# Safety-Constrained Reinforcement Learning for MDPs[⋆]

Sebastian Junges[1], Nils Jansen[1], Christian Dehnert[1], Ufuk Topcu[2], and Joost-Pieter Katoen[1]

[1] RWTH Aachen University
[2] University of Texas at Austin

**Abstract.** We consider controller synthesis for stochastic and partially unknown environments in which safety is essential. Specifically, we abstract the problem as a Markov decision process in which the expected performance is measured using a cost function that is *unknown* prior to run-time exploration of the state space. Standard learning approaches synthesize cost-optimal strategies without guaranteeing safety properties. To remedy this, we first compute safe, permissive strategies. Then, exploration is constrained to these strategies and thereby meets the imposed safety requirements. Exploiting an iterative learning procedure, the resulting strategy is safety-constrained and optimal. We show correctness and completeness of the method and discuss the use of several heuristics to increase its scalability. Finally, we demonstrate the applicability by means of a prototype implementation.

## 1 Introduction

*Probabilistic model checking.* Many formal system models are inherently stochastic, consider for instance randomized distributed algorithms (where randomization breaks the symmetry between processes), security (e.g., key generation at encryption), systems biology (where species randomly react depending on their concentration), or embedded systems (interacting with unknown and varying environments). These various applications made the *verification* of stochastic systems such as discrete-time Markov chains (MCs) or Markov decision processes (MDPs) an important research topic in the last decade, resulting in several tools like PRISM [1], LiQuoR [2], MRMC [3] or FMurphi [4]. The always growing set of case studies in the PRISM benchmark suite [5] witnesses the applicability of MDP and MC model checking.

*Controller synthesis.* Contrarily, controller synthesis is a relatively new topic in this setting. Consider a controllable system like, e. g., a robot or some other

⋆ This work is supported by the Excellence Initiative of the German Research Council and the Sino-German project CAP. UT's work has been partly funded by the awards AFRL # FA8650-15-C-2546, ONR # N000141310778, ARO # W911NF-15-1-0592, NSF # 1550212 and DARPA # W911NF-16-1-0001

machine which is embedded into an environment. Having a formal model of both the controllable entity and the environment, the goal is to synthesize a controller that satisfies certain requirements. Again, often faithful models are stochastic, imagine, e. g., sensor imprecisions of a robot, message loss, or unpredictable behavior of the environment. Moreover, it might be the case that certain information—such as cost caused by energy consumption—is not exactly known prior to exploring and observation.

*Our problem.* Given an MDP with a cost structure, synthesize an optimal strategy subject to safety constraints, where optimality refers to expected performance (cost). This multi-objective model checking problem is studied in [6–8]. But what if the cost function is not known? Consider for instance the following motion planning scenario, placed in a grid-world where a robot wants to move to a certain position. Acting unpredictably, a janitor moves randomly through the grid. The robot reaches its goal *safely* if it moves according to a strategy that avoids the janitor. Moreover, each movement of the robot occasions cost depending on the surface. However, the robot only learns the actual costs during physically executing actions within the environment; this requires the exclusion of unsafe behavior prior to exploration. Consequently, a safe strategy for the robot which simultaneously induces minimal cost is to be found.

We model robot behavior by an MDP and the stochastic behavior of the environment by a MC. We are given a *safety condition* specified as a probabilistic reachability objective. Additionally, we have a *performance condition* bounding the *expected costs* for reaching a certain goal. A significant problem we are facing is that the costs of certain actions are not known before they are executed. This calls for using reinforcement learning [9] algorithms like Q-learning [10], where optimal strategies are obtained without prior knowledge about the system. While this is usually a suitable solution, in this case we have to ensure that no unsafe actions are taken during exploration to ensure an optimal and safe strategy.

*Our approach.* The setting does neither allow for using plain verification nor direct reinforcement learning. On the one hand, verifying safety and performance properties—in the form of multi-objective model checking—is not possible because the costs of actions are not known. On the other hand, in practice learning means that the robot will explore parts of the system. Doing that, we need to ensure that all unsafe behavior is avoided *beforehand*. Our solution to these problems is to use the notion of *permissive schedulers*. In contrast to standard schedulers, where for each system run the next action to take is fixed, more permissiveness is given in the sense that several actions are allowed. The first step is to compute a *safe* permissive scheduler which allows only safe behavior. The system is then restricted according to this scheduler (or strategy) and fit for *safe exploration*.

It would be desirable to compute a permissive scheduler which encompasses the set of *all* safe schedulers. Having this would ensure that via reinforcement learning a safe scheduler inducing *optimal* cost would obtained. Unfortunately, there is no efficient representation of such a *maximal permissive scheduler*. Therefore, we propose an iterative approach utilizing SMT-solving where a safe permissive scheduler is computed. Out of this, reinforcement learning determines the *locally optimal* scheduler. In the next iteration, this scheduler is explicitly excluded and a new permissive scheduler is obtained. This is iterated until the performance criterion is satisfied or until the solution is determined to be globally optimal which can be done using known lower bounds on the occurring costs.

*Related work.* In [11], the computation of permissive schedulers for stochastic 2-player games is proposed for reward properties without additional safety-constraints. A dedicated mixed-integer linear programming (MILP) encoding optimizes w. r. t. certain *penalties* for actions. In [12], permissive safe scheduling is investigated for transition systems and LTL properties. Safe or constrained (e.g., by temporal logic specifications) exploration has also been investigated in the learning literature. Some recent examples include [13, 14]. In [15], safety guarantees are added via the usage of Gaussian processes. An overview on safe exploration using reinforcement learning can be found in [16].

*Summary of the contributions.* We give the first approach to controller synthesis for stochastic systems regarding safety and performance in a setting where models are known but costs are not. This encompasses:

– an iterative approach to the computation of safe permissive schedulers based on SMT-solving;
– exploitation of permissive schedulers for reinforcement learning towards globally optimal solutions;
– a discussion of several heuristics to both speed up the computations and avoid too many iterations; and
– a prototype implementation showing promising results on several case studies.

The paper is structured as follows: First, we provide basic notations and formal prerequisites in Section 2. In Section 3 we introduce our notion of permissive schedulers, discuss efficient representations, and introduce technicalities that are needed afterwards. Section 4 presents our main results on computing safe and optimal schedulers. After presenting several case studies and benchmark results in Section 5, we finally draw a conclusion and point to future work in Section 6.

## 2 Preliminaries

In this section, we introduce the required models and specifications considered in this paper, and provide a formal problem statement.

