, $p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$ (resp. $p_z^{\mathsf{min}}(\phi\ \mathcal{U}\ \phi')$) is the maximum (resp. minimum) probability of satisfying $\phi\ \mathcal{U}\ \phi'$. We note that both $p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$ and $p_z^{\mathsf{min}}(\phi\ \mathcal{U}\ \phi')$ exist [9,10,14,16,52]). Thus, in order to check whether $\mathcal{M}, z_0 \models \mathcal{P}_{\bowtie p}[\phi\ \mathcal{U}\ \phi']$, it suffices to compute $p_{z_0}^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$ when $\bowtie\ \in \{<, \leq\}$ and to compute $p_{z_0}^{\mathsf{min}}(\phi\ \mathcal{U}\ \phi')$ if $\bowtie\ \in \{>, \geq\}$. We explain below how these are computed.

In order to compute $p_{z_0}^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$, we compute the function $\lambda z.\ p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$ that maps each $z \in Z$ to $p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$. For each $z \in Z$, pick a variable y_z. Consider the following linear optimization problem:

$$\begin{array}{lll} & \min \sum\limits_{z \in Z} y_z \text{ subject to} & \\ y_z = 0 & & \text{if } z \in \mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \\ y_z = 1 & & \text{if } z \in \mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \\ y_z \geq \sum\limits_{z' \in Z} \Delta(z, \alpha, z') \cdot y_{z'} & & \text{if } \begin{array}{l} z \in Z \backslash (\mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \cup \mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']) \\ \alpha \in \mathsf{enabled}(z) \end{array} \end{array} \tag{3}$$

where $\mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']$ ($\mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']$ respectively) is the set of states z such that $p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$ is 0 (1 respectively). The sets $\mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']$ and $\mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']$ can be computed using graph-theoretic algorithms. Now, the vector $\lambda z.\ p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$ is the *unique* solution set for this linear optimization problem, ie, objective is minimized and constraints satisfied if and only if we replace y_z by $p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$.

Computation of $\lambda z.\ p_z^{\mathsf{min}}(\phi\ \mathcal{U}\ \phi')$, the state-indexed vector that maps $z \in Z$ to $p_z^{\mathsf{min}}(\phi\ \mathcal{U}\ \phi')$, is along similar lines; the objective changes to maximization, $\mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']$ and $\mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']$ are replaced by $\mathsf{Prob}_0^{\mathsf{min}}[\phi\ \mathcal{U}\ \phi']$ and $\mathsf{Prob}_1^{\mathsf{min}}[\phi\ \mathcal{U}\ \phi']$ respectively, and the direction the last inequality is reversed. Here $\mathsf{Prob}_0^{\mathsf{min}}[\phi\ \mathcal{U}\ \phi']$ ($\mathsf{Prob}_1^{\mathsf{min}}[\phi\ \mathcal{U}\ \phi']$ respectively) is the set of states z for which p_z^{min} is 0 (1 respectively), and can be computed using graph-theoretic algorithms.

$\mathcal{E}_{\bowtie p}[\phi]$ *operator.* For checking whether a state z_0 satisfies $\mathcal{E}_{\bowtie p}[\phi]$, we recursively compute the set of states Z_ϕ as in the case of DTMCs. Given a state z, let $e_z^{\mathsf{max}}(\phi) = \max_{\mathfrak{S}\in\mathcal{S}} e_z^{\mathfrak{S}}(\phi)$ and $e_z^{\mathsf{min}}(\phi) = \min_{\mathfrak{S}\in\mathcal{S}} e_z^{\mathfrak{S}}(\phi)$. Thus, $e_z^{\mathsf{max}}(\phi)$ ($e_z^{\mathsf{min}}(\phi)$ respectively) is the maximum (minimum respectively) expected cost of reaching the set Z_ϕ. Again, we note that both $e_z^{\mathsf{max}}(\phi)$ and $e_z^{\mathsf{min}}(\phi)$ exist [10,14,52]). Thus, it suffices to compute $e_{z_0}^{\mathsf{max}}(\phi)$ when $\bowtie\ \in \{<, \leq\}$ and to compute $e_{z_0}^{\mathsf{min}}(\phi)$ if $\bowtie\ \in \{>, \geq\}$.

In order to compute $e_{z_0}^{\mathsf{max}}(\phi)$, we compute the state-indexed vector $\lambda z.\, e_z^{\mathsf{max}}(\phi)$. For each $z \in Z$, pick a variable y_z. Consider the following linear optimization problem (with the convention that $0 \cdot \infty = 0$):

$$\begin{array}{ll} \min \sum_{z\in Z} y_z \text{ subject to} & \\ y_z = 0 & \text{if } z \in Z_\phi \\ y_z = \infty & \text{iff } z \in \mathsf{Cost}_\infty^{\mathsf{max}}[\phi] \\ y_z \geq \mathbf{C}(z,\alpha) + \sum_{z'\in Z} \Delta(z,\alpha,z') \cdot y_{z'} & \text{if } z \in Z \backslash (Z_\phi \cup \mathsf{Cost}_\infty^{\mathsf{max}}[\phi]), \alpha \in \mathsf{enabled}(z) \end{array} \tag{4}$$

where $\mathsf{Cost}_\infty^{\mathsf{max}}$ is the set of states z such that $e_z^{\mathsf{max}}(\phi) = \infty$. Observe that $z \in \mathsf{Cost}_\infty^{\mathsf{max}}$ if and only if there is a scheduler $\mathfrak{S}$ such that $p_z^{\mathfrak{S}}(\phi) < 1$. This allows computation of the set $\mathsf{Cost}_\infty^{\mathsf{max}}$ using graph-theoretic methods. Now, the vector $\lambda z.\, e_z^{\mathsf{max}}(\phi)$ is the *unique* solution for this linear optimization problem, i.e., the objective is minimized *and* constraints satisfied if and only if we replace y_z by $e_z^{\mathsf{max}}(\phi)$. Computation of $\lambda z.\, e_z^{\mathsf{min}}(\phi)$ is along similar lines; the objective changes to maximization, $\mathsf{Cost}_\infty^{\mathsf{max}}[\phi]$ is replaced by $\mathsf{Cost}_\infty^{\mathsf{min}}[\phi]$, and the direction the last inequality is reversed. Here $\mathsf{Cost}_\infty^{\mathsf{min}}[\phi]$ is the set of states z such that e_z^{min} is ∞. Observe that $z \in \mathsf{Cost}_\infty^{\mathsf{min}}[\phi]$ if and only if $p_z^{\mathfrak{S}}(\phi) < 1$, for all schedulers $\mathfrak{S}$. The set $\mathsf{Cost}_\infty^{\mathsf{min}}[\phi]$ can also be computed graph-theoretically.

3 Approximate model checking

As discussed before, solving quantitative properties of DTMCs and MDPs by a reduction to linear programming does not scale well enough to make it a viable solution technique in practice. As a result, techniques for approximating solutions to the model checking problem using floating-point arithmetic have been widely adopted. In this section, we describe two such techniques, value iteration and interval iteration, and demonstrate how each approach can produce incorrect solutions.

3.1 Iterative techniques

The linear program described in Eq. (1) for DTMCs can equivalently be expressed in the below, for some appropriate matrix A and vector b.

$$\bar{x} = A\bar{x} + b$$

This allows for an alternate approach to solving the linear program from Eq. (1) known as *value iteration*. In the case of DTMCs, the unique solution to Eq. (1) can be computed iteratively as the the limit of the following sequence.

$$\bar{x}_0(z) = \begin{cases} 1 & \text{if } z \in \mathsf{Prob}_1[\phi\ \mathcal{U}\ \phi'] \\ 0 & \text{otherwise .} \end{cases}$$
$$\bar{x}_{i+1} = A\bar{x}_i + \bar{b} \tag{5}$$

For the case of MDPs, the unique solution that minimizes the objective function of the linear program in Eq. (3) and used to compute maximum probabilities of satisfying $[\phi\ \mathcal{U}\ \phi']$ can be obtained as the limit of the iterative sequence $\{x_i\}_{i\geq 0}$:

$$\bar{x}_0(z) = \begin{cases} 1 & \text{if } z \in \mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \\ 0 & \text{otherwise .} \end{cases}$$
$$\bar{x}_{i+1}(z) = \begin{cases} 1 & \text{if } z \in \mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \\ 0 & \text{if } z \in \mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \\ \max\{\sum_{z'\in Z} \Delta(z,\alpha,z')\cdot \bar{x}_i(z') \mid \alpha{\in}\mathsf{enabled}(z)\} & \text{otherwise} \end{cases} \tag{6}$$

For the solution to the linear program that is used to compute minimum probabilities, the iterative sequence is similar except that max is replaced by min. The iterative sequences for computing expected costs can be similarly defined with one notable variation. For computing min expected costs, the MDPs have to be transformed to get rid of cost 0 cycles. We refer the reader to [9,28,52] for details.

In many cases, the sequence does not converge in a finite number of steps, and therefore model checkers terminate the sequence when successive vectors v_k and v_{k+1} become “close enough”. The choice of stopping criterion is based mainly on heuristics. The PRISM model checker, for example, implements two criteria (i) *absolute convergence*, and (ii) *relative convergence*. Under the absolute criterion, value iteration terminates if the norm $\|v_{k+1} - v_k\| < \epsilon$ for some $\epsilon > 0$. Under the relative criterion, termination occurs when $\frac{\|v_{k+1}-v_k\|}{\|v_k\|} < \epsilon$. In spite of the fact that iterative techniques only approximate solutions, value iteration remains the popular choice for widely used tools that analyze PCTL properties as it vastly outperforms linear programming techniques, despite their theoretically better asymptotic complexity.

As originally observed in [27], value iteration provides no guarantees about the quality of the solution, regardless of the stopping criterion used. To help rectify this problem, Haddad et. al. [31] and Brázdil et. al. [17] concurrently introduced *interval iteration* for computing min/max reachability probabilities in DTMCs and MDPs. In this approach, one simultaneously computes two sequences of vectors, one converging to the solution from below and one converging to the exact solution from above. In this setting, the stopping criterion becomes straightforward; terminate when the distance between the two vectors is within some ϵ threshold. Assuming the absence of floating-point errors, this effectively gives a small ϵ-neighborhood that contains the actual solution. In order to achieve convergence, interval iteration requires a pre-processing step that transforms the underlying graph of the model. The interval iteration technique was extended to expected costs in [11].

Both iterative techniques described above can be further enhanced by performing arithmetic operations using *Multi-terminal binary decision diagrams* (MTBDDs) [29,36]. MTBDDs generalize BDDs [18] by allowing terminal values to be different from 0 or 1. Similar to the role of BDDs in symbolic model checking [45], MTBDD based model checkers leverage the performance benefit due to the succinct representations of the data structures involved.

3.2 Shortcomings of iterative techniques

When computing constrained reachability probabilities using value iteration, both the absolute and relative convergence criteria can result in solutions that are very far from the actual answers. In [31], the authors give a DTMC and a PCTL property whose solution is $\frac{1}{2}$, yet PRISM reports 9.77×10^{-4} for the absolute criterion and 0.198 for the relative criterion. This drastic error is the result of a premature termination of value iteration. Several other sources of imprecision can also cause state-of-the-art quantitative model checkers to produce unsound results. For example, consider a PCTL formula of the form $\mathcal{P}_{\geq p}(\psi)$ and a system $\mathcal{M}$ such that the probability measure of the formula ψ is exactly p. When value iteration, with floating-point numbers, is used to compute this measure, the value p may only be approached in the limit, and hence the procedure will return some p' that approximates p from below. This means that the formula $\mathcal{P}_{\geq p}(\psi)$ will evaluate to false, where of course the correct value is true. This phenomenon was first pointed out in [54]. We also demonstrate a similar phenomenon with the DTMC from Example 1. For the sake of illustration, let $\mathrm{E}_s = \frac{1}{2}$. Clearly, from the initial state, the probability of reaching a state where sensor 1 fails is exactly $\frac{1}{2}$ and hence the formula $\mathcal{P}_{<\frac{1}{2}}$ [$\mathsf{true}\ \mathcal{U}\ (e_1^s{=}1)$] evaluates to false for the initial state. However, PRISM returns true. Errors such as these can be compounded in PCTL formulas containing nested operators, wherein the recursive step of the model checking algorithm returns an incorrect set of states. This can lead to substantial logical errors in model analysis, as we demonstrate with the example below.

Example 3 Let us instantiate the DTMC from Example 1 with $n = 14$ sensors, $m = 1$ actuator, MAX_FAILURES=1 and with $\mathrm{E}_s = \mathrm{E}_a = \frac{1}{2}$. Recall the third PCTL property of the embedded control system given in Example 2:

$$\mathcal{P}_{\bowtie' p}\ [\ \mathcal{P}_{\bowtie \frac{1}{2}}[\mathsf{true}\ \mathcal{U}\ (e_1^s{=}1)]\ \mathcal{U}\ \mathcal{P}_{\leq 0}[\mathsf{true}\ \mathcal{U}\ (e_1^a{=}1)]\ ].$$

When $\bowtie$ is $\leq$, the PRISM model checker returns "0.7096993582589287″ as the probability for the initial state with both value iteration and interval iteration[2]. With our tool RationalSearch, one can verify that the correct probability is 212895/229376, or "0.9281485421316964″. Further, when $\bowtie$ is $<$, PRISM again returns the value given above for both iterative techniques. This time, the actual solution, as generated by RationalSearch, is 0. The errors above are the result of the fact that PRISM incorrectly computes the set of states satisfying $\mathcal{P}_{\bowtie \frac{1}{2}}[\mathsf{true}\ \mathcal{U}\ (e_1^s{=}1)]$. This error in the recursive step results in an incorrect formulation of the constraints in the outermost constrained reachability problem.