*Models.* For a set $X$, let $2^X$ denote the power set of $X$. A *probability distribution* over a finite or countably infinite set $X$ is a function $\mu\colon X \to [0,1] \subseteq \mathbb{R}$ with $\sum_{x\in X} \mu(x) = \mu(X) = 1$. In this paper, all probabilities are taken from $\mathbb{Q}$. Let the set of all distributions on $X$ be denoted by $\mathit{Distr}(X)$. The set $\mathit{supp}(\mu) = \{x \in X \mid \mu(x) > 0\}$ is the *support* of $\mu \in \mathit{Distr}(X)$. If $\mu(x) = 1$ for $x \in X$ and $\mu(y) = 0$ for all $y \in X \setminus \{x\}$, $\mu$ is called a *Dirac distribution.*

**Definition 1. (MDP)** *A* Markov decision process (MDP) $\mathcal{M} = (S, s_I, Act, \mathcal{P})$ *is a tuple with a finite set $S$ of states, a unique initial state $s_I \in S$, a finite set $Act$ of actions, and a (partial) probabilistic transition function $\mathcal{P} \colon S \times Act \to Distr(S)$.*

MDPs operate by means of *nondeterministic choices* of actions at each state, whose successors are then determined *probabilistically* w. r. t. the associated probability distribution. The set of *enabled* actions at state $s \in S$ is denoted by $Act(s) = \{a \in Act \mid \exists \mu \in Distr(S).\, \mu = \mathcal{P}(s, \alpha)\}$. To avoid deadlock states, we assume that $|Act(s)| \geq 1$ for all $s \in S$. A *cost function* $\rho \colon S \times Act \to \mathbb{R}_{\geq 0}$ for an MDP $\mathcal{M}$ adds a cost to each *transition* $(s, a) \in S \times Act$ with $a \in Act(s)$.

A *path* in an $\mathcal{M}$ is a finite (or infinite) sequence $\pi = s_0 a_0 s_1 a_1 \ldots$ with $\mathcal{P}(s_i, \alpha, s_{i+1}) > 0$ for all $i \geq 0$. The set of all paths in $\mathcal{M}$ is denoted by $Paths^{\mathcal{M}}$, all paths starting in state $s \in S$ by $Paths^{\mathcal{M}}(s)$. The cost of finite path $\pi$ is defined as the sum of the costs of all transitions in $\pi$, i.e., $\rho(\pi) = \sum_{i=0}^{n-1} \rho(s_i, a_i)$ where $n$ is the number of transitions in $\pi$.

If $|Act(s)| = 1$ for all $s \in S$, all actions can be disregarded and the MDP $\mathcal{M}$ reduces to a *discrete-time Markov chain (MC)*, sometimes denoted by $\mathcal{D}$, yielding a transition probability transition function of the form $\mathcal{P} \colon S \to Distr(S)$. The *unique probability measure* $\Pr^{\mathcal{D}}(\mathit{\Pi})$ for set $\mathit{\Pi}$ of infinite paths of MC $\mathcal{D}$ can be defined by the usual cylinder set construction, see [17] for details. The *expected cost* of the set $\mathit{\Pi}$ of paths, denoted by $\mathrm{EC}^{\mathcal{D}}(\mathit{\Pi})$, is defined as $\sum_{\pi \in \mathit{\Pi}} \Pr(\pi) \cdot \rho(\pi)$.

In order to define a probability measure and expected cost on MDPs, the nondeterministic choices of actions are resolved by so-called *schedulers*[3]. As in [11], for practical reasons we restrict ourselves to *memoryless* schedulers; more details about schedulers can be found in [17].

**Definition 2. (Scheduler)** *A* scheduler *for an MDP $\mathcal{M}$ is a function $\sigma \colon S \to Distr(Act)$ such that $\sigma(s)(a) > 0$ implies $a \in Act(s)$. Schedulers using only Dirac distributions are called* deterministic. *The set of all schedulers over $\mathcal{M}$ is denoted by $Sched^{\mathcal{M}}$.*

Deterministic schedulers are functions of the form $\sigma \colon S \to Act$ with $\sigma(s) \in Act(s)$. Schedulers that are not deterministic are also called *randomized.* Applying a scheduler to an MDP yields a so-called *induced Markov chain*, as all nondeterminism is resolved.

**Definition 3. (Induced MC)** *Let MDP $\mathcal{M} = (S, s_I, Act, \mathcal{P})$ and scheduler $\sigma \in Sched^{\mathcal{M}}$. The* MC induced by $\mathcal{M}$ and $\sigma$ *is $\mathcal{M}^\sigma = (S, s_I, Act, \mathcal{P}^\sigma)$ where*

$$\mathcal{P}^\sigma(s, s') = \sum_{a \in Act(s)} \sigma(s)(a) \cdot \mathcal{P}(s, a)(s') \quad \text{for all } s, s' \in S \ .$$

Intuitively, the transition probabilities in $\mathcal{M}^\sigma$ are obtained w. r. t. the random choices of action of the scheduler.

[3] Also referred to as strategies or policies.

*Remark 1. Deterministic schedulers* pick just one action at each state and the associated probability distribution determines the probabilities. In this case we write for all states $s \in S$ and $a \in \mathit{Act}$ with $\sigma(s)(a) = 1$:

$$\mathcal{P}^{\sigma}(s, s') = \mathcal{P}(s, a)(s') \ .$$

*Specifications.* Specifications are given by combining *reachability properties* and *expected cost properties*. A reachability property $\mathbb{P}_{\leq\lambda}(\lozenge T)$ with upper probability bound $\lambda \in [0, 1] \subseteq \mathbb{Q}$ and target set $T \subseteq S$ constrains the probability to finally reach $T$ from $s_I$ in $\mathcal{M}$ to be at most $\lambda$. Analogously, expected cost properties $\mathbb{E}_{\leq\kappa}(\lozenge G)$ impose an upper bound $\kappa \in \mathbb{Q}$ on the expected cost to reach goal states $G \subseteq S$. Combining both types of properties, the intuition is that a set of bad states $T$ shall only be reached with a certain probability $\lambda$ (safety specification) while the expected cost for reaching a set of goal states $G$ has to be below $\kappa$ (performance specification). This can be verified using multi-objective model checking [6–8], provided all problem data (i.e., probabilities and costs) are a-priori known.

We overload the notation $\lozenge T$ to denote both a reachability property and the set of all paths that finally reach $T$ from the initial state $s_I$ of an MC. The probability and the expected cost for reaching $T$ from $s_I$ are denoted by $\Pr(\lozenge T)$ and $\mathrm{EC}(\lozenge T)$, respectively. Hence, $\Pr^{\mathcal{D}}(\lozenge T) \leq \lambda$ and $\mathrm{EC}^{\mathcal{D}}(\lozenge G) \leq \kappa$ express that the properties $\mathbb{P}_{\leq\lambda}(\lozenge T)$ and $\mathbb{E}_{\leq\kappa}(\lozenge G)$ respectively are satisfied by MC $\mathcal{D}$.