When using interval iteration, we may be unable to conclude whether the DTMC or the MDP being model checked satisfies the given formula. For example, when checking whether a DTMC $\mathcal{M}$ satisfies a formula $\mathcal{P}_{\geq p}(\psi)$, we cannot provide a definite answer if the interval iteration returns that the probability of satisfying ψ lies in the interval (a, b) where $p \in (a, b)$.

4 Fixpoint formulations for constrained reachability and expected costs

As discussed in Sect. 2.3, the probability, associated with each state z in a DTMC, of satisfying a PCTL path formula $\phi\ \mathcal{U}\ \phi'$ can be characterized as the unique solution to a system of linear

[2] Using the Hybrid engine, the absolute convergence criterion and $\epsilon = 10^{-16}$.

equations. Similarly, the expected cost of reaching some state satisfying ϕ in a DTMC $\mathcal{M}$ starting from any given state z in $\mathcal{M}$ can also be characterized as the unique solution to a linear program. In both these cases, the solution can be seen as the unique fix point of a linear transformation. Thus, when given a candidate solution for the collection of probabilities (or the collection of expected costs), we can check the correctness of this collection by *plugging* the candidate solution in the corresponding system of equations. In the case of MDPs, the max probabilities, min probabilities, max expected and min expected costs can also be characterized as a solution to a system of equations. For MDPs, however, the system of equations may have multiple solutions. We will show below that when the system of equations for MDPs is not guaranteed to have a unique solution, we can perform an additional graph-theoretic check to confirm that a given candidate solution for the set of probabilities (or the set of expected costs) is correct. Such a confirmation check, as we will discuss in Sect. 5, is crucial to our algorithm for computing exact answers.

4.1 Fixpoint formulation for constrained reachability in MDPs

Let $\mathcal{M} = (Z, \mathsf{Act}, \Delta, \mathbf{C}, L)$ be an MDP and ϕ, ϕ' be PCTL state formulas. The state-indexed vector $P^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi') = \lambda z.\ p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$ can be characterized as the *least fix point* (least under pointwise ordering) of the set of equations:

$$\begin{array}{lll} y_z = 0 & & \text{if } z \in \mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \\ y_z = 1 & & \text{if } z \in \mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \\ y_z = \displaystyle\max_{\alpha \in \mathsf{enabled}(z)} \sum_{z' \in Z} \Delta(z, \alpha, z') \cdot y_{z'} & & \text{if } z \in Z \backslash (\mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \cup \mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']) \end{array} \tag{7}$$

The state-indexed vector $P^{\mathsf{min}}(\phi\ \mathcal{U}\ \phi') = \lambda z.\ p_z^{\mathsf{min}}(\phi\ \mathcal{U}\ \phi')$ can be similarly characterized by replacing max by min. For min, the fix point, in fact, turns out to be *unique* [9]. For max, the fix point is not unique, although several references claim this to be case (see Theorem 10.100 in [10] for example). The non-uniqueness has also been pointed out by [31]. However, for the max case, we show that a simple graph-theoretic check can be performed to verify if a given fix point to the set of equations is indeed the exact solution P^{max}. We describe this below. We shall need the notion of a *memoryless scheduler*, namely a scheduler that assigns the same action to any two finite paths ending in the same state (see [9,10,52]). A memoryless scheduler $\mathfrak{S}_{\mathsf{V}}$ can be considered as a function from states to actions instead of a function from paths to actions.

Let $\mathsf{V} : Z \to [0, 1]$ be a solution of the set of equations in Eq. (7). We start by defining a directed graph that is obtained from $\mathcal{M}$ by *selecting*, for each state, the set of actions that potentially achieve the maximum reachability probabilities.

Definition 4 Let $\mathsf{V} : Z \to [0, 1]$ be a fix point of Eq. (7). Let $Z^? = Z \backslash (\mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \cup \mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'])$ For each state $z \in Z^?$, let

$$\mathsf{argmax}_z^{\mathsf{V}} = \{\alpha \in \mathsf{enabled}(z) \mid \mathsf{V}(z) = \sum_{z' \in Z} \Delta(z, \alpha, z') \cdot \mathsf{V}(z')\}.$$

Let $\mathcal{G}_{\mathsf{V}} = (Z, E)$ be a directed graph such that $(z_1, z_2) \in E$ iff $z_1 \in Z^?$ and $\exists \alpha \in \mathsf{argmax}_{z_1}^{\mathsf{V}}$ such that $\Delta(z_1, \alpha, z_2) > 0$.

With the above definition of the graph $\mathcal{G}_{\mathsf{V}}$, we can characterize the solution to the max reachability problem for MDPs as follows.

Lemma 1 *Let $\mathcal{M} = (Z, Act, \Delta, \mathbf{C}, L)$ be a MDP and ϕ, ϕ' be PCTL state formulas. For each state z of $\mathcal{M}$, let $p_z^{max}(\phi\ \mathcal{U}\ \phi')$ be the maximum probability of satisfying $\phi\ \mathcal{U}\ \phi'$. Let $\mathsf{V} : Z \to [0, 1]$ be a solution of the set of equations given by Eq. (7). Consider $\mathcal{G}_{\mathsf{V}}$ as defined in Definition 4 above. Let Z_0 be the set of states z such that there is no path from z to any state $z' \in Prob_1^{max}[\phi\ \mathcal{U}\ \phi']$ in the graph $\mathcal{G}_{\mathsf{V}}$. Then,*

$$Z_0 = Prob_0^{max}[\phi\ \mathcal{U}\ \phi'] \Leftrightarrow \forall z \in Z.\, \mathsf{V}(z) = p_z^{max}(\phi\ \mathcal{U}\ \phi').$$

Proof If $Z = \mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \cup \mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']$ then the lemma is immediate. So we will consider the case that $Z \backslash (\mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \cup \mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']) \neq \emptyset$. Note also that we have that for each state $z \in Z$, $\mathsf{V}(z) \geq p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$ as the state-indexed vector $P^{\mathsf{max}} = \lambda z.\, p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$ is the least fix point of Eq. (7).

It can be easily shown that in order to establish the Lemma, we can assume that ϕ is true, ϕ' is some $a \in \mathsf{AP}$, $\mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']$ and $\mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']$ consists of exactly one state (say rej and acc respectively), exactly one action α_0 is enabled in rej and acc, $\Delta(\mathsf{rej}, \alpha_0) = \mathsf{rej}$, and $\Delta(\mathsf{acc}, \alpha_0) = \mathsf{acc}$.

($\Rightarrow$) It suffices to show that $\mathsf{V}(z) \leq p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$ for each state $z \in Z$. Let $Z^? = Z \backslash (\mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \cup \mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'])$. Let m be the cardinality of $Z^?$. From the fact that $\mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] = Z_0$, we can construct inductively an enumeration $z_1, \ldots, z_m$ of states in $Z^?$ and an enumeration of actions $\alpha_1, \ldots, \alpha_m$ in Act such that for each $1 \leq i \leq m$,

1. $\alpha_i \in \mathsf{argmax}_{z_i}^{\mathsf{V}}$, and
2. $\Delta(z_i, \alpha_i, z) \neq 0$ for some $z \in \{\mathsf{acc}, z_1, \ldots, z_{i-1}\}$.

Consider the memoryless scheduler $\mathfrak{S}_{\mathsf{V}}$ for the MDP $\mathcal{M}$, that picks α_i when the last state in the execution is $z_i \in Z^?$ and picks α_0 otherwise. By definition, $\mathsf{prob}_z^{\mathfrak{S}_{\mathsf{V}}}(\mathsf{true}\ \ \mathcal{U}\ \ a) \leq p_z^{\mathsf{max}}(\mathsf{true}\ \ \mathcal{U}\ \ a)$ for each $z \in Z$. Thus, it suffices to show that $\mathsf{prob}_z^{\mathfrak{S}_{\mathsf{V}}}(\mathsf{true}\ \ \mathcal{U}\ \ a) = \mathsf{V}(z)$ for each $z \in Z$.

Let us now construct a DTMC, $\mathcal{M}_0$, from $\mathcal{M}$ which picks for each state z, the action $\mathfrak{S}_{\mathsf{V}}(z)$. Formally, the DTMC $\mathcal{M}_0 = (Z, \Delta_0, \mathbf{C}, L)$ where $\Delta_0(z, z') = \Delta(z, \mathfrak{S}_{\mathsf{V}}(z), z')$ for all $z, z' \in Z$. It is easy to see that $\mathsf{prob}_z^{\mathfrak{S}_{\mathsf{V}}}(\mathsf{true}\ \mathcal{U} a)$ is the probability that z satisfies the formula $\mathsf{true}\ \mathcal{U} a$ in $\mathcal{M}_0$. By construction of $\mathcal{M}_0$, this probability is 0 (1 respectively) if and only if z is rej (acc respectively). Thus, $\{\mathsf{prob}_z^{\mathfrak{S}_{\mathsf{V}}}(\mathsf{true}\ \ \mathcal{U}\ \ a)\}_{z \in Z}$ is the *unique* solution of the set of equations:

$$\begin{aligned} x_{\mathsf{rej}} &= 0 \\ x_{\mathsf{acc}} &= 1 \\ x_z &= \sum_{z' \in Z} \Delta(z, \mathfrak{S}_{\mathsf{V}}(z), z') \cdot x_{z'} \qquad \text{otherwise.} \end{aligned} \tag{8}$$

As $\alpha_i \in \mathsf{argmax}_{z_i}^{\mathsf{V}}$, we get by construction, V is also a solution to Eq. (8). By uniqueness, we must have that $\mathsf{prob}_z^{\mathfrak{S}_{\mathsf{V}}}(\mathsf{true}\ \ \mathcal{U}\ \ a) = \mathsf{V}(z)$ for each $z \in Z$.

($\Leftarrow$) The maximum probability of reaching acc is realized by a *memoryless scheduler*, namely a scheduler that assigns the same action to any two finite paths ending in the same state (see [9,10,52]). Fix one such scheduler $\mathfrak{S}$. We have that for all states $z \in Z, \mathsf{V}(z) = p_z^{\mathsf{max}}(\mathsf{true}\ \mathcal{U}\ a) = \mathsf{prob}_z^{\mathfrak{S}}(\mathsf{true}\ \mathcal{U}\ a)$. From this, it is easy to show that the following hold:

1. $\mathfrak{S}(z) \in \mathsf{argmax}_{z,\mathsf{V}}$ for all $z \in Z \backslash \{\mathsf{acc}, \mathsf{rej}\}$.
2. For each $z \in Z \backslash \{\mathsf{acc}, \mathsf{rej}\}$, there is a finite path $\rho = z_1' \xrightarrow{\mathfrak{S}(z_1')} \cdots \xrightarrow{\mathfrak{S}(z_{\ell-1}')} z_\ell'$ such that $z_1' = z$ and $z_\ell' = \mathsf{acc}$.

From the above two observations, we have that $\mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] = Z_0$. □

4.2 Fixpoint formulation for expected costs in MDPs.

Let $\mathcal{M} = (Z, \mathsf{Act}, \Delta, \mathbf{C}, L)$ be an MDP and ϕ be a PCTL state formula. The state-indexed vector $E^{\mathsf{max}} = \lambda z.\, e_z^{\mathsf{max}}(\phi)$ can be characterized as a fix point of the following set of equations [28] (with the convention that $0 \cdot \infty = 0$):

$$\begin{aligned} y_z &= 0 && \text{if } z \in Z_\phi \\ y_z &= \infty && \text{if } z \in \mathsf{Cost}_\infty^{\mathsf{max}}[\phi] \\ y_z &= \max_{\alpha \in \mathsf{enabled}(z)} \mathbf{C}(z, \alpha) + \sum_{z' \in Z} \Delta(z, \alpha, z') \cdot y_{z'} && \text{otherwise} \end{aligned} \tag{9}$$

While E^{max} is described to be the least fix point of Eq. (9) in [28], we, in fact, show that Eq. (9) admits only one solution.

Lemma 2 *Let $\mathcal{M} = (Z, \mathit{Act}, \Delta, \mathbf{C}, L)$ be a MDP and ϕ be a PCTL state formula. Let Z_ϕ be the set of states of $\mathcal{M}$ that satisfy ϕ. For each state z of $\mathcal{M}$, let $e_z^{max}(\phi)$ be the maximum expected cost of reaching the set of states Z_ϕ. Then $E^{max} = \lambda z.\, e_z^{max}(\phi)$ is the unique solution to Eq. (9).*

Proof We only need to show that Eq. (9) has a unique solution. Let V^1 and V^2 be two solutions of Eq. (9). Observe that $\mathsf{V}^1(z) = \mathsf{V}^2(z)$ for each $z \in Z_\phi \cup \mathsf{Cost}_\infty^{\mathsf{max}}[\phi]$. Let $U = Z \backslash (Z_\phi \cup \mathsf{Cost}_\infty^{\mathsf{max}}[\phi])$ and $d = \mathsf{max}_{z \in U} |\mathsf{V}^1(z) - \mathsf{V}^2(z)|$. It suffices to show that $d = 0$.

We will establish the result reductio ad absurdum. Assume $d > 0$. By definition, each state $z \in U$ does not belong to the set $\mathsf{Cost}_\infty^{\mathsf{max}}[\phi]$. This implies that for all schedulers $\mathfrak{S}$ and state $z \in U$, $p_z^{\mathfrak{S}}(\mathsf{true}\ \mathcal{U}\ \phi) = 1$. This leads to the following observations:

1. For $z \in U$ and $\alpha \in \mathsf{enabled}(z)$, probability of transitioning from z on action α to each state in $\mathsf{Cost}_\infty^{\mathsf{max}}[\phi]$ is 0.
2. For each $k = 1, 2, z \in U$ and $\alpha \in \mathsf{enabled}(z)$, let $v_z^{k,\alpha} = \mathbf{C}(z, \alpha) + \sum_{z' \in U} \Delta(z, \alpha, z') \cdot \mathsf{V}^k(z')$.