An MDP $\mathcal{M}$ satisfies both reachability property $\varphi$ and expected cost property $\psi$, iff *for all* schedulers $\sigma$ it holds that the induced MC $\mathcal{M}^{\sigma}$ satisfies the properties $\varphi$ and $\psi$, i.e., $\mathcal{M}^{\sigma} \models \varphi$ and $\mathcal{M}^{\sigma} \models \psi$. In our setting, we are rather interested in the so-called *synthesis problem*, where the aim is to find a scheduler $\sigma$ such that both properties are satisfied (while this does not necessarily hold for all schedulers). If $\mathcal{M}^{\sigma} \models \varphi$, scheduler $\sigma$ is said to *admit* the property $\varphi$; this is denoted by $\sigma \models \varphi$.

*Formal problem statement.* Given an MDP $\mathcal{M}_1$ modeling possible controllable behaviors and an MC $\mathcal{D}$ modeling the stochastic behavior of an environment, the synchronous product (see e. g. [18]) is denoted by $\mathcal{M}_1 \times \mathcal{D} = \mathcal{M} = (S, s_I, \mathit{Act}, \mathcal{P})$. Let $\rho$ be a cost function over $\mathcal{M}$ that is *unknown* to the robot prior to exploring the state space. We assume that for each transition $(s, a)$, the cost is bounded from below and from above, i. e. $l_{(s,a)} \leq \rho(s, a) \leq u_{(s,a)}$ with $l_{(s,a)}, u_{(s,a)} \in \mathbb{Q}$ for any $(s, a) \in S \times \mathit{Act}$. Let safety specification $\varphi = \mathbb{P}_{\leq\lambda}(\lozenge T)$ and performance specification $\psi = \mathbb{E}_{\leq\kappa}(\lozenge G)$ for $\mathcal{M}$ with $T, G \subseteq S$.

The *synthesis problem* is to find a scheduler $\sigma \in \mathit{Sched}^{\mathcal{M}}$ such that $\mathcal{M}^{\sigma} \models \varphi$ and $\mathcal{M}^{\sigma} \models \psi$. The *optimal* synthesis problem is to find a scheduler $\sigma^* \in \mathit{Sched}^{\mathcal{M}}$ such that $\mathcal{M}^{\sigma^*} \models \varphi$ and $\sigma^*$ minimizes the expected cost to reach $G$.

## 3 Permissive schedulers

As mentioned before, we will utilize the notion of *permissive* schedulers, where not all nondeterminism is to be resolved. A permissive scheduler may select a

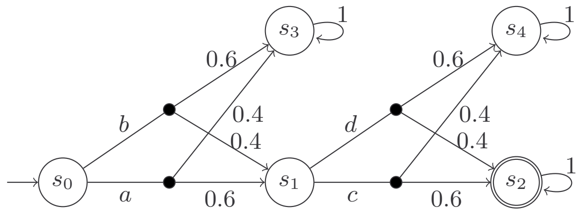


**Fig. 1.** Example MDP $\mathcal{M}$ illustrating conflicting schedulers

set of actions at each state, such that at a state there might be several possible actions or probability distributions over actions left open. In this sense, permissive schedulers can be seen as sets of schedulers. Here, we discuss properties and efficient representations that are needed later on. Analogously to schedulers, we consider only memoryless notions.

**Definition 4. (Permissive scheduler)** *A* permissive scheduler *of MDP* $\mathcal{M} = (S, s_I, Act, \mathcal{P})$ *is a function* $\theta\colon S \to 2^{Distr(Act)} \setminus \emptyset$ *and* $\forall s \in S. \forall \mu \in \theta(s). \, supp(\mu) \subseteq Act(s)$. *The set of all permissive schedulers for* $\mathcal{M}$ *is* $PSched^{\mathcal{M}}$.

Intuitively, at each state there is not only one but several distributions over actions available. *Deterministic* permissive schedulers are functions of the form $S \to 2^{Act}$, i. e., there are different choices of action left open. We use the following notations for connections to (non-permissive) schedulers.

**Definition 5. (Compliance)** *A scheduler* $\sigma$ *for the MDP* $\mathcal{M}$ *is* compliant *with a permissive scheduler* $\theta$, *written* $\sigma \in \theta$, *iff for all* $s \in S$ *it holds that* $\sigma(s) \in \theta(s)$.

*A permissive scheduler* $\theta_{\mathcal{S}}$ *for* $\mathcal{M}$ *is* induced *by a set of schedulers* $\mathcal{S} \subseteq Sched^{\mathcal{M}}$, *iff for each state* $s \in S$ *and each distribution* $\mu \in \theta_{\mathcal{S}}(s)$ *there is a scheduler* $\sigma \in \mathcal{S}$ *with* $\sigma(s) = \mu$.

We are interested in sets of schedulers that admit our safety specification.

**Definition 6. (Safe and maximal permissive scheduler)** *A permissive scheduler* $\theta \in PSched^{\mathcal{M}}$ *for the MDP* $\mathcal{M}$ *is* safe *for a reachability property* $\varphi = \mathbb{P}_{\leq\lambda}(\lozenge T)$ *iff for all* $\sigma \in \theta$ *it holds that* $\sigma \models \varphi$, *denoted by* $\theta \models \varphi$. *The permissive scheduler* $\theta$ *is called* maximal, *if there exists no scheduler* $\sigma \in Sched^{\mathcal{M}}$ *with* $\sigma \notin \theta$ *and* $\sigma \models \varphi$.

A safe permissive scheduler contains *only* schedulers that admit the safety specification while a maximal safe permissive scheduler contains *all* such schedulers (and probably more). Note that even for a set of safe schedulers, the induced permissive scheduler might be unsafe; contradicting choices might evolve, i. e., choosing a certain action (or distribution) at one state might rule out certain memoryless choices at other states; this is illustrated by the following example.

*Example 1.* Consider the MDP $\mathcal{M}$ depicted in Figure 1, where the only nondeterministic choices occur at states $s_0$ and $s_1$. Assume a reachability property $\varphi = \mathbb{P}_{\leq 0.3}(\Diamond\{s_2\})$. This property is violated by the deterministic scheduler $\sigma_1 \coloneqq \{s_0 \mapsto a, s_1 \mapsto c\}$ as $s_2$ is reached with probability 0.36 exceeding the threshold 0.3. This is the only unsafe scheduler; removing either action $a$ or $c$ from $\mathcal{M}$ leads to a *safe* MDP, i. e. the possible deterministic schedulers $\sigma_2 \coloneqq \{s_0 \mapsto a, s_1 \mapsto d\}$, $\sigma_3 \coloneqq \{s_0 \mapsto b, s_1 \mapsto c\}$, and $\sigma_4 \coloneqq \{s_0 \mapsto b, s_1 \mapsto d\}$ are all safe. However, consider the induced permissive scheduler $\theta_{\sigma_2,\sigma_3,\sigma_4} \in \mathit{PSched}^{\mathcal{M}}$ with $\theta \coloneqq \big\{s_0 \mapsto \{a, b\}, s_1 \mapsto \{b, c\}\big\}$, where in fact all nondeterministic choices are left open. Unfortunately, it holds that the unsafe scheduler $\sigma_1$ is compliant with $\theta_{\sigma_2,\sigma_3,\sigma_4}$, therefore $\theta$ is unsafe.

Example 1 shows that in order to form a safe permissive scheduler it is not sufficient to just consider the set of safe schedulers. Actually, one needs to keep track that the very same safe scheduler is used in every state. Theoretically, this can be achieved by adding finite memory to the scheduler in order to avoid conflicting actions.

A succinct representation of the maximal permissive scheduler can be gained by enumerating all *minimal* sets of conflicting action choices (now only considering deterministic schedulers), and excluding them from all possible schedulers. We investigate the worst case size of such a set. Assume without loss of generality that for all $s \in S$ the sets $\mathit{Act}(s)$ are pairwise disjoint.

**Definition 7. (Conflict set)** *$C \subseteq \mathit{Act}$ is a* conflict set *for MDP $\mathcal{M}$ and property $\varphi$ iff there exists a deterministic scheduler $\sigma \in \mathit{Sched}^{\mathcal{M}}$ such that $(\forall a \in C.\, \exists s \in S.\, \sigma(s) = a)$ and $\sigma \not\models \varphi$. The set of* all conflict sets *for $\mathcal{M}$ and $\varphi$ is denoted by $\mathit{Conf}^{\mathcal{M}}_{\varphi}$. $C \in \mathit{Conf}^{\mathcal{M}}_{\varphi}$ is a* minimal *conflict set iff $\forall C' \subsetneq C.\, C' \notin \mathit{Conf}^{\mathcal{M}}_{\varphi}$.*

**Lemma 1.** *The size of the set of all minimal conflict sets for $\mathcal{M}$ and $\varphi$ potentially grows exponentially in the number of states of $\mathcal{M}$.*

*Proof sketch.* Let $\mathcal{M}_n = (S, s_I, \mathit{Act}, \mathcal{P})$ be given by $S = \{s_0, \ldots, s_n, \bot\}$, $s_I = s_0$, $\mathit{Act} = \{a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}, c, d\}$ and

$$\mathcal{P}(s,\alpha)(t) = \begin{cases} 0.5 & \text{if } i < n, \alpha = a_i, s = s_i, t = s_{i+1} \\ 0.5 & \text{if } i < n, \alpha = a_i, s = s_i, t = \bot \\ 1 & \text{if } i < n, \alpha = b_i, s = s_i, t = s_{i+1} \\ 1 & \text{if } \alpha = c, s = s_n, t = s_n \\ 1 & \text{if } \alpha = d, s = \bot, t = \bot \\ 0 & \text{otherwise} \end{cases}$$

Figure 2 shows the instance $\mathcal{M}_4$ where several copies of the $\bot$-states have been drawn and the $d$ self-loops have been omitted for ease of presentation. Consider the property $\varphi = \mathbb{P}_{\leq \lambda}(\Diamond\{s_n\})$ with $\lambda = 0.5^{\frac{n}{2}+1}$. Choosing any combination of $\frac{n}{2}$

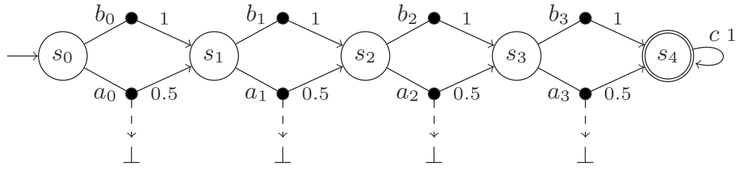


**Fig. 2.** MDP $\mathcal{M}_4$ inducing exponentially many (minimal) conflict sets

of the $b_i$ actions yields a minimal conflict set. Hence, there are at least

$$\binom{n}{\frac{n}{2}} \overset{n:=2m}{=} \frac{(2m)!}{2m!} = \underbrace{\frac{(m+1)}{1} \cdots \frac{2m}{m}}_{m \text{ factors } \geq 2} \geq 2^m \overset{m:=\frac{n}{2}}{=} 2^{\frac{n}{2}} \in \Omega\left(\left(\sqrt{2}\right)^n\right)$$

minimal conflict sets. □

This strongly indicates that an exact representation of the maximal permissive scheduler is not feasible. For algorithmic purposes, we strive for a more compact representation. It seems natural to investigate the possibilities of using MDPs as representation of permissive schedulers. Therefore, analogously to induced MCs for schedulers (cf. Definition 3), we define induced MDPs for permissive schedulers. For a permissive scheduler $\theta \in \mathit{PSched}^{\mathcal{M}}$, we will uniquely identify the nondeterministic choices of probability distributions $\mu \in \theta(s)$ at each state $s \in S$ of the MDP by new actions $a_{s,\mu}$.

**Definition 8. (Induced MDP)** *For an MDP $\mathcal{M} = (S, s_I, \mathit{Act}, \mathcal{P})$ and permissive scheduler $\theta$ for $\mathcal{M}$, the* MDP induced by $\mathcal{M}$ and $\theta$ *is $\mathcal{M}^\theta = (S, s_I, \mathit{Act}^\theta, \mathcal{P}^\theta)$ with $\mathit{Act}^\theta = \{a_{s,\mu} \mid s \in S, \mu \in \theta(s)\}$ and:*

$$\mathcal{P}^\theta(s, a_{s,\mu})(s') = \sum_{a \in \mathit{Act}(s)} \mu(s)(a) \cdot \mathcal{P}(s,a)(s') \quad \text{for } s, s' \in S \text{ and } a_{s,\mu} \in \mathit{Act}^\theta \ .$$

Intuitively, we nondeterministically choose between the distributions over actions induced by the permissive scheduler $\theta$. Note that if the permissive scheduler contains only one distribution for each state, i. e., in fact the permissive scheduler is just a scheduler, the actions can be discarded which yields an induced MC as in Definition 3, making this definition backward compatible.