   By definition and the previous observation, $\mathsf{V}^k(z) = \max_{\alpha \in \mathsf{enabled}(z)} v_z^{k,\alpha}$.
3. There is an enumeration $z_1, \ldots, z_n$ of states in U such that for each $1 \le i \le n$ and action α, if $\alpha \in \mathsf{enabled}(z_i)$ then $\Delta(z_i, \alpha, z) \neq 0$ for some state $z \in Z_\phi \cup \{z_j \mid 1 \le j < i\}$.

Claim $|\mathsf{V}^1(z_i) - \mathsf{V}^2(z_i)| < d$ for each $1 \le i \le n$.

Proof The proof proceeds by induction on i.
Base case Fix $\alpha_0 \in \mathsf{enabled}(z_1)$. By construction of z_1, $\sum_{z' \in U} \Delta(z_1, \alpha_0, z') < 1$.

We have that for each $k = 1, 2$,

$$\begin{aligned} v_{z_1}^{k,\alpha_0} &= \mathbf{C}(z_1, \alpha_0) + \sum_{z' \in U} \Delta(z_1, \alpha_0, z') \cdot \mathsf{V}^k(z') \\ &= \mathbf{C}(z_1, \alpha_0) + \sum_{z' \in U} \Delta(z_1, \alpha_0, z') \cdot (\mathsf{V}^k(z') - \mathsf{V}^{3-k}(z') + \mathsf{V}^{3-k}(z')) \\ &= \mathbf{C}(z_1, \alpha_0) + \sum_{z' \in U} \Delta(z_1, \alpha_0, z') \cdot \mathsf{V}^{3-k}(z') \\ &\quad + \sum_{z' \in U} \Delta(z_1, \alpha_0, z') \cdot (\mathsf{V}^k(z') - \mathsf{V}^{3-k}(z')) \\ &= v_{z_1}^{3-k,\alpha_0} + \sum_{z' \in U} \Delta(z_1, \alpha_0, z') \cdot (\mathsf{V}^k(z') - \mathsf{V}^{3-k}(z')) \end{aligned}$$

$$
\begin{aligned}
&\le v_{z_1}^{3-k,\alpha_0} + \sum_{z' \in U} \Delta(z_1, \alpha_0, z') \cdot d \\
&\le v_{z_1}^{3-k,\alpha_0} + d \cdot \sum_{z' \in U} \Delta(z_1, \alpha_0, z') \\
&< v_{z_1}^{3-k,\alpha_0} + d \cdot 1 \le \max_{\alpha \in \mathsf{enabled} z_1} v_{z_1}^{3-k,\alpha_0} + d = \mathsf{V}^{3-k}(z_1) + d.
\end{aligned}
$$

Since α_0 is an arbitrary action in $\mathsf{enabled}(z_1)$, we get that

$$\mathsf{V}^k(z_1) < \mathsf{V}^{3-k}(z_1) + d \text{ for each } k \in \{1, 2\}.$$

Thus, both $\mathsf{V}^1(z_1) - \mathsf{V}^2(z_1) < d$ and $\mathsf{V}^2(z_1) - \mathsf{V}^1(z_1) < d$ establishing the base case.

Induction step Assume that we have $|\mathsf{V}^1(z_i) - \mathsf{V}^2(z_i)| < d$ for each $1 \le i \le \ell$. Now, consider $z_{\ell+1}$ and fix $\alpha_0 \in \mathsf{enabled}(z_{\ell+1})$. By construction of $z_{\ell+1}$,

- Either $\sum_{z' \in U} \Delta(z_{\ell+1}, \alpha_0, z') < 1$
- Or $\Delta(z_{\ell+1}, \alpha_0, z_j) > 0$ for some $1 \le j \le \ell$.

If $\sum_{z' \in U} \Delta(z_{\ell+1}, \alpha_0, z') < 1$ then we can show by an argument similar to the one used in base case that

$$v_{z_{\ell+1}}^{k,\alpha_0} < v_{z_{\ell+1}}^{3-k} + d \text{ for each } k = 1, 2.$$

Now, consider the case when $\Delta(z_{\ell+1}, \alpha_0, z_j) > 0$ for some $1 \le j \le \ell$. Fix one such j_0. Thus, we have $\Delta(z_{\ell+1}, \alpha_0, z_{j_0}) > 0$. By Induction hypothesis, we also have that $|\mathsf{V}^1(z_{j_0}) - \mathsf{V}^2(z_{j_0})| < d$. For each $k = 1, 2$,

$$
\begin{aligned}
v_{z_{\ell+1}}^{k,\alpha_0} &= v_{z_{\ell+1}}^{3-k,\alpha_0} + \sum_{z' \in U} \Delta(z_{\ell+1}, \alpha_0, z') \cdot (\mathsf{V}^k(z') - \mathsf{V}^{3-k}(z')) \\
&= v_{z_{\ell+1}}^{3-k,\alpha_0} + \Delta(z_{\ell+1}, \alpha_0, z_{j_0}) \cdot (\mathsf{V}^k(z_{j_0}) - \mathsf{V}^{3-k}(z_{j_0})) \\
&\quad + \sum_{z' \in U \setminus \{z_{j_0}\}} \Delta(z_{\ell+1}, \alpha_0, z') \cdot (\mathsf{V}^k(z') - \mathsf{V}^{3-k}(z')) \\
&\le v_{z_{\ell+1}}^{3-k,\alpha_0} + \Delta(z_{\ell+1}, \alpha_0, z_{j_0}) \cdot (\mathsf{V}^k(z_{j_0}) - \mathsf{V}^{3-k}(z_{j_0})) \\
&\quad + \sum_{z' \in U \setminus \{z_{j_0}\}} \Delta(z_{\ell+1}, \alpha_0, z') \cdot d \\
&< v_{z_{\ell+1}}^{3-k,\alpha_0} + \Delta(z_{\ell+1}, \alpha_0, z_{j_0}) \cdot d + \sum_{z' \in U \setminus \{z_{j_0}\}} \Delta(z_{\ell+1}, \alpha_0, z') \cdot d \\
&= v_{z_{\ell+1}}^{3-k,\alpha_0} + d \cdot \sum_{z' \in U} \Delta(z_{\ell+1}, \alpha_0, z') \\
&= v_{z_{\ell+1}}^{3-k,\alpha_0} + d \cdot 1 \le \mathsf{V}^{3-k}(z_{\ell+1}) + d.
\end{aligned}
$$

Since α_0 is an arbitrary action in $\mathsf{enabled}(z_{\ell+1})$, we get once again that

$$\mathsf{V}^k(z_{\ell+1}) < \mathsf{V}^{3-k}(z_{\ell+1}) + d \text{ for each } k \in \{1, 2\}.$$

Thus, we get both $\mathsf{V}^1(z_{\ell+1}) - \mathsf{V}^2(z_{\ell+1}) < d$ and $\mathsf{V}^2(z_{\ell+1}) - \mathsf{V}^1(z_{\ell+1}) < d$ establishing the induction step.
(End: Proof of claim) □

Thus, we have that $d = \mathsf{max}_{z \in U} |\mathsf{V}^1(z) - \mathsf{V}^2(z)| < d$, which is a contradiction. □

The state-indexed vector $E^{\mathsf{min}}(\phi) = \lambda z.\, e_z^{\mathsf{min}}(\phi)$ can also be characterized as a fix point of the following set of equations [9,28]:

$$\begin{array}{lll} y_z &= 0 & \text{if } z \in Z_\phi \\ y_z &= \infty & \text{iff } z \in \mathsf{Cost}_\infty^{\mathsf{min}}[\phi] \\ y_z &= \mathsf{min}_{\alpha \in \mathsf{enabled}(z)} \mathbf{C}(z,\alpha) + \sum_{z' \in Z} \Delta(z,\alpha,z') \cdot y_{z'} & \text{otherwise} \end{array} \tag{10}$$

However, in this case, the fix point may not be unique. $E^{\mathsf{min}}(\phi)$ is the greatest fix point of Eq. (10) [9]. Nevertheless, we can perform an additional check to see if a given solution of Eq. (10) is indeed the function $E^{\mathsf{min}}(\phi)$.

Let $\mathsf{V} : Z \rightarrow \mathbb{Q}^{\geq 0}$ be a solution of the set of equations given by Eq. (10). We start by defining a directed graph that is obtained from $\mathcal{M}$ by *selecting* for each state, the set of actions that potentially achieve the minimum expected costs.

Definition 5 Let $\mathsf{V} : Z \rightarrow \mathbb{Q}^{\geq 0}$ be a fix point of Eq. (10). For each state $z \in Z \backslash (Z_\phi \cup \mathsf{Cost}_\infty^{\mathsf{min}}[\phi])$, let

$$\mathsf{argmin}_z^{\mathsf{V}} = \{\alpha \in \mathsf{enabled}(z) \mid \mathsf{V}(z) = \mathbf{C}(z,\alpha) + \sum_{z' \in Z} \Delta(z,\alpha,z') \cdot \mathsf{V}(z')\}.$$

Let $\mathcal{H}_{\mathsf{V}} = (Z, E)$ be a directed graph such that $(z_1, z_2) \in E$ iff $z_1 \in Z \backslash (Z_\phi \cup \mathsf{Cost}_\infty^{\mathsf{min}}[\phi])$ and $\exists \alpha \in \mathsf{argmin}_{z_1}^{\mathsf{V}}$ st $\Delta(z_1, \alpha, z_2) > 0$.

The following can be proved along the same lines as Lemma 1:

Lemma 3 *Let $\mathcal{M} = (Z, Act, \Delta, \mathbf{C}, L)$ be a MDP and ϕ be a PCTL state formula. For each state z of $\mathcal{M}$, let $e_z^{min}(\phi)$ be the minimum expected cost of reaching the set Z_ϕ. Let $V : Z \rightarrow \mathbb{Q}^{\geq 0}$ be a solution of the set of equations given by Eq. (10). Consider $\mathcal{H}_V$ as defined in Definition 5 above. Let Z_∞ be the set of states z such that there is no path from z to any state $z' \in Z_\phi$ in the graph $\mathcal{H}_V$. Then*

$$Z_\infty = Cost_\infty^{min}[\phi] \Leftrightarrow \forall z \in Z.\, V(z) = e_z^{min}(\phi).$$

5 Exact model checking

As demonstrated in Sect. 3, approximate solution techniques can lead to unreliable results and the incorrect analysis of systems. To rectify this serious limitation, tools such as PRISM and STORM have implemented exact model checking engines, which make heavy use of techniques from parametric model checking [21,23,33,34]. The idea behind these engines is to interpret the probabilistic model (DTMC or MDP) as a finite automaton in which transitions probabilities are described by letters of an alphabet. When one is interested in costs, states are additionally labeled by a cost structure. Using techniques derived from state elimination [37], one can then calculate a regular expression representing the language of this automaton. The core idea of this translation is to eliminate a state s by increasing the probability of moving from each predecessor s_1 of s to each successor s_2 of s by the probability of moving from s_1 to s_2 when passing through s. In the case of parametric model checking, various techniques can then be used to translate the regular expression into a rational function over the parameters of the model. When using this approach for exact model checking, one can likewise derive a parameter-free function that describes the property in question.

Although they rectify the problems with approximation techniques, the exact quantitative model checking engines implemented in tools like PRISM and STORM don't scale as well as their iterative counterparts. See Example 4 below and Sect. 7 for a complete analysis. The goal of our technique, to which the remainder of this section is dedicated, is to utilize the advantages of fast approximate model checking techniques to produce exact solutions.

Example 4 Again consider the DTMC modeling an embedded control system with the parameters given in Example 3. To guarantee the correctness of one's analysis, exact solution techniques must be employed. Unfortunately, the exact model checking engines of PRISM and STORM do not scale well enough to analyze this example, which contains about 4.8 million states and about 44 million transitions. Under our test setup (see Sect. 7), both tools reached a 30-min timeout when trying to analyze the properties from Example 3. On the other hand, RationalSearch found the exact answer to both the formulae in under a minute.

We now describe our approach for exact model checking. The broad idea is to utilize approximate solutions generated by an iterative technique, and then successively refine these solutions to the exact solution. We begin by first describing the first ingredient of our solution—the Kwek Mehlhorn algorithm [40] in Section 5.1. We then describe the overall algorithm in Sect. 5.2.

5.1 The Kwek–Mehlhorn algorithm

Given an ordered set of integers of bounded size, the classical binary search algorithm can be used to find the smallest element in the set that is larger than a given value, in logarithmic time. Kwek and Mehlhorn [40] extend this methodology to efficiently locate the rational number with the smallest size in a given interval. In our paper, we present a novel application of this technique, where approximate answers to quantitative model checking problems can be used to generate exact solutions efficiently.

Let $I = [\frac{\alpha}{\beta}, \frac{\gamma}{\delta}]$ be an arbitrary interval with rational end-points. It was established [40] that for such an interval, there exists a unique rational $a_{\min}(I)/b_{\min}(I)$ such that for all rational numbers $\frac{a}{b} \in I$, $a_{\min}(I) \leq a$ and $b_{\min}(I) \leq b$. We will call $a_{\min}(I)/b_{\min}(I)$ the minimal fraction of I. Further, this minimal fraction $a_{\min}(I)/b_{\min}(I)$ can be found using Algorithm 1 from [40]. The input to the findFraction procedure are integers denoting the numerators and denominators of the endpoints of the interval I, and the output is a pair of integers, corresponding to the numerator and denominator of the unique minimal fraction of the input interval.