*Remark 2.* Each deterministic scheduler $\sigma \in \mathit{Sched}^{\mathcal{M}^\theta}$ for the induced MDP $\mathcal{M}^\theta$ *induces a (randomized) scheduler* for the original MDP $\mathcal{M}$. In particular, $\sigma$ induces a scheduler $\sigma' \in \theta$ for $\mathcal{M}$ which is compliant with the permissive scheduler $\theta$: For all $s \in S$ there exists an action $a_{s,\mu} \in \mathit{Act}^\theta$ such that $\sigma(s) = a_{s,\mu}$. The randomized scheduler $\sigma'$ is then given by $\sigma'(s) = \mu$ and it holds that

$$\sum_{a \in \mathit{Act}(s)} \sigma'(s)(a) \cdot \mathcal{P}(s,a)(s') = \mathcal{P}^\theta(s, a_{s,\mu})(s') \ .$$

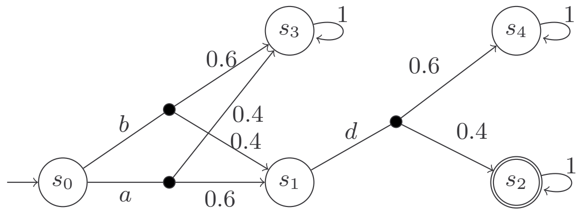


**Fig. 3.** Induced MDP $\mathcal{M}_{\theta_{safe}}$

*Remark 3.* A *deterministic permissive scheduler* $\theta_{\text{det}} \in PSched^{\mathcal{M}}$ for the MDP $\mathcal{M}$ simply *restricts* the nondeterministic choices of the original MDP to the ones that are chosen with probability one by $\theta_{\text{det}}$. The transition probability function $\mathcal{P}^{\theta_{\text{det}}}$ of the induced MDP $\mathcal{M}^{\theta_{\text{det}}}$ can be written as

$$\mathcal{P}^{\theta}(s, a_{s,\mu})(s') = \mathcal{P}(s,a)(s') \quad \text{for all } s \in S \text{ and } a_{s,\mu} \in Act^{\theta_{\text{det}}} \text{ with } \mu(a) = 1 \ .$$

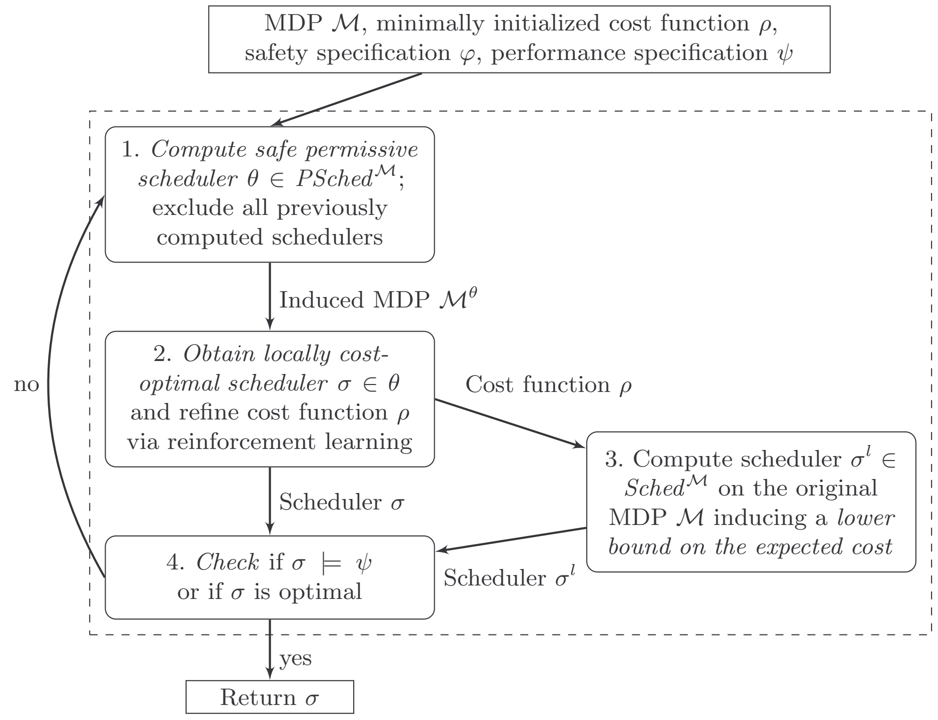
The induced MDP $\mathcal{M}^{\theta}$ can be seen as a sub-MDP $\mathcal{M}^{sub} = (S, s_I, Act, \mathcal{P}^{sub})$ of $\mathcal{M}$ by omitting all actions that are not chosen. Hence, for all $s, s' \in S$:

$$\mathcal{P}^{sub}(s,a)(s') = \begin{cases} \mathcal{P}(s,a)(s') & \text{if } \exists \mu \in \theta(s).\, \mu(a) = 1 \\ 0 & \text{otherwise .} \end{cases}$$

*Example 2.* Recall Example 1 with $\varphi = \mathbb{P}_{\leq 0.3}(\lozenge\{s_2\})$. The MDP $\mathcal{M}^{\theta}$ induced by the permissive scheduler $\theta$ is the same as $\mathcal{M}$, as all available choices of actions are included (see Example 1). Note that we use the simplified notation from Remark 3. However, consider the safe (but not maximal) permissive scheduler $\theta_{safe}$ formed by $\{s_0 \mapsto a, s_1 \mapsto d\}$ and $\{s_0 \mapsto b, s_1 \mapsto d\}$. The induced MDP is the sub-MDP $\mathcal{M}^{\theta_{safe}}$ of $\mathcal{M}$ depicted in Figure 3. This sub-MDP has no scheduler $\sigma$ with $\sigma \not\models \varphi$.

## 4 Safety-constrained reinforcement learning

Recall that the synthesis problem amounts to determining a scheduler $\sigma^*$ of the MDP $\mathcal{M}$ such that $\sigma^*$ admits the safety specification $\varphi$ and minimizes the expected cost (of reaching $G$). A naive approach to this problem is to iterate over all safe schedulers $\sigma_1, \sigma_2, \sigma_3, \ldots$ of $\mathcal{M}$ and pursue in the $j$-th iteration as follows. Deploy the (safe) scheduler $\sigma_j$ on the robot. By letting the robot safely explore the environment (according to $\sigma_j$), one obtains the expected costs $c_j$, say, of reaching $G$ (under $\sigma_j$). By doing so for all safe schedulers, one obtains the minimum cost. After checking all safe schedulers, we have obtained a safe minimal one whenever some $c_n$ is below the threshold $\kappa$. The solution to the synthesis problem is then the scheduler $\sigma_n$ for which $c_n$ is minimal. Otherwise, we can conclude that the synthesis problem has no solution. Note that while deploying



MDP $\mathcal{M}$, minimally initialized cost function $\rho$, safety specification $\varphi$, performance specification $\psi$
1. *Compute safe permissive scheduler* $\theta \in PSched^{\mathcal{M}}$; exclude all previously computed schedulers
Induced MDP $\mathcal{M}^{\theta}$
2. *Obtain locally cost-optimal scheduler* $\sigma \in \theta$ and refine cost function $\rho$ via reinforcement learning
Cost function $\rho$
3. Compute scheduler $\sigma^l \in Sched^{\mathcal{M}}$ on the original MDP $\mathcal{M}$ inducing a *lower bound on the expected cost*
Scheduler $\sigma$
Scheduler $\sigma^l$
4. *Check* if $\sigma \models \psi$ or if $\sigma$ is optimal
no
yes
Return $\sigma$

**Fig. 4.** Overview of safety-constrained reinforcement learning

the safe schedulers, the robot explores more and more possible trajectories, thus becoming more knowledgeable about the (a-priori) unknown cost structure of the MDP.

Although this approach is evidently sound and complete, the number of deployments is excessive. Our approach avoids this by:

1. Testing permissive (i. e. *sets* of) schedulers rather than one scheduler at a time. This is done by employing reinforcement learning.
2. Using that the expected costs $c^*$ under $\sigma^*$ cannot be smaller than the minimal expected cost $c$ in the MDP $\mathcal{M}$ (possibly achieved by some unsafe scheduler). This allows for deciding minimality of scheduler $\sigma_j$ by checking $c_j = c$, possibly avoiding exploration of any further schedulers.
3. Preventing the deployment of safe scheduler $\sigma_j$ whenever the minimal expected cost $c_i$ of all schedulers checked so far $(i < j)$ is smaller than the expected cost under $\sigma_j$.

Let us now briefly explain our approach to synthesize a safe and optimal scheduler; further details are given in the rest of this section. Figure 4 surveys the approach. We initialize the cost function of the MDP by setting the cost of transition $(s, a)$ to its lower bound $l_{(s,a)}$. The synthesis of a safe and optimal scheduler is done by iteratively considering permissive schedulers $\theta_1, \theta_2, \theta_3, \ldots$ according to which the MDP $\mathcal{M}$ is explored. This yields a scheduler $\sigma$ whose expected cost is minimal among the schedulers deployed so far. This search is finished whenever either the expected costs under $\sigma$ is below $\kappa$, $\sigma$ is globally optimal, or no further permissive schedulers can be found. In the $j$-th iteration, the following four steps are carried out:

1. Determine the $j$-th safe permissive scheduler $\theta_j$ (if it exists) such that $\theta_j \models \varphi$. All previously considered schedulers are excluded from $\theta$. This ensures that $\theta_j$ is a fresh permissive scheduler; see Section 4.1 for details.
2. Check all compliant schedulers of $\theta_j$ by reinforcement learning. This yields scheduler $\sigma^j \in \theta_j$ that minimizes the expected cost of reaching $G$. By Remark 2 on Page 8, $\sigma_j$ induces a (randomized) scheduler $\sigma$ for $\mathcal{M}$. The scheduler $\sigma$ is safe w. r. t. $\varphi$ and cost-minimal among all compliant schedulers to $\theta$. During the learning process, the cost function $\rho$ is *refined* with the actual costs for the (newly) explored actions. See Section 4.2 for details.
3. Using the refined cost function, a scheduler $\sigma_l$ inducing minimal expected cost $c_l$ is computed for the original MDP $\mathcal{M}$ (neglecting being safe or not). As this is computed using lower bounds on local costs and potentially using an unsafe scheduler, the expected cost forms a lower bound on the cost obtained using full knowledge of the cost function and only safe schedulers.
4. After learning the scheduler $\sigma$, we check whether $\mathrm{EC}^{\mathcal{M}^\sigma}(\lozenge G) \leq \kappa$. Moreover, if the expected cost equals the lower bound computed in Step 3, i. e., $\mathrm{EC}^{\mathcal{M}^\sigma}(\lozenge G) = c_l$, the scheduler $\sigma$ is globally optimal (and safe).

Furthermore, the best scheduler found so far induces an *upper bound* on the performance as it is optimal for the already learned parts of the MDP. After computing a new candidate (permissive) scheduler, we can re-compute its performance using the lower bounds on actions on the original MDP. If it does not (potentially) admit a better performance, it does not need to be deployed at all.

Note that in the worst case, we actually enumerate all possible safe schedulers, i. e. the maximal permissive scheduler. However, the iterative nature of the procedure together with the optimizations allows for earlier termination as soon as the optimum is reached or the gap between the lower and upper bounds for the minimal expected cost is sufficiently small.

**Theorem 1.** *Safety-constrained reinforcement learning is sound and complete.*

The method is sound and complete because finally we iterate over all safe permissive schedulers and thereby over all possible safe schedulers.

## 4.1 Computing permissive schedulers

In the following, we discuss how to compute a safe deterministic permissive scheduler that induces a safe sub-MDP such as illustrated in Example 2. Moreover, we indicate how a safe permissive scheduler can be computed in general (for randomized schedulers). Recall that according to our setting we are given an MDP $\mathcal{M} = (S, s_I, Act, \mathcal{P})$ and a safety specification $\varphi = \mathbb{P}_{\leq \lambda}(\lozenge T)$ for $T \subseteq S$.

The computation will be performed by means of an SMT encoding. This is similar to the mixed linear integer programming (MILP) approach used in [11]. The intuition is that a satisfying assignment for the encoding induces a *safe* permissive scheduler according to Definition 6. We use the following variables.

$y_{s,a} \in \mathbb{B} = \{\texttt{true}, \texttt{false}\}$ for each state $s \in S$ and each action $a \in Act(s)$ is assigned `true` iff action $a$ is allowed to be taken in state $s$ by the permissive scheduler. These variables form the permissive scheduler.

$p_s \in [0,1] \subseteq \mathbb{R}$ for each state $s \in S$ captures the *maximal* probability to reach the set of target states $T \subseteq S$ under each possible scheduler that is compliant to the permissive scheduler.

The SMT encoding reads as follows.

$$p_{s_I} \leq \lambda \tag{1}$$

$$\forall s \in S. \quad \bigvee_{a \in Act(s)} y_{s,a} \tag{2}$$

$$\forall s \in T. \quad p_s = 1 \tag{3}$$

$$\forall s \in S. \forall a \in Act(s). \quad y_{s,a} \to p_s \geq \sum_{s' \in S} \mathcal{P}(s,a)(s') \cdot p_{s'} \tag{4}$$

First, Constraint 1 ensures that the maximal probability at the initial state $s_I$ achieved by any scheduler that can be constructed according to the valuation of the $y_{s,a}$-variables does not exceed the given safety threshold $\lambda$. Due to Constraint 2, at least one action $a \in Act(s)$ is chosen by the permissive scheduler for every state $s \in S$ as at least one $y_{s,a}$-variable needs to be assigned `true`. The probability of target states is set to 1 by Constraint 3. Finally, Constraint 4 puts (multiple) lower bounds on each state's probability: For all $s \in S$ and $a \in Act$ with $y_{s,a} = \texttt{true}$, the probability to reach the target states is computed according to this particular choice and set as a lower bound. Therefore, only combinations of $y_{s,a}$-variables that induce safe schedulers can be assigned `true`.