Algorithm 1 Compute the minimal rational in $[\frac{\alpha}{\beta}, \frac{\gamma}{\delta}]$

```
function findFraction($\alpha, \beta, \gamma, \delta$):
    if $\lfloor \frac{\alpha}{\beta} \rfloor = \lfloor \frac{\gamma}{\delta} \rfloor$ and $\frac{\alpha}{\beta} \notin \mathbb{N}$ then
        $b, a \leftarrow$ findFraction($\delta, \gamma \bmod \delta, \beta, \alpha \bmod \beta$)
        return $\lfloor \frac{\alpha}{\beta} \rfloor b + a, b$
    else
        return $\lceil \frac{\alpha}{\beta} \rceil, 1$
    end if
end function
```

Let $Q_M = \{p/q \mid p, q \in \{1, ..., M\}\} \cap [0, 1]$. For $\mu \in \mathbb{N}$, if $\frac{a}{b} \in Q_M$ is contained in the interval $[\frac{\mu}{2M^2}, \frac{\mu+1}{2M^2}]$ of length $\frac{1}{2M^2}$ then $\frac{a}{b}$ is the unique element of Q_M in $[\frac{\mu}{2M^2}, \frac{\mu+1}{2M^2}]$. It turns out that $\frac{a}{b}$ must also be the minimal element of $[\frac{\mu}{2M^2}, \frac{\mu+1}{2M^2}]$, meaning it can be found using Algorithm 1 in time $O(\log M)$.

5.2 Rational search

In this section, we explain our approach for the exact quantitative model checking of PCTL formulas. The critical insight we exploit is that iterative techniques for solving constrained reachability typically converge very fast and produce a precise enough answer. Using this precise approximation, we can then effectively construct a small interval so that the minimal fraction in the interval corresponds to an element of the exact solution vector, and thus the Kwek–Mehlhorn algorithm can be employed to find the exact solution.

Recall that each iterative technique for approximating a set of equations, like those given in Eqs. (1) and (3), yields a different guarantee on the precision of an approximate solution. The difference between the approximation generated by interval iteration and the actual solution is bounded by a given ϵ value, provided there are no errors generated by floating-point arithmetic. Value iteration, on the other hand, comes with no such guarantees. When an approximate solution vector contains values of known precision, like in the case of interval iteration, one can translate it into an exact solution vector as follows. For each value q in the vector, construct the interval $[q - \epsilon, q + \epsilon]$ and run Algorithm 1 to find the smallest rational in this interval. Then, check that the generated rational values $\mathsf{V}^\star$ are correct by verifying that they satisfy the fix point equations for constrained reachability and expected costs. In addition, if the algorithm also checks that condition on the graph $\mathcal{G}_{\mathsf{V}^\star}$ (or $\mathcal{H}_{\mathsf{V}^\star}$) also hold in accordance with Lemma 1 (Lemma 3 respectively) if we are computing max reachability probabilities (min reachability respectively) properties. Lemmas 1 and 3, along with the uniqueness of the fix points for the rest of the cases, imply that these checks are only satisfied by the desired solution vector. If these checks fail for the candidate solution vector, one obtains a more precise approximation and re-runs the procedure.

When a solution vector contains values of unknown quality, we can find exact solutions using a similar technique. Here the idea is to “guess” a sequence intervals, with decreasing sizes, that may contain the actual value. This process is formalized in Algorithm 2, which takes as input the model $\mathcal{M}$, a maximum precision P and a state-indexed vector $\mathsf{V}^\dagger$ that approximates the exact solution vector V.

Algorithm 2 Sharpen values of unknown precision

```
function SHARPEN($\mathcal{M}$, $P$, $\mathsf{V}^\dagger$, $\xi$, obj):
    for all $p \in \{1, ..., P\}$ do
        for all $z \in Z$ do
            $\alpha, \beta, \gamma, \delta \leftarrow$ BOUNDS($p$, $\mathsf{V}^\dagger(z)$)
            $\mathsf{V}^\star(z) \leftarrow \lfloor \mathsf{V}^\dagger(z) \rfloor +$ FINDFRACTION($\alpha, \beta, \gamma, \delta$)
        end for
        if FIXPOINT($\mathcal{M}$, $\mathsf{V}^\star$, $\xi$, obj ) then
            return $\mathsf{V}^\star$
        end if
    end for
    return null
end function
```

For a given precision p and state z, BOUNDS$(p, \mathsf{V}^{\dagger}(z))$ returns $\alpha, \beta, \gamma, \delta$ such that α is the first p decimal digits of the fractional part of $\mathsf{V}^{\dagger}(z)$, $\beta = 10^p$, $\gamma = \alpha + 1$ and $\delta = \beta$. Observe that α/β is the rational representation of the first p digits of the fractional part of $\mathsf{V}^{\dagger}(z)$. From this approximation, we identify a *sharpened* solution vector $\mathsf{V}^{\star}$ using the FINDFRACTION procedure from Algorithm 1. The procedure FIXPOINT then tests if $\mathsf{V}^{\star}$ is the correct solution by checking if the equation satisfies the appropriate fix point equation in addition to the check, if needed, required by Lemma 1 or 3. If the input vector $\mathsf{V}^{\dagger}$ is not precise enough, then SHAPREN returns "null", indicating that more precision is required to infer an exact solution. The guarantees of Algorithm 2 are formalized as follows. Let V^b satisfying $|\mathsf{V}(z) - \mathsf{V}^b(z)| \leq 10^{-b}$ for all $z \in Z$ be an approximate solution vector of precision b. Then, Lemma 4 establishes that starting from a close enough approximation, Algorithm 2 finds the actual solution vector.

Lemma 4 *Let $\mathcal{M}$ be an MDP with the set of states Z. Let ξ be a PCTL path formula or a PCTL state formula. Given an objective $\mathsf{obj} \in \{\mathsf{max}, \mathsf{min}\}$, let V be the vector $\lambda z \cdot p_z^{\mathsf{obj}}[\xi]$ if ξ is a path formula and the vector $\lambda z \cdot e_z^{\mathsf{obj}}[\xi]$ if ξ is a state formula. Let $b, P \in \mathbb{N}$ be such that $P \geq b$ and V^b is an approximate solution vector of precision b. If $V(z) \in Q_{\lfloor\sqrt{10^b/2}\rfloor}$ for every $z \in Z$, then* SHARPEN$(\mathcal{M}, P, V^b, \xi, \mathsf{obj}) = V$.

Proof Fix a state z and assume $\mathsf{V}(z) \in Q_M$ for $M{=}\lfloor\sqrt{10^b/2}\rfloor$. If $P{\geq}b$ then SHARPEN$(\mathcal{M}, P, \mathsf{V}^b, \xi, \mathsf{obj})$ searches for $\mathsf{V}(z)$ in $I = [\alpha/\beta, \gamma/\delta]$ for $\alpha, \beta, \gamma, \delta =$ BOUNDS$(b, \mathsf{V}^b(z))$. Now, $\mathsf{V}(z) \in I$ since $\mathsf{V}^b(z)$ satisfies $|\mathsf{V}(z) - \mathsf{V}^b(z)| \leq 10^{-b}$. Further, $|I| = 10^{-b} \leq \frac{1}{2M^2}$. Due to Kwek et. al. [40], we have that an interval of size $\frac{1}{2M^2}$ contains at most 1 element of Q_M. Clearly, FINDFRACTION$(\alpha, \beta, \gamma, \delta)$ returns $\mathsf{V}(z)$ which is the unique "minimal" element in $I \cap Q_M$. □

Using the techniques for sharpening an approximate solution into an exact value from Algorithm 2, we can now derive a procedure for solving constrained reachability (and hence PCTL) formulas exactly. The procedure is given in Algorithm 3, which takes as arguments an MDP or DTMC $\mathcal{M}$, a constrained reachability formula ϕ and a precision ϵ. The ITERATION procedure can be either of value iteration or interval iteration. Algorithm 3 begins by running the iteration procedure up to a given precision ϵ. If the procedure is value iteration, ϵ is used in the convergence criterion—absolute or relative—described in Section 3. In the case of interval iteration, ϵ defines the bound on the maximum error in the approximate solution vector. The approximate solution vector $\mathsf{V}^{\dagger}$ generated by the iteration procedure is then used by the SHAPREN procedure, which attempts to strengthen the approximate answer to an exact one. Note the version of the SHAPREN varies according to the iterative method being utilized. If it succeeds, the whole process terminates. Otherwise, $\mathsf{V}^{\dagger}$ is further refined by re-invoking ITERATION with an increased ϵ precision, and the sharpening process is repeated.

When successive approximations in value iteration or interval iteration are computed using arbitrary precision arithmetic, the correctness guarantees of Algorithm 3 can be stated as follows.

Theorem 1 *Let $\mathcal{M}$ be an MDP with the set of states Z. Let ξ be a PCTL path formula or a PCTL state formula. Given an objective $\mathsf{obj} \in \{\mathsf{max}, \mathsf{min}\}$, let V be the state-indexed vector $\lambda z \cdot p_z^{\mathsf{obj}}[\xi]$ if ξ is a path formula and the vector $\lambda z \cdot e_z^{\mathsf{obj}}[\xi]$ if ξ is a state formula. Then,* RATIONALSEARCH$(\mathcal{M}, \xi, \mathsf{obj}, \epsilon_0)$ *(with $\epsilon_0 > 0$) terminates and returns the exact solution vector V.*

Algorithm 3 Rational Search

```
function RATIONALSEARCH($\mathcal{M}$, $\xi$, obj, $\epsilon_0$):
    $\mathsf{V}^{\text{init}} \leftarrow$ INIT($\mathcal{M}$, $\phi$)
    $\epsilon \leftarrow \epsilon_0$
    while true do
        $\mathsf{V}^{\dagger} \leftarrow$ ITERATION($\mathcal{M}$, $\xi$, obj, $V^{\text{init}}$, $\epsilon$)
        $\mathsf{V}^{\star} \leftarrow$ SHARPEN($\mathcal{M}$, $\lceil \log_{10}(\frac{1}{\epsilon}) \rceil$, $\mathsf{V}^{\dagger}$, $\xi$, obj)
        if $\mathsf{V}^{\star} \neq$ null then
            return $\mathsf{V}^{\star}$
        end if
        $\mathsf{V}^{\text{init}} \leftarrow \mathsf{V}^{\dagger}$
        $\epsilon \leftarrow \epsilon/10$
    end while
end function
```

Proof It is easy to see that there is a $b > 0$ such that, for every state z, $\mathsf{V}(z) \in Q_N$ for $N = \lfloor \sqrt{10^b/2} \rfloor$. Now, since value iteration converges in the limit, we have that the first b digits of $\mathsf{V}^{\dagger}(z)$ match that of $\mathsf{V}(z)$ for each state $z \in Z$, eventually. Also, in every iteration of the loop in Algorithm 3, SHARPEN is invoked with an incremented value of P and eventually $P \geq b$. □

We now state the complexity of computing the exact solutions using RATIONALSEARCH. As before, we assume that the transition probabilities are given as rational numbers.

Theorem 2 *Let $\mathcal{M}$ be an MDP with the set of states Z. Let $n = |Z|$, $m = |\{(z, \alpha, z') | \alpha \in \mathit{enabled}(z), \Delta(z, \alpha)(z') > 0\}|$ and let δ be the largest denominator in any probability value in the transition function of $\mathcal{M}$. Let ξ be a PCTL path formula. Let p_{min} be the* $\min\{\delta(z, \alpha, z') \mid \Delta(z, \alpha)(z') > 0\}$*. Given an objective $obj \in \{max, min\}$ and let V be the state-indexed vector $\lambda z \cdot p_z^{obj}[\xi]$. Let $\ell = \frac{n(m+n)\log\delta}{p_{min}^n}$. Then,* RATIONALSEARCH($\mathcal{M}$, ξ, *obj*, 1) *makes at most $O(\ell)$ value iteration steps, $O(n\ell^2)$ calls to* FINDFRACTION*, and $O(\ell^2)$ calls to* FIXPOINT*, assuming arbitrary precision arithmetic.*

Proof Observe that we can assume without loss of generality that there is at least one transition probability that is contained in the open interval (0, 1) (Otherwise, the value iteration finishes in zero steps as all probabilities are 0 or 1).

We will proceed as follows. We assume that the objective obj is min. We first estimate the number of iterations k of value iteration that are required to reach an approximate solution state-indexed vector $\mathsf{V}^{\dagger}$ of precision b such that $\mathsf{V}^{\dagger}$ can be used to obtain the exact solution V using one call to SHARPEN based on Lemma 4..

From [19], we know that the maximum denominator (and thus maximum numerator) of any value in $\{\mathsf{V}(z) | z \in Z\}$ is less than δ^{4m}. Now, the required precision b satisfies $\lfloor \sqrt{\frac{10^b}{2}} \rfloor \geq \delta^{4m}$, giving us

$$10^{-b} \leq \frac{\delta^{-8m}}{2}.$$

Let us now estimate an upper bound on the number of steps of value iteration that are required to guarantee that the resulting approximate solution vector $||\mathsf{V} - \mathsf{V}^{\dagger}|| < 10^{-b}$. Here, the norm $|| \cdot ||$ is defined to be the pointwise maximum.

Let $\mathcal{U} = [0,1]^Z$ be the set of all state-indexed vectors. For a vector $\bar{x}$ we denote its z^{th} component by $\bar{x}(z)$. Consider the function $f : \mathcal{U} \to \mathcal{U}$ be the function such that

$$f(\bar{x})(z) = \begin{cases} 0 & \text{if } z \in \mathsf{Prob}_0^{\mathsf{min}}[\xi] \\ 1 & \text{if } z \in \mathsf{Prob}_1^{\mathsf{min}}[\xi] \\ \min\limits_{\alpha \in \mathsf{enabled}(z)} \sum\limits_{z' \in Z} \Delta(z, \alpha, z') \cdot \bar{x}(z') & \text{otherwise} \end{cases}$$

Observe that value-iteration described in Section 3 is such that $\bar{x}_0$ is the vector all of whose components is 0, and $\bar{x}_{i+1} = f(\bar{x}_i)$. The n-th iterate of f, namely f^n, is a contracting mapping (Please see Appendix A for the proof):

Claim For all vectors $\bar{x}, \bar{y} \in \mathcal{U}$,

$$||f^n(\bar{x}) - f^n(\bar{y})|| \leq q||\bar{x} - \bar{y}||$$

where $q = (1 - p_{\mathsf{min}}^n)$ and p_{min} is the smallest non-zero probability in the description of $\mathcal{M}$.

Let $\mathsf{V}^{\dagger} = \bar{x}_{i \cdot n}$ be the required approximation, obtained after $i \cdot n$ value iteration steps. Using, Banach's fixpoint theorem [12], we have

$$||\mathsf{V} - \bar{x}_{i \cdot n}|| \leq \frac{q^i}{1-q}||\bar{x}_n - \bar{x}_0|| = \frac{q^i}{1-q}||\bar{x}_n|| < \frac{q^i}{1-q}.$$

Based on our requirement for $k = i \cdot n$, we will need only one function call to SHARPEN if

$$\frac{q^i}{1-q} < 10^{-b} \leq \frac{\delta^{-8m}}{2}.$$

Let i_0 be an integer such that

$$q^{i_0} < \frac{\delta^{-8m}(1-q)}{2}.$$

We have that i_0 is an upper bound on i.

Now $q^{i_0} < \frac{\delta^{-8m}(1-q)}{2}$ iff

$$i_0 \log(1 - p_{\mathsf{min}}^n) < -1 - 8m \log \delta + n \log p_{\mathsf{min}}.$$

Since $\log(1 - p_{\mathsf{min}}^n)$ is negative, we get that $q^{i_0} < \frac{\delta^{-8m}(1-q)}{2}$ iff

$$i_0 > \frac{-1 - 8m \log \delta + n \log p_{\mathsf{min}}}{\log(1 - p_{\mathsf{min}}^n)}.$$

Observe that $p_{\mathsf{min}} \geq \frac{1}{\delta}$. Thus, $\log p_{\mathsf{min}} \geq -\log \delta$ and hence

$$\frac{\log p_{\mathsf{min}}}{\log(1 - p_{\mathsf{min}}^n)} \leq \frac{-\log \delta}{\log(1 - p_{\mathsf{min}}^n)}.$$

Thus, $q^{i_0} < \frac{\delta^{-8m}(1-q)}{2}$ if

$$i_0 > \frac{-1 - 8m \log \delta - n \log \delta}{\log(1 - p_{\mathsf{min}}^n)}.$$

Using the inequality $\ln(1+x) \leq x$ for $x > -1$, we have that $\ln(1-p_{\mathsf{min}}^n) \leq -p_{\mathsf{min}}^n$ and hence $\frac{-\ln 2}{p_{\mathsf{min}}^n} \leq \frac{1}{\log(1-p_{\mathsf{min}}^n)}$ and hence $\frac{-1}{p_{\mathsf{min}}^n} \leq \frac{1}{\log(1-p_{\mathsf{min}}^n)}$. Since multiplying an inequality by a negative number changes signs, we get that

$$\frac{1+8m\log\delta+n\log\delta}{p_{\mathsf{min}}^n} \geq \frac{-1-8m\log\delta-n\log\delta}{\log(1-p_{\mathsf{min}}^n)}.$$

Thus, $q^{i_0} < \frac{\delta^{-8m}(1-q)}{2}$ if

$$i_0 > \frac{1+8m\log\delta+n\log\delta}{p_{\mathsf{min}}^n}.$$

Thus, we are guaranteed to terminate the algorithm using one call to SHARPEN after k steps, where

$$k = i_0 \cdot n = \frac{1+8m\log\delta+n\log\delta}{p_{\mathsf{min}}^n} \cdot n = O(\frac{n(m+n)\log\delta}{p_{\mathsf{min}}^n}) = O(\ell).$$

Now, let us analyze the calls to SHARPEN. Note that the jth call to SHARPEN has precision $P_j = j$. The maximum value of P_j is k. Every call to SHARPEN gives rise to nP_j calls to FINDFRACTION and P_j calls to FIXPOINT, giving us $O(n\ell^2)$ calls to FINDFRACTION and $O(\ell^2)$ calls to FIXPOINT.

When obj is max, then please note that there is a memoryless scheduler $\mathfrak{S} : Z \to \mathsf{Act}$ such that for each state $z \in Z$, $p_z^{\mathsf{max}}(\xi) = p_z^{\mathfrak{S}}(\xi)$. Consider the functions $f_1, f_2 : \mathcal{U} \to \mathcal{U}$ defined as follows:

$$f_1(\bar{x})(z) = \begin{cases} 0 & \text{if } z \in \mathsf{Prob}_0^{\mathsf{max}}[\xi] \\ 1 & \text{if } z \in \mathsf{Prob}_1^{\mathsf{max}}[\xi] \\ \sum\limits_{z' \in Z} \Delta(z, \mathfrak{S}(z), z') \cdot \bar{x}(z') & \text{otherwise} \end{cases}$$

and

$$f_2(\bar{x})(z) = \begin{cases} 0 & \text{if } z \in \mathsf{Prob}_0^{\mathsf{max}}[\xi] \\ 1 & \text{if } z \in \mathsf{Prob}_1^{\mathsf{max}}[\xi] \\ \mathsf{max}_{\alpha \in \mathsf{enabled}(z)} \sum\limits_{z' \in Z} \Delta(z, \alpha, z') \cdot \bar{x}(z') & \text{otherwise} \end{cases}$$

Observe that value-iteration described in Sect. 3 is such that $\bar{x}_0$ is the vector all of whose components is 0, and $\bar{x}_{i+1} = f_2(\bar{x}_i) = f_2^i(\bar{x}_0)$.

Now, it is easy to see that the required solution vector V is the pointwise limit $\lim_{i\to\infty} f_1^i(\bar{x}_0) = \lim_{i\to\infty} f_2^i(\bar{x}_0)$. Further, we also have that for each i, $f_1^i(\bar{x}_0) \leq f_2^i(\bar{x}_0) \leq \mathsf{V}$ and hence $||\mathsf{V} - \bar{x}_i|| \leq ||\mathsf{V} - f_1^i(\bar{x}_0)||$. Observe that we can show f_1^i is contracting with factor $1 - p_{\mathsf{min}}^n$ exactly like the claim above. The theorem now follows similar to the case when obj is min. □

Example 5 Our experiments show that Algorithm 3 can make non-trivial improvements to solution quality. Consider the standard example of tossing N biased coins independently, where each coin yields heads with probability 1/3 and tails with probability 2/3. Analyzing the DTMC model to compute the probability of the event that 11 coins land heads, PRISM's floating-point model checker returned the decimal "0.000005645029269476758". Our tool could correctly determine the exact probability to be 1/177,147 by examining with the first 12 digits of this approximate answer. This is remarkable given that the period of this fraction (and hence its most succinct decimal representation) is almost 20,000 digits long. Moreover, the algorithm is able to simultaneously infer the reachability probabilities for *all* of the roughly 200,000 states of the model with a single fixpoint check. This illustrates another advantage of our technique; the algorithm is agnostic of the number of initial states in the system. The exact model checking engine of PRISM, on the other hand, currently only supports systems with a single initial state.

6 Implementation

We have implemented Algorithm 3 in our tool RationalSearch, which is an extension of the PRISM model checker (version 4.3.1). RationalSearch is available for download at [8]. Before describing our integration with PRISM, we briefly describe the relevant portions of its architecture. PRISM is a Java-based tool comprised of four solution engines, three of which (Mtbdd, Hybrid, Sparse) are based (entirely or partially) on symbolic methods using compact data structures like MTBDDs. The fourth engine (Explicit) manipulates sparse matrices, vectors and bit-sets directly (without any symbolic data structures).

The Sparse engine is similar to the Explicit engine in that it uses explicit data structures for storing vectors and matrices. However, it makes use of symbolic data structures during model construction, allowing it to efficiently remove portions of the state space that are not reachable. This is achieved through a conjunction of the MTBDD representing the model's state space with a BDD representing the characteristic function for the reachable states of the model. The Mtbdd engine is based entirely on symbolic data structures. During value iteration, the transition matrix and solution vector are both given as MTBDDs. The matrix-vector multiplications used to update the solution vector are carried out over these data structures. As described in [48], one drawback of this approach is that the size of the MTBDD storing the solution vector can grow substantially as more computations are performed. To tackle the MTBDD size explosion, the Hybrid engine combines the advantages present in both the symbolic and explicit engines. In particular, it stores the solution vector as a fixed size array and the transition matrix as an MTBDD (which can usually be done succinctly due to symmetry in the model). Updates to the solution vector are carried out by operations over these mixed-type data structures.

RationalSearch implements Algorithm 3 on top of all four engines for model checking DTMCs against PCTL specifications. For exact model checking of MDPs, our tool RationalSearch implements Algorithm 3 for all four engines when the PCTL specification does not involve computing any max probabilities and minimum expected costs. RationalSearch only supports the Explicit engine for the case of max probabilities and min rewards in MDPs, for which the fixpoint check involves additional graph-theoretic analyses (see Section 4). The architecture of our extension is outlined in Fig. 2. It intercepts PRISM's routine for solving constrained reachability probabilities and expected costs, sharpening the probabilities every time it is invoked. These engines are built using floating-point numbers, which can store at most 16 digits in the fractional part of the decimal expansion of any floating-point number. Hence, the convergence criteria support a minimum ϵ of 10^{-16}. Our implementation, thus, bypasses the ϵ refinement loop from Algorithm 3 and directly invokes the procedure iteration for the maximum precision supported by doubles. Further, for computing max reachability probabilities, checking whether the candidate solution vector returned by the Explicit engine is a fixpoint, we do not take recourse to Lemma 1. Instead, we take advantage of PRISM's ability to return a candidate memoryless scheduler that achieves the maximum reachability property. The candidate scheduler $\mathfrak{S}$ returned by PRISM is a *proper* scheduler, whose definition we articulate below.

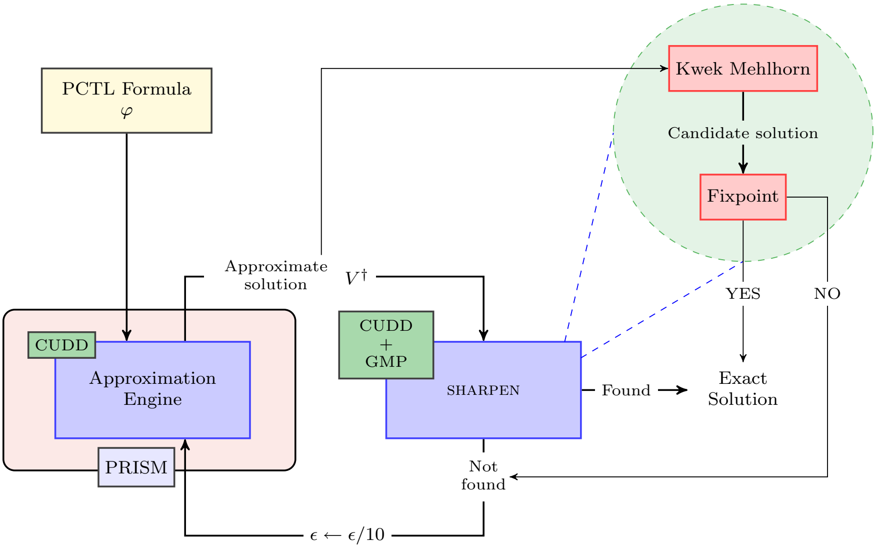

Fig. 2 RationalSearch Architecture: Given a PCTL formula φ, PRISM (equipped with CUDD) approximates the solution using value/interval iteration. The SHARPEN procedure uses this approximation $V^\dagger$ and employs FINDFRACTION, in conjunction with the rational extension to CUDD (CUDD + GMP), to generate a candidate rational vector. If this candidate rational vector satisfies an appropriate fixpoint check, it is guaranteed to be correct. Otherwise, the process is repeated with a better approximation

Definition 6 Let $\mathcal{M} = (Z, \mathsf{Act}, \Delta, \mathbf{C}, L)$ be a MDP and ϕ, ϕ' be PCTL state formulas. A memoryless scheduler $\mathfrak{S}$ for $\mathcal{M}$ is said to be proper for $\mathcal{M}, \phi, \phi'$ if for each $z \in Z \backslash \mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \cup \mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']$, there is a sequence of states $z_1, \ldots, z_\ell$ such that

- $z_1 = z$,
- $z_\ell \in \mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']$, and
- $\Delta(z_i, \mathfrak{S}(z), z_{i+1}) > 0$ for each $1 \leq i < \ell$.