**Theorem 2.** *The SMT encoding given by Constraints 1–4 is sound and complete.*

*Proof sketch. Soundness* refers to the fact that each satisfying assignment for the encoding induces a safe deterministic permissive scheduler for MDP $\mathcal{M}$ and safety specification $\varphi$. This is shown by constructing a permissive scheduler according to an arbitrary assignment of $y_{s,a}$-variables. Applying the other (satisfied) constraints ensures that this scheduler is safe. *Completeness* means that for each safe deterministic permissive scheduler, a corresponding satisfying assignment of the constraints exists. This is done by assuming a safe deterministic permissive scheduler and constructing a corresponding assignment. Checking all the constraints ensures that this assignment is satisfying.

Now, consider a deterministic scheduler $\sigma \in Sched^{\mathcal{M}}$ which we want to *explicitly exclude* from the computation. It needs to be ensured that for a satisfying assignment at least for one state the corresponding $y_{s,\sigma(s)}$ variable is assigned `false` in order to at least make one different decision. This can be achieved by adding the disjunction $\bigvee_{s \in S} \neg y_{s,\sigma(s)}$ to the encoding.

Using an SMT solver like `Z3`, this encoding does not ensure a certain grade of permissiveness, i. e., that as many $y_{s,a}$-variables as possible are assigned `true`. While this is a typical setting for MAX-SMT [19], in the current stable version of `Z3` this feature is not available yet. Certain schedulers inducing high probabilities or desired behavior can be included using the *assumptions* of the SMT solver. An alternative would be to use an MILP encoding like, e. g., in [11, 20], and optimize towards a maximal number of available nondeterministic choices. However, in our setting it is crucial to ensure *incrementality* in the sense that if certain changes to the constraints are necessary this does not trigger a complete restart of the solving process.

Finally, there might be safe *randomized schedulers* that induce better optimal costs than all deterministic schedulers [7, 8]. To compute *randomized permissive schedulers*, the difficulty is that there are arbitrarily (or even infinitely) many probability distributions over actions. A reasonable approach is to bound the number of possible distributions by a fixed number $n$ and introduce for each state $s$, distribution $\mu_i$, and action $a$ a real-valued variable $y_{s,\mu_i,a}$ for $1 \leq i \leq n$. Constraint 2 is modified such that for all states and actions the $y_{s,\mu_i,a}$-variables sum up to one and the probability computation in Constraint 4 has to take probabilities over actions into account. Note that the MILP approach from [11] cannot be adapted to randomized schedulers as non-linear constraints are involved.

### 4.2 Learning

In the learning phase, the main goal of this learning phase is the *exploration* of this MDP, as we thereby learn the cost function. In a more practical setting, we should balance this with *exploitation*, i. e., performing close to optimal—within the bounds of the permissive scheduler—during the learning. The algorithm we use for the reinforcement learning is *Q-learning* [10]. To favor the exploration, we initialize the learning with *overly-optimistic* expected rewards. Thereby, we explore large portions of the MDP while favoring promising regions of the MDP.

Proper balancing of exploration vs. exploitation depends on the exact scenario [21]. Here, the balance is heavily affected by the construction of permissive schedulers. For instance, if we try to find permissive schedulers which do not exclude the currently best known scheduler, then the exploitation during the learning phase might be higher, while we might severely restrict the exploration.

## 5 Experiments

We implemented a prototype of the aforementioned synthesis loop in `C++` and conducted experiments using case studies motivated by robotic motion planning. Our prototype uses the SMT-based permissive scheduler computation described in Section 4.1 and seeks a *locally* maximal permissive scheduler by successively adding as many actions as possible.

Every MDP considered in the case studies has a set of bad states (that may only be reached with a certain probability) and a set of goal states that the system tries to reach. All case studies feature a relatively large number of nondeterministic choices in each state and a high amount of probabilistic branching to illustrate the applicability of our method to practically relevant models with a high number of schedulers that achieve various performances.

*Janitor.* This benchmark is loosely based on the grid world robot from [5]. It features a grid world with a controllable robot. In each step, the robot either changes its direction (while remaining on the same tile) or moves forward in the currently selected direction. Doing so, the robot consumes fuel depending on the surface it currently occupies. The goal is to minimize the fuel consumption for reaching the opposite corner of the grid world while simultaneously avoiding collision with a janitor that moves randomly across the board.

*Following a line fragment.* We consider a (discretized) variant of a machine that is bound to follow a straight line, e. g. a sawmill. In each step, there is a certain probability to deviate from the line depending on the speed the machine currently operates at. That is, higher speeds come at the price of an increased probability to deviate from the center. Given a fixed tolerable distance $d$, the system must avoid to reach states in which the distance from the center exceeds $d$. Also, the required time to complete the task or the required energy are to be minimized, both of which depend on the currently selected speed mode of the system.

*Communicating explorer.* Finally, we use the model of a semi-autonomous explorer as described in e. g. [22]. Moving through a grid-like environment, the system communicates with its controller via two lossy channels for which the probability of a message loss depends on the location of the explorer. The explorer can choose between performing a limited number of attempts to communicate or moving in any direction in each time step. Similarly to the janitor case study, the system tries to reach the opposite corner of the grid while avoiding states in which the explorer moved too far without any (successful) intermediate communication. For this model, the cost to be optimized is the energy consumption of the electronic circuit, which induces cost for movement, e. g. by utilizing sensors, and (significantly higher) cost for utilizing the communcation channels.

*Benchmark results.* Table 1 summarizes the results we obtained using our prototype on a MacBook Pro with an 2.67GHz Intel Core i5 processor and a memory limit of 2GB. As SMT-backend, we used Z3 [23] in version 4.4.0. For several instances of each case study, we list the number of states, transitions, and probabilistic branches (i. e., the size of the support set of the distributions). Furthermore, we give the bound $\lambda$ used in the safety property and the optimal performance over all safe schedulers. The following columns provide information about the progress of the synthesis procedure over several selected iterations. The first of these columns ($i$) shows the number of iterations performed thus far, i. e., the number of permissive schedulers on which we applied learning. For

| Benchmark | | states | trans. | branch. | $\lambda$ | **opt.** | $i$ | $t$ | lower | **upper** |
|---|---|---|---|---|---|---|---|---|---|---|
| Janitor | 5,5 | 625 | 1125 | 3545 | 0.1 | **88.6** | 1 | 813 | 84 | **88.6** |
| | | | | | | | 2 | 2578 | 84 | **88.6** |
| FolLine | 30,15 | 455 | 1265 | 3693 | 0.01 | **716.0** | 1 | 41 | 715.4 | **717.1** |
| | | | | | | | 3 | 85 | 715.62 | **716.83** |
| | | | | | | | 13 | 306 | 715.9 | **716.5** |
| | 40,15 | 625 | 1775 | 5223 | 0.12 | **966.0** | 1 | 304 | 964.8 | **968.2** |
| | | | | | | | 3 | 420 | 965.4 | **967.2** |
| | | | | | | | 8 | 738 | 965.6 | **966.7** |
| ComExp | 6,6,6 | 823 | 2603 | 3726 | 0.08 | **54.5** | 1 | 5 | 0.3 | **113.3** |
| | | | | | | | 2 | 26 | 0.3 | **74.9** |
| | | | | | | | 3 | 105 | 0.3 | **57.3** |
| | 8,8,6 | 1495 | 4859 | 6953 | 0.12 | **72.9** | 1 | 15 | 0.42 | **163.1** |
| | | | | | | | 2 | 80 | 0.42 | **122.0** |
| | | | | | | | 3 | 112 | 0.42 | **90.1** |
| | | | | | | | 7 | 1319 | 0.42 | **78.2** |