In order to check whether a given candidate solution, $\hat{\mathsf{V}}$, to the set of Eqs. (7) is indeed the actual exact solution $\mathsf{V} = \lambda z \cdot p_z^{\mathsf{min}}(\phi\ \mathcal{U}\ \phi')$, it suffices to check that the proper scheduler, $\mathfrak{S}$, returned by PRISM is such that $\mathfrak{S}(z) \in \mathsf{argmax}_z^{\hat{\mathsf{V}}}$ for every $z \in Z \backslash \mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \cup \mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']$:

Proposition 1 *Let $\mathcal{M} = (Z, \mathsf{Act}, \Delta, \mathbf{C}, L)$ be a MDP, ϕ, ϕ' be PCTL state formulas and $\mathfrak{S}$ a proper memoryless scheduler for $\mathcal{M}, \phi, \phi'$. For each state z of $\mathcal{M}$, let $p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$ be the maximum probability of satisfying $\phi\ \mathcal{U}\ \phi'$ in $\mathcal{M}$. Let $\hat{\mathsf{V}} : Z \to [0, 1]$ be a solution of the set of equations given by Eq. (7). Suppose further that*

$$\hat{\mathsf{V}}(z) = \sum_{z' \in Z} \Delta(z, \mathfrak{S}(z), z') \cdot \hat{\mathsf{V}}(z') \textit{ for each } z \in Z \backslash \mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \cup \mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'].$$

Then $\forall z \in Z,\ \hat{\mathsf{V}}(z) = p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$.

Proof As $\mathsf{V} = \lambda z \cdot p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi')$ is the least fix point of Eq. (7), we have that for each state $z \in Z$,

$$p_z^{\mathfrak{S}}(\phi\ \mathcal{U}\ \phi') \leq p_z^{\mathsf{max}}(\phi\ \mathcal{U}\ \phi') \leq v_z.$$

Thus, it suffices to show that $p_z^{\mathfrak{S}}(\phi\ \mathcal{U}\ \phi') = \hat{\mathsf{V}}(z)$ for each $z \in Z$.

Let $\mathsf{prop}_\phi, \mathsf{prop}_{\phi'}$ be distinct propositions. Given a proper memoryless scheduler $\mathfrak{S}$ for $\mathcal{M}, \phi, \phi'$, let $\mathcal{M}^{\mathfrak{S}}_{\phi,\phi'} = (Z, \mathsf{Act}, \Delta^{\mathfrak{S}}, \mathbf{C}, L^{\mathfrak{S}})$ be the DTMC such that $\Delta^{\mathfrak{S}}(z, z') = \Delta(z, \mathfrak{S}(z), z')$, $L^{\mathfrak{S}}(z) = \{\mathsf{prop}_\phi\}$ if $\mathcal{M}, z \models \phi$ and $L^{\mathfrak{S}}(z) = \{\mathsf{prop}_{\phi'}\}$ if $\mathcal{M}, z \models \phi'$. It is easy to see that $p_z^{\mathfrak{S}}(\phi\ \mathcal{U}\ \phi')$ is exactly the probability of z satisfying the formula $\mathsf{prop}_{\phi'}\ \mathcal{U}\ \mathsf{prop}_{\phi'}$ in $\mathcal{M}^{\mathfrak{S}}_{\phi,\phi'}$. Observe further that from the fact that $\mathfrak{S}$ is proper, the set of states of $\mathcal{M}^{\mathfrak{S}}_{\phi,\phi'}$ that satisfy $\mathsf{prop}_{\phi'}\ \mathcal{U}\ \mathsf{prop}_{\phi'}$ with probability 0 (1 respectively) is exactly the set $\mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']$ ($\mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi']$ respectively). Since $\mathcal{M}^{\mathfrak{S}}$ is a DTMC, $\mathsf{V}^{\mathfrak{S}} = \lambda z \cdot p_z^{\mathfrak{S}}(\phi\ \mathcal{U}\ \phi')$ is the *unique* solution to the set of equations:

$$\begin{array}{ll} y_z = 0 & \text{if } z \in \mathsf{Prob}_0^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \\ y_z = 1 & \text{if } z \in \mathsf{Prob}_1^{\mathsf{max}}[\phi\ \mathcal{U}\ \phi'] \\ y_z = \sum\limits_{z' \in Z} \Delta^{\mathfrak{S}}(z, z') \cdot y_{z'} = \sum\limits_{z' \in Z} \Delta(z, \alpha, z') \cdot y_{z'} & \text{otherwise} \end{array} \tag{11}$$

Finally, observe that $\hat{\mathsf{V}}$ is a solution to the above Eq. (11). Hence, we must have $p_z^{\mathfrak{S}}(\phi\ \mathcal{U}\ \phi') = \hat{\mathsf{V}}(z)$ for each $z \in Z$. □

Among the four engines, Explicit is the only one implemented entirely in Java. To support the Explicit engine, our tool uses the libraries JScience [7] and Apfloat [4] to construct the transition matrix using rational entries, perform matrix-vector multiplications for the fixpoint check in Algorithm 3, and implement the Kwek–Mehlhorn algorithm (Algorithm 1).

PRISM implements the remaining three engines using an extension of the CUDD library [5]. The off-the-shelf version of CUDD only supports floating-point numbers at the terminals. RationalSearch enhances CUDD by allowing terminals to hold either floating-points or arbitrary-precision rational numbers provided by the GNU MP library [6]. Our extension allows the data type at a terminal node to be easily interchanged, and the full suite of MTBDD operations can be performed regardless of the data type.

RationalSearch makes use of this extended CUDD functionality in the following manner. When the model is parsed, it constructs two transition matrices, one with doubles at the terminal nodes and one with rationals. The procedure iteration uses double-precision transition matrix to generate a double-precision solution vector. RationalSearch translates this solution vector into a candidate solution vector stored as a rational MTBDDs using sharpen. The fixpoint check from sharpen can then be performed by an MTBDD matrix-vector multiplication between rational MTBDDs.

Algorithm 3 has also been integrated into the STORM model checker. Their implementation[3] differs from ours in that it supports running iteration with both floating-point and arbitrary-precision numbers. It begins by running value iteration using floating-point numbers and attempts to infer and exact solution from the approximation. If double-precision is determined to be insufficient for extracting the precise solution, the approximation engine is re-invoked using arbitrary-precision numbers. Another significant difference in the STORM implementation is that STROM uses the Sylvan [24] MTBDD library instead of CUDD. Sylvan provides built-in support for arbitrary precision arithmetic.

[3] Information about the implementation of Algorithm 3 in STORM was obtained through private email conversations with the developers.

7 Evaluation

7.1 Setup

We evaluated our tool against examples involving quantitative reachability and costs from the PRISM benchmark suite and case studies [2,3] and compared the results with the exact parametric engines implemented in PRISM and STORM. In particular, we used version 4.3.1 of PRISM and version 1.0.0 of STORM. Our tests were carried out on an Intel core i7 dual-core processor @2.2 GHz with 8 Gb RAM running macOS 10.12.4.

7.2 Benchmarks

Our focus has been to evaluate the performance of our technique on different probabilistic models (MDPs and DTMCs) against different objectives (Reachability, Cost, full-fledged PCTL). Our PCTL examples, in particular, have been crafted from scratch. Our benchmarks have been selected from the PRISM benchmark suite and case studies [2,3] by keeping some key objectives in mind. First, in order to stress-test our technique, we tried to choose benchmarks with large state spaces. In fact, most of our benchmarks have state spaces of the order of 10^5–10^6. Second, we also selected some benchmarks (for example, biased coins, ECS, leader election) for which the probability values corresponding to the properties have high precision, that is, their decimal representation requires many digits. We believe that such benchmarks demonstrate the need for exact model checking, as well as, the effectiveness of our technique in determining the correct rational representations of the probabilities. In this process of benchmark selection, we omitted benchmarks for which the resulting answers are trivial (either 0 or 1 probability) or those for which our technique could not result in a fix point. We recall that due to floating-point errors, PRISM's approximate answer may never get close enough (in a precise sense stated in Theorem 1, Sect. 5) to the actual exact answer (despite an arbitrary number of iterations) and as a consequence, RationalSearch may declare that it did not find an exact answer. We note that our tool never reports an incorrect answer.

7.3 Performance overhead

We examined the overhead incurred by RationalSearch's extension of PRISM. The results are given in Table 1 for the approximation engines Explicit, MTBDD and Hybrid of PRISM. Due to the similarity between the Explicit and Sparse engines, we chose to only report metrics for the former. In Table 1, all of the tests were conducted using value iteration as the approximation scheme. The overhead incurred for interval iteration is similar and thus not reported. The quantitative properties tested against in two of the MDP benchmarks ('Fair Exch.' and 'Dice Coin') involve computation of max probabilities. Recall that RationalSearch supports this combination only for the Explicit engine, and as a result, the corresponding entries in columns 8–11 (MTBDD and Hybrid engines) are marked '-' for these benchmarks.

On several examples with large state spaces, the Explicit engine fails with an out-of-memory exception. This can be attributed to the fact that the implementation stores two copies of the transition matrix in memory. On all the examples where Explicit fails, the symbolic engines (MTBDD and Hybrid) find the solution quickly, typically with an overhead of less than 50%. For the examples on which the Explicit engine did not encounter an

Table 1 Experimental evaluation of RationalSearch *Overhead*

1 Model	2	3	4	5	6 Explicit	7	8 MTBDD	9	10 Hybrid	11
Name	Type	Prop	Param	States	Time	Overhead	Time	Overhead	Time	Overhead
Biased coins	DTMC	Reach	15	14,348,907	OOM	n/a	.18	62%	2.23	3%
IPv4	DTMC	Reach	100,000	100,003	4.1	254%	1708	1%	1702	1%
Crowds	DTMC	Reach	15	119,800	MP	n/a	MP	n/a	MP	n/a
Lead. Elec.	DTMC	Cost	4	12,302	1.5	117%	6.3	27%	19.6	7%
ECS	DTMC	PCTL	14	4,815,782	OOM	n/a	.4	70%	11.1	1%
Dice	MDP	Reach	6	4,826,809	OOM	n/a	.57	48%	2.4	6%
Din. Crypt.	MDP	Reach	9	855,095	OOM	n/a	.381	41%	.84	13%
Fair Exch.	MDP	Reach	400	321,600	11.4	490%	–	–	–	–
Firewire	MDP	Reach	11,000	428,364	87.7	640%	15.1	7%	16.7	7%
Din. Phil.	MDP	Cost	3	956	.54	55%	2.86	1%	.22	10%
Virus	MDP	Cost	3	809	.47	70%	2.3	1%	.2	19%
Dice Coin	MDP	PCTL	1	728	.59	114%	–	–	–	–

Columns 1–5 describe the benchmark examples. Columns 6–10 report the performance and overhead metrics for RationalSearch's extension of the various PRISM engines. Running times are reported in seconds. Overhead percentages were calculated by examining the time the routines added by RationalSearch contributed to the overall running time. All tests were conduced with the absolute convergence criterion ($\epsilon = 10^{-16}$), `javamaxmem=4g` and `cuddmaxmem=4g`. TO represents a timeout (set to 30 min), OOM indicates an out of memory exception and MP indicates that more than double precision is required to produce an exact answer. We write n/a if information could not be determined due to a timeout or an out of memory exception

out-of-memory exception, overhead times where much higher. One major reason for this difference is that the EXPLICIT engine stores the solution vector as an array. Further, in this case, RATIONALSEARCH runs the SHARPEN procedure for each element of this array, thus resulting in redundant computation when a number appears multiple times. By contrast, the symbolic engines perform symmetry reductions on the data structures and store only distinct values at the terminal nodes of the solution vector. As a result, SHARPEN needs only be run once for each terminal node.

An encouraging observation from our results was that the overhead times did not vary drastically with the size of the model or the type of property being checked. In particular, both PCTL properties that we examined required solving three instances of constrained reachability properties. In spite of this, the overhead induced by RATIONALSEARCH on these examples remained consistent with the other examples.

Comparison with exact engines We also compared RATIONALSEARCH with the exact engines implemented in PRISM and STORM. The results are reported in Table 2. The existing exact engines of both PRISM and STORM were invoked with the `-exact` flag. In addition, STORM also uses the flag `-minmax:method pi`. RATIONALSEARCH was run with the underlying HYBRID engine and value iteration with absolute convergence criterion (with $\epsilon = 10^{-16}$) as the underlying approximation scheme. We set `javamaxmem=4g` and `cuddmaxmem=4g` wherever applicable. As before, Table 2 does not include benchmarks 'Fair Exch.' and 'Dice Coin' from Table 1. This is because these benchmarks are MDP models, and the specifications to be tested involve computation of max probabilities, which the HYBRID engine of RATIONALSEARCH does not currently support.

RATIONALSEARCH drastically outperformed PRISM's exact engine; in many cases, by several orders of magnitude. For about half of the examples, PRISM's exact engine reached the 30-min timeout. In every case, RATIONALSEARCH was able to find the exact solution in a matter of seconds. The comparison with the STORM tool is more competitive. For the majority of the small and medium-size examples (IPv4, Fair Exchange, Firewire, Dining Philosophers), the running times for both engines were within the same order of magnitude. However, the performance benefit of RATIONALSEARCH became apparent with large models (Biased Coins, Dice, ECS). RATIONALSEARCH achieved a 200x speed-up on the example of the biased coins and 45x speed-up on the dice example. For the embedded control system example, RATIONALSEARCH returned a solution in a matter of seconds while both PRISM and STORM hit the 30 min timeout.

In order to check the scalability of each of the exact engines, we also compared the running times on specific models (Biased Coins and Dice) where the number of states is governed by parameters that can be tuned to change the size of the underlying models. The results are depicted in Fig. 3, where we use an approximate engine of PRISM as a baseline for our comparative analysis. Several interesting observations can be made here. As expected, the approximate engine of PRISM is the fastest. Since, RATIONALSEARCH is crucially tied to the approximate engine(s) in PRISM, it is not surprising again, that (RATIONALSEARCH) scales very well on large models, with comparable performance to the underlying approximate engine because of the low overhead our technique imposes. While the existing exact model checking engines in PRISM and STORM do perform well when the models are small, the performance quickly degrades when the models become reasonably large (the scale is a logarithmic scale). This clearly demonstrates the power of the insight that the approximate answers from fast iterative model checking techniques can be utilized to obtain exact rational solutions with only a little overhead.

Comparison of iterative techniques. The final goal of our evaluation was to determine which approximation technique, amongst value iteration and interval iteration, could be more effec-

Table 2 Experimental comparison of exact engines

1 Model	2	3	4	5	6 PRISM Exact	7	8 STORM Exact	9	10 RationalSearch	11
Name	Type	Prop	Param	States	Time	Model	Time	Model	Time	Model
Biased Coins	DTMC	Reach	15	14,348,907	TO	n/a	458	375	2.23	.02
IPv4	DTMC	Reach	100,000	100,003	1141	6	342	.6	1702	1701
Lead. Elec.	DTMC	Cost	4	12,302	70	1.7	1.37	0.2	19.6	1.2
ECS	DTMC	PCTL	14	4,815,782	TO	1435	TO	104	11.1	.04
Dice	MDP	Reach	6	4,826,809	TO	1016	109	76	2.4	.05
Din. Crypt.	MDP	Reach	9	855,095	TO	39	12	11.5	.84	.06
Firewire	MDP	Reach	11,000	428,364	244	6.8	27	2.4	16.7	6.6
Din. Phil.	MDP	Cost	3	956	2.1	.2	.13	.125	.22	.03
Virus	MDP	Cost	3	809	1.3	.5	PE	PE	.2	.05

Columns 1–5 describe the benchmark examples. Columns 6, 8, 10 report the running times (in s) for each of the tools. Columns 7, 9, 11 report the portion of the model checking times (Columns 6, 8, 10) used for model construction. The configuration options for each of the tools is described in the main text. TO represents a timeout (set to 30 min) and OOM indicates an out of memory exception. We write n/a if information could not be determined due to a timeout or an out of memory exception. The PE in Columns 8 and 9 represent a parsing error in STORM

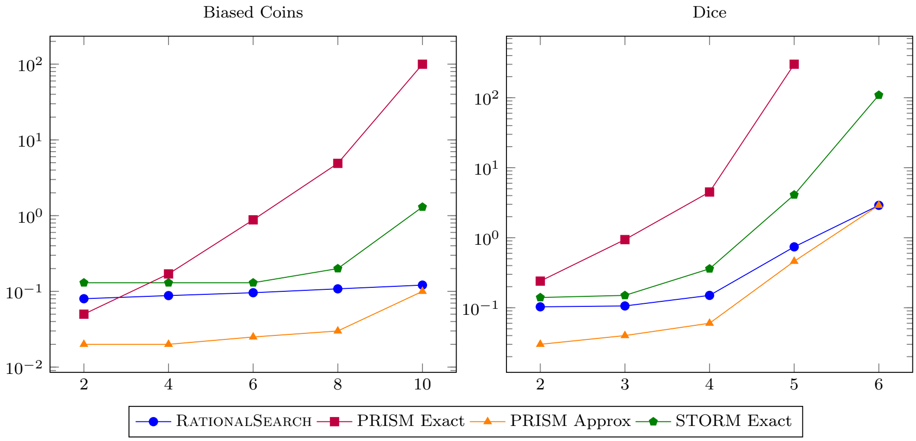

Fig. 3 **Scaling Comparison**. Running times for various model checking engines on the biased coins (left) and dice (right) examples. In both graphs, the values on the x-axis represent the parameters of the given model, and the values on the y-axis represent the running times (in $\log_{10}$ scale). The configuration options for RationalSearch, PRISM Exact and STORM exact identical to those in Table 2. PRISM approx was invoked using the same base options as RationalSearch. No data point is given for PRISM Exact with parameter six on the dice example as a 30-min timeout was reached

tively integrated with Algorithm 3. In particular, we compared the two approaches for speed and the quality of their approximations. The results are given in Table 3. We integrated RationalSearch with the implementation of interval iteration in PRISM from prior work [11], available at [1].

To our surprise, we found that the interval iteration implementation from [1] did not always produce an approximate solution within the specified ϵ threshold. In particular, for the dice example under parameter six, the approximations for both $\epsilon = 10^{-6}$ and $\epsilon = 10^{-12}$ were not within the given threshold. This resulted in RationalSearch not being able to infer an exact solution. Several other examples also suffered from this symptom. Although the approximate probabilities for the initial states were precise enough, poor approximations for the other states in the solution vector prevented RationalSearch from finding an exact solution.

The accuracy and precision of solution produced by approximation techniques varied according to the ϵ threshold and the iterative technique used. Although we have not reported the numbers in Table 3, there are also examples for which the approximations for value (interval) iteration differ across the solution engines (for the same value of ϵ). In spite of the difference in the approximations, RationalSearch is able to infer an exact solution for all of these different approximations.

In terms of speed, we observed only a small variance in the performance of the two techniques on the benchmarks we used. In most cases, value iteration slightly outperformed interval iteration. The difference is primarily a result of the extra cost incurred by interval iteration to perform the additional pre-processing steps it requires. This cost outweighs the savings afforded by the version of SHARPEN used with interval iteration that requires only a single fixpoint. In addition, our benchmarks did not identify any examples for which the improved precision of interval iteration allowed RationalSearch to infer an exact solution where value iteration could not. The preceding observations, in conjunction, lead us to conclude value iteration is the more effective partner for Algorithm 3.

Table 3 Experimental comparison of iterative techniques

1	2	3	4	5	6	7	8	9	10
Model					Value Iteration			Interval Iteration	
Name	Param	States	Epsilon	Solution	Approx	FP	Time	Approx	Time
Firewire	11, 000	428, 364	10^{-6}	2,087,481/2,097,152	0.9953885078430176	n/a	n/a	0.9953885078430176	n/a
Firewire	11, 000	428, 364	10^{-12}	2,087,481/2,097,152	0.9953885078430176	11	16.2	0.9953885078430176	27.7
Dice	3	2197	10^{-6}	1/216	0.004629455506801605	4	.1	0.004629705101251602	n/a
Dice	3	2197	10^{-12}	1/216	0.00462962962906488	4	.1	0.004629629629 7008155	n/a
Dice	6	4, 826, 809	10^{-6}	1/46,656	2.131238579750061E−5	n/a	n/a	2.143591779395712E−5	n/a
Dice	6	4, 826, 809	10^{-12}	1/46,656	2.143347024102793E−5	9	2.6	2.1433470555450964E−5	n/a
Din. crypt.	9	855, 095	10^{-6}	1/256	0.00390625	4	.71	0.00390625	.97
Din. crypt.	9	855, 095	10^{-12}	1/256	0.00390625	4	1	0.00390625	1
Biased coins	11	177, 147	10^{-6}	1/177,147	5.645029269476758E−6	10	.11	5.645029269476758E−6	n/a
Biased coins	11	177, 147	10^{-12}	1/177147	5.645029269476758E−6	10	.15	5.645029269476758E−6	.1
Din. phil.	3	956	10^{-6}	27	26.999990834143837	1	.13	27.00000014876298	.28
Din. Phil.	3	956	10^{-12}	27	26.99999999999123	1	.14	27.000000000000142	.22
Lead. elec.	4	12, 302	10^{-6}	256/49	5.2244897630362175	3	12.2	5.224489867467293	30.1
Lead. elec.	4	12, 302	10^{-12}	256/49	5.224489795918261	3	12.4	5.22448979591833	29.7

Columns 1–5 describe the benchmark examples. Columns 6 and 9 are the approximate values generated by value iteration and interval iteration, respectively. Columns 8 and 10 report the running times for each engine (including the time for model construction). Column 7 gives the number of fixpoints checks computed by Algorithm 2. We do not report the number of fixpoint checks for interval iteration as the implementation of SHARPEN for this technique always calculates a single fixpoint. The probabilities given in columns 5,6 and 9 represent the probability of satisfying the given property from the initial state. The model types and properties for the evaluated examples are the same as in Table 1. Both iterative techniques were invoke using the HYBRID engine with the options `javamaxmem=4g` and `cuddmaxmem=4g`. We write n/a in column 10 if no fixpoint was found by the SHAPREN procedure

8 Conclusion

Techniques for exact model checking allow one to avoid logical errors in system analysis that can arise due to approximation techniques. We presented an algorithm and tool, RationalSearch, that computes the exact probabilities described by PCTL formulas for DTMCs and MDPs. Our tool works by sharpening approximate results obtained through value iteration, allowing it to benefit from the performance enhancements gained through approximation techniques. Our experimental evaluation concurs with this hypothesis, and shows that our approach often performs significantly better than existing exact quantitative model checking tools while also scaling to large model sizes. We believe there are also performance enhancements that can be achieved by a tighter integration with the Kwek–Mehlhorn algorithm, wherein computations from previous iterations can be reused.

Acknowledgements We thank the anonymous reviewers for their useful comments. In particular, we would like to thank the reviewer who pointed out that it is insufficient to check that a proposed solution was a solution to a system of linear equations when computing max reachability probabilities and min expected costs.

A Proof of the claim in Theorem 2

It can be shown easily that f is non-expanding, i.e, for any $\bar{x}_1, \bar{x}_2 \in \mathcal{U}$,

$$||f(\bar{x}_2) - f(\bar{x}_1)|| \leq ||\bar{x}_1 - \bar{x}_2||.$$

We will assume without loss of generality that $\mathsf{Prob}_1^{\mathsf{min}}[\xi]$ consists of exactly one element z_0. Further, we assume that $\mathsf{Prob}_0^{\mathsf{min}}[\xi]$ consists of at least 1 element as otherwise the claim is trivially true.

Let $Z^? = Z \backslash (\mathsf{Prob}_0^{\mathsf{min}}[\xi] \cup \mathsf{Prob}_1^{\mathsf{min}}[\xi])$. For $\bar{x} \in \mathcal{U}$, $z \in Z^?$ and $\alpha \in \mathsf{enabled}(z)$, we denote the sum $\sum\limits_{z' \in Z} \Delta(z, \alpha, z') \cdot \bar{x}(z')$ by $h_{\bar{x},z,\alpha}$. By definition

$$f(\bar{x})(z) = \min_{\alpha \in \mathsf{enabled}(z)} h_{\bar{x},z,\alpha}.$$

Fix $\bar{x}, \bar{y} \in \mathcal{U}$. The definition of $Z^?$ implies that for any scheduler $\mathfrak{S}$, the probability of reaching z_0 from a state $z \in Z^?$ is not zero. From this, there it follows that there is an enumeration $z_1, z_2, \ldots z_r$ of $Z^?$ such that for any $1 \leq i \leq r$ and any action $\alpha \in \mathsf{enabled}(z_i)$, $\Delta(z_i, \alpha, z_j) > 0$ for some $0 \leq j < i$.

We will show by induction on $0 \leq i \leq r$,

$$|f^{i+1}(\bar{x})(z_i) - f^{i+1}(\bar{y})(z_i)| \leq (1 - p_{\mathsf{min}}^i)||\bar{x} - \bar{y}||.$$

Observe that this suffices to conclude the claim since this implies for any $z_i \in Z^?$,

$$\begin{aligned} |f^n(\bar{x})(z_i) - f^n(\bar{y})(z_i)| &\leq ||f^{i+1}(\bar{x})(z_i) - f^{i+1}(\bar{y})(z_i)|| \\ &\leq (1 - p_{\mathsf{min}}^i)||\bar{x} - \bar{y}|| \leq (1 - p_{\mathsf{min}}^n)||\bar{x} - \bar{y}||. \end{aligned}$$

Now we show, by induction, that for each $0 \leq i \leq r$, $|f^{i+1}(\bar{x})(z_i) - f^{i+1}(\bar{y})(z_i)| \leq (1 - p_{\mathsf{min}}^i)||\bar{x} - \bar{y}||$.
Base case: The base case is trivial since $f(\bar{x})(z_0) = 1 = f(\bar{y})(z_0)$.