**Table 1.** Benchmark results

iteration $i$, we give the cumulative time $t$ in seconds required for the computation of the permissive scheduler as well as the current lower and upper bound on the cost (w. r. t. the performance measure). The computation time for simulating deployment and reinforcement learning are negligible.

*Discussion of the results.* For the Janitor and FolLine case studies, we observe that the investment of computing a locally maximal permissive scheduler pays off, meaning that we get very tight lower and upper bounds already after the first deployment. This investment comes at the cost of a higher computational effort (per iteration). This could be reduced by more elaborate heuristics which limit our search for (local) maximal permissiveness.

For the communicating explorer, the situation is more difficult. Since a scheduler that does not communicate at all has very low expected costs, a loose lower bound has been obtained. This bound could be severely improved upon by obtaining tighter lower bounds via multi-objective model checking.

*Lessons learned.* Based on our experiments, we learned that quantifying permissiveness via the enabled number of actions yields counterintuitive results. Observe that for *unreachable states* literally *all actions* can be included in the scheduler without affecting the satisfaction of a property. This leads to the effect that—in order to achieve a high permissiveness—it is best to have only few reachable states and allow all actions for unreachable states. This effect is unmentioned by prior work in [11]. It thus follows that quantifying permissiveness should only consider actually reachable states. This observation is related to the general problem of forcing a solver to ensure reachability of certain states, which would also be beneficial for ensuring the reachability of, e. g., goal states. However, any guidance towards this proved to drastically decrease the solver performance.

## 6 Conclusion and future work

We presented the—to the best of our knowledge—first approach on iteratively computing safe and optimal strategies in a setting subject to random choices, unknown cost, and safety hazards. Our method was shown to work on practical benchmarks involving a high degree of nondeterminism. Future work will concern improving the scalability by employing multi-objective model checking in order to prove optimality at earlier iterations of the process. Moreover, extensions to stochastic 2-player games for modeling adversarial environment behavior or the investigation of unknown probability distributions seem very interesting.

*Acknowledgements.* We want to thank Benjamin Lucien Kaminski for the valuable discussion on the worst case size of conflicting sets.

## References

1. Kwiatkowska, M., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: Proc. of CAV. Volume 6806 of LNCS, Springer (2011) 585–591
2. Ciesinski, F., Baier, C.: Liquor: A tool for qualitative and quantitative linear time analysis of reactive systems. In: Proc. of QEST. (2006) 131–132
3. Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Performance Evaluation **68**(2) (2011) 90–104
4. Penna, G.D., Intrigila, B., Melatti, I., Tronci, E., Zilli, M.V.: Finite horizon analysis of Markov chains with the Murphi verifier. Software Tools for Technology Transfer **8**(4-5) (2006) 397–409
5. Kwiatkowska, M., Norman, G., Parker, D.: The PRISM benchmark suite. In: Proc. of QEST, IEEE CS (2012) 203–204
6. Forejt, V., Kwiatkowska, M.Z., Parker, D.: Pareto curves for probabilistic model checking. In: Proc. of ATVA. Volume 7561 of LNCS, Springer (2012) 317–332
7. Etessami, K., Kwiatkowska, M.Z., Vardi, M.Y., Yannakakis, M.: Multi-objective model checking of Markov decision processes. Logical Methods in Computer Science **4**(4) (2008)
8. Baier, C., Dubslaff, C., Klüppelholz, S.: Trade-off analysis meets probabilistic model checking. In: Proc. CSL-LICS, ACM (2014) 1:1–1:10
9. Sutton, R., Barto, A.: Reinforcement Learning – An Introduction. MIT Press (1998)
10. Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: Proc. of ICML, Morgan Kaufmann (1994) 157–163

11. Dräger, K., Forejt, V., Kwiatkowska, M.Z., Parker, D., Ujma, M.: Permissive controller synthesis for probabilistic systems. In: Proc. of TACAS. Volume 8413 of LNCS, Springer (2014) 531–546
12. Wen, M., Ehlers, R., Topcu, U.: Correct-by-synthesis reinforcement learning with temporal logic constraints. CoRR (2015)
13. Moldovan, T.M., Abbeel, P.: Safe exploration in Markov decision processes. In: Proc. of ICML, icml.cc / Omnipress (2012)
14. Fu, J., Topcu, U.: Probably approximately correct MDP learning and control with temporal logic constraints. In: Proc. of RSS. (2014)
15. Akametalu, A., Fisac, J., Gillula, J., Kaynama, S., Zeilinger, M., Tomlin, C.: Reachability-based safe learning with Gaussian processes. In: Proc. of CDC. (2014) 1424–1431
16. Pecka, M., Svoboda, T.: Safe exploration techniques for reinforcement learning–an overview. In: Proc. of MESAS. Volume 8906 of LNCS, Springer (2014) 357
17. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
18. Sokolova, A., de Vink, E.P.: Probabilistic automata: System types, parallel composition and comparison. In: Validation of Stochastic Systems. Volume 2925 of LNCS. Springer (2004) 1–43
19. Bjørner, N., Phan, A.: $\nu$Z - maximal satisfaction with Z3. In: Proc. of SCSS. Volume 30 of EPiC Series, EasyChair (2014) 1–9
20. Wimmer, R., Jansen, N., Ábrahám, E., Katoen, J.P., Becker, B.: Minimal counterexamples for linear-time probabilistic verification. Theoretical Computer Science **549** (2014) 61–100
21. Brafman, R.I., Tennenholtz, M.: R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learning Research **3** (2002) 213–231
22. Stückler, J., Schwarz, M., Schadler, M., Topalidou-Kyniazopoulou, A., Behnke, S.: Nimbro explorer: Semiautonomous exploration and mobile manipulation in rough terrain. Journal of Field Robotics (2015) to appear.
23. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: Proc. of TACAS. Volume 4963 of LNCS, Springer (2008) 337–340