Induction hypothesis: Let $|f^{i+1}(\bar{x})(z_i) - f^{i+1}(\bar{y})(z_i)| \leq (1 - p^i_{\mathsf{min}})||\bar{x} - \bar{y}||$ for each $0 \leq i \leq \ell$. Fix $\beta \in \mathsf{enabled}(z_{\ell+1})$. Denote the set $\{z_0, z_1, \dots, z_\ell\}$ by Z_ℓ. We have that

$$\begin{aligned} h_{f^{\ell+2}(\bar{x}), z_{\ell+1}, \beta} &= \sum_{z' \in Z} \Delta(z_{\ell+1}, \beta, z') \cdot f^{\ell+1}(\bar{x})(z') \\ &= h_{f^{\ell+2}(\bar{y}), z_{\ell+1}, \beta} + \sum_{z' \in Z} \Delta(z_{\ell+1}, \beta, z') \cdot (f^{\ell+1}(\bar{x})(z') - f^{\ell+1}(\bar{y})(z')) \\ &= h_{f^{\ell+2}(\bar{y}), z_{\ell+1}, \beta} + \sum_{z' \in Z_\ell} \Delta(z_{\ell+1}, \beta, z') \cdot (f^{\ell+1}(\bar{x})(z') - f^{\ell+1}(\bar{y})(z')) \\ &\quad + \sum_{z' \in Z \setminus Z_\ell} \Delta(z_{\ell+1}, \beta, z') \cdot (f^{\ell+1}(\bar{x})(z') - f^{\ell+1}(\bar{y})(z')). \end{aligned}$$

Now, note that $(1 - p^i_{\mathsf{min}}) \leq (1 - p^\ell_{\mathsf{min}})$ for each $i \leq \ell$. Thus, we get by induction hypothesis,

$$\begin{aligned} h_{f^{\ell+2}(\bar{x}), z_{\ell+1}, \beta} \leq h_{f^{\ell+2}(\bar{y}), z_{\ell+1}, \beta} + (1 - p^\ell_{\mathsf{min}}) \sum_{z' \in Z_\ell} \Delta(z_{\ell+1}, \beta, z') \cdot ||\bar{x} - \bar{y}|| \\ \sum_{z' \in Z \setminus Z_\ell} \Delta(z_{\ell+1}, \beta, z') \cdot (f^{\ell+1}(\bar{x})(z') - f^{\ell+1}(\bar{y})(z')). \end{aligned}$$

As f is non-expanding, we get that

$$\begin{aligned} h_{f^{\ell+2}(\bar{x}), z_{\ell+1}, \beta} &\leq h_{f^{\ell+2}(\bar{y}), z_{\ell+1}, \beta} + (1 - p^\ell_{\mathsf{min}}) \sum_{z' \in Z_\ell} \Delta(z_{\ell+1}, \beta, z') \cdot ||\bar{x} - \bar{y}|| \\ &\quad + \sum_{z' \in Z \setminus Z_\ell} \Delta(z_{\ell+1}, \beta, z') \cdot ||\bar{x} - \bar{y}|| \\ &\leq h_{f^{\ell+2}(\bar{y}), z_{\ell+1}, \beta} + ||\bar{x} - \bar{y}|| \cdot \sum_{z' \in Z} \Delta(z_{\ell+1}, \beta, z') \\ &\quad - p^\ell_{\mathsf{min}} ||\bar{x} - \bar{y}|| \cdot \sum_{z' \in Z_\ell} \Delta(z_{\ell+1}, \beta, z') \\ &\leq h_{f^{\ell+2}(\bar{y}), z_{\ell+1}, \beta} + ||\bar{x} - \bar{y}||(1 - p^\ell_{\mathsf{min}} \sum_{z' \in Z_\ell} \Delta(z_{\ell+1}, \beta, z')). \end{aligned}$$

By construction, $\sum_{z' \in Z_\ell} \Delta(z_{\ell+1}, \beta, z')) \geq p_{\mathsf{min}}$ and hence

$$h_{f^{\ell+2}(\bar{x}), z_{\ell+1}, \beta} \leq h_{f^{\ell+2}(\bar{y}), z_{\ell+1}, \beta} + ||\bar{x} - \bar{y}||(1 - p^{\ell+1}_{\mathsf{min}})||.$$

Now, we have that

$$f^{\ell+2}(\bar{x})(z_{\ell+1}) \leq h_{f^{\ell+2}(\bar{x}), z_{\ell+1}, \beta} \leq h_{f^{\ell+2}(\bar{y}), z_{\ell+1}, \beta} + ||\bar{x} - \bar{y}||(1 - p^{\ell+1}_{\mathsf{min}})||.$$

As β is arbitrary, the above inequality also holds for the β that minimizes $h_{f^{\ell+2}(\bar{y}), z_{\ell+1}, \beta}$. Hence,

$$f^{\ell+2}(\bar{x})(z_{\ell+1}) \leq f^{\ell+2}(\bar{y})(z_{\ell+1}) + ||\bar{x} - \bar{y}||(1 - p^{\ell+1}_{\mathsf{min}})||.$$

Similarly, we can show that

$$f^{\ell+2}(\bar{y})(z_{\ell+1}) \leq f^{\ell+2}(\bar{x})(z_{\ell+1}) + ||\bar{x} - \bar{y}||(1 - p^{\ell+1}_{\mathsf{min}})||.$$

Thus, we get

$$|f^{\ell+2}(\bar{x})(z_{\ell+1}) - f^{\ell+2}(\bar{y})(z_{\ell+1})| \leq (1 - p^{\ell+1}_{\mathsf{min}})||\bar{x} - \bar{y}||$$

as required.

References

1. (2017) Ensuring the reliability of your model checker: interval iteration for Markov decision processes. https://wwwtcs.inf.tu-dresden.de/ALGI/PUB/CAV17/
2. (2017) PRISM benchmark suite,http://www.prismmodelchecker.org/benchmarks/. Accessed 5 May 2020
3. (2017) PRISM case studies, http://www.prismmodelchecker.org/casestudies/. Accessed 5 May 2020
4. (2019) Apfloat. http://www.apfloat.org/
5. (2019) CUDD. http://vlsi.colorado.edu/~fabio/CUDD/html/
6. (2019) GNU multiple precision arithmetic library. https://gmplib.org/
7. (2019) JScience. http://jscience.org/
8. (2019) RationalSearch. https://publish.illinois.edu/rationalmodelchecker/
9. de Alfaro L (1997) Formal verification of probabilistic systems. Ph.D. thesis, Stanford University
10. Baier C, Katoen JP (2008) Principles of model checking (representation and mind series). The MIT Press, Cambridge
11. Baier C, Klein J, Leuschner L, Parker D, Wunderlich S (2017) Ensuring the reliability of your model checker: interval iteration for Markov decision processes. In: Computer aided verification
12. Banach S (1922) Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. Fundamenta Mathematicae 3(1):133–181
13. Bauer MS, Mathur U, Chadha R, Sistla AP, Viswanathan M (2017) Exact quantitative probabilistic model checking through rational search. In: Proceedings of the 17th conference on formal methods in computer-aided design, FMCAD Inc, Austin, TX, FMCAD '17, pp 92–99. https://doi.org/10.23919/FMCAD.2017.8102246. http://dl.acm.org/citation.cfm?id=3168451.3168475
14. Benini L, Bogliolo A, Paleologo GA, De Micheli G (1999) Policy optimization for dynamic power management. IEEE Trans Comput-Aided Des Integr Circuits Syst 13:813–833
15. Bhaduri D, Shukla SK, Graham PS, Gokhale MB (2007) Reliability analysis of large circuits using scalable techniques and tools. IEEE Trans Circuits Syst I: Regul Pap 54:2447–2460
16. Bianco A, de Alfaro L (1995) Model checking of probabilistic and nondeterministic systems. In: 15th Conference foundations of software technology and theoretical computer science, lecture notes in computer science. Springer, Berlin, vol 1026, pp 499–513
17. Brázdil T, Chatterjee K, Chmelík M, Forejt V, Křetínský J, Kwiatkowska M, Parker D, Ujma M (2014) Verification of markov decision processes using learning algorithms. In: Automated technology for verification and analysis. Springer, Cham, pp 98–114
18. Bryant RE (1986) Graph-based algorithms for boolean function manipulation. EEE Trans Comput 100(8):677–691
19. Chatterjee K, Henzinger TA (2008) Value iteration. Springer, Berlin, pp 107–138. https://doi.org/10.1007/978-3-540-69850-0_7
20. Chaum D (1988) The dining cryptographers problem: Unconditional sender and recipient untraceability. J Cryptol 1(1):65–75
21. Daws C (2004) Symbolic and parametric model checking of discrete-time Markov chains. In: International Colloquium on theoretical aspects of computing. Springer, Berlin, pp 280–294
22. Dehnert C, Junges S, Katoen JP, Volk M (2017) A storm is coming: A modern probabilistic model checker. In: 29th international conference computer aided verification CAV 2017
23. Dehnert C, Junges S, Jansen N, Corzilius F, Volk M, Bruintjes H, Katoen JP, Abraham E (2015) Prophesy: a probabilistic parameter synthesis tool. In: International conference on computer aided verification, CAV
24. van Dijk T, van de Pol J (2015) Sylvan: Multi-core decision diagrams. In: International conference on tools and algorithms for the construction and analysis of systems. Springer, Berlin, pp 677–691
25. Dijkstra EW (1982) Self-stabilization in spite of distributed control. In: Selected writings on computing: a personal perspective. Springer, Berlin
26. Duflot M, Kwiatkowska M, Norman G, Parker D (2006) A formal analysis of bluetooth device discovery. Int J Softw Tools Technol Transf (STTT) 8(6):621–632
27. Forejt V, Kwiatkowska M, Norman G, Parker D (2011a) Automated verification techniques for probabilistic systems. In: International school on formal methods for the design of computer, communication and software systems. Springer, Berlin, pp 53–113
28. Forejt V, Kwiatkowska MZ, Norman G, Parker D (2011b) Automated verification techniques for probabilistic systems. In: Formal methods for eternal networked software systems—11th international school on formal methods for the design of computer, communication and software systems, SFM, pp 53–113
29. Fujita M, McGeer PC, Yang JY (1997) Multi-terminal binary decision diagrams: an efficient data structure for matrix representation. Formal Methods Syst Des 10(2–3):149–169
30. Giro S (2012) Efficient computation of exact solutions for quantitative model checking. In: Proceedings of 10th workshop on quantitative aspects of programming languages (QAPL'12)

31. Haddad S, Monmege B (2014) Reachability in MDPS: refining convergence of value iteration. In: International workshop on reachability problems. Springer, Berlin, pp 125–137
32. Hahn EM, Hermanns H, Wachter B, Zhang L (2010) PARAM: a model checker for parametric Markov models. In: International conference on computer aided verification (CAV'10)
33. Hahn EM, Han T, Zhang L (2011a) Synthesis for PCTL in parametric Markov decision processes. In: NASA formal methods symposium. Springer, Berlin, pp 146–161
34. Hahn EM, Hermanns H, Zhang L (2011b) Probabilistic reachability for parametric Markov models. Int J Softw Tools Technol Transf 13(1):3–19
35. Han J, Chen H, Boykin E, Fortes J (2011) Reliability evaluation of logic circuits using probabilistic gate models. Microelectron Reliab 51:468–476
36. Hoey J, St-Aubin R, Hu A, Boutilier C (1999) Spudd: Stochastic planning using decision diagrams. In: Proceedings of the fifteenth conference on uncertainty in artificial intelligence
37. Hopcroft JE (2008) Introduction to automata theory, languages, and computation. Pearson Education India, Delhi
38. Jeannet B, D'Argenio P, Larsen K (2002) Rapture: a tool for verifying Markov decision processes. In: Proceeding of tools day, affiliated to 13th international conference concurrency theory (CONCUR'02)
39. Katoen JP, Khattri M, Zapreevt I (2005) A Markov reward model checker. In: Second international conference on the quantitative evaluation of systems (QEST'05), IEEE
40. Kwek S, Mehlhorn K (2003) Optimal search for rationals. Inf Process Lett 86(1):23–26
41. Kwiatkowska M, Norman G, Sproston J (2002) Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In: Proceedings of 2nd joint international workshop on process algebra and probabilistic methods, performance modeling and verification (PAPM/PROBMIV'02)
42. Kwiatkowska M, Norman G, Sproston J (2003) Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. Formal Aspects Comput 14(3):295–318
43. Kwiatkowska M, Norman G, Parker D (2004) Controller dependability analysis by probabilistic model checking. In: 11th IFAC symposium on information control problems in manufacturing (INCOM'04)
44. Kwiatkowska M, Norman G, Parker D (2011) Prism 4.0: verification of probabilistic real-time systems. In: International conference on computer aided verification. Springer, Berlin, pp 585–591
45. McMillan KL (1993) Symbolic model checking. Kluwer Academic Publishers, Norwell
46. Mohyuddin N, Pakbaznia E, Pedram M (2011) Probabilistic error propagation in a logic circuit using the Boolean difference calculus. In: Advanced techniques in logic synthesis, optimizations and applications. Springer, Berlin, pp 359–381
47. Norman G, Parker D, Kwiatkowska M, Shukla S (2005) Evaluating the reliability of NAND multiplexing with PRISM. IEEE Trans Comput-Aided Des Integr Circuits Syst 24:1629–1637
48. Parker D (2002) Implementation of symbolic model checking for probabilistic systems. Ph.D. thesis, University of Birmingham
49. Qiu Q, Qu Q, Pedram M (2001) Stochastic modeling of a power-managed system-construction and optimization. IEEE Trans Comput-Aided Des Integr Circuits Syst 20:1200–1217
50. Rabin M (1983) Randomized Byzantine generals. In: Proceedings of symposium on foundations of computer science, pp 403–409
51. Rutten J, Kwiatkowska M, Norman G, Parker D (2004a) Mathematical techniques for analyzing concurrent and probabilistic systems. In: Panangaden P, van Breugel F (eds) CRM monograph series, vol 23. American Mathematical Society, Providence
52. Rutten JJ, Kwiatkowska M, Norman G, Parker D (2004b) Mathematical techniques for analyzing concurrent and probabilistic systems. American Mathematical Society, Providence
53. St-Aubin R, Hoey J, Boutilier C (2001) APRICODD: approximate policy construction using decision diagrams. In: Advances in neural information processing systems, pp 1089–1095
54. Wimmer R, Kortus A, Herbstritt M, Becker B (2008) Probabilistic model checking and reliability of results. In: 11th IEEE workshop on design and diagnostics of electronic circuits and systems, 2008. DDECS, IEEE, pp 1–6

Affiliations

Umang Mathur[1] · **Matthew S. Bauer**[2] · **Rohit Chadha**[3] · **A. Prasad Sistla**[4] · **Mahesh Viswanathan**[1]

Matthew S. Bauer
mbauer@galois.com

Rohit Chadha
chadhar@missouri.edu

A. Prasad Sistla
sistla@cs.uic.edu

Mahesh Viswanathan
vmahesh@illinois.edu

[1] University of Illinois, Urbana Champaign, Champaign, USA

[2] Galois Inc., Portland, USA

[3] University of Missouri, Columbia, USA

[4] University of Illinois, Chicago, Chicago, USA