arXiv:1807.06096v1 [cs.AI] 16 Jul 2018

Shielded Decision-Making in MDPs

Nils Jansen
Radboud University, Nijmegen
The Netherlands

Bettina Könighofer
TU Graz
Austria

Sebastian Junges
RWTH Aachen University
Germany

Roderick Bloem
TU Graz
Austria

Abstract

A prominent problem in artificial intelligence and machine learning is the safe exploration of an environment. In particular, reinforcement learning is a well-known technique to determine optimal policies for complicated dynamic systems, but suffers from the fact that such policies may induce harmful behavior. We present the concept of a shield that forces decision-making to provably adhere to safety requirements with high probability. Our method exploits the inherent uncertainties in scenarios given by Markov decision processes. We present a method to compute probabilities of decision making regarding temporal logic constraints. We use that information to realize a shield that—when applied to a reinforcement learning algorithm—ensures (near-)optimal behavior both for the safety constraints and for the actual learning objective. In our experiments, we show on the arcade game PAC-MAN that the learning efficiency increases as the learning needs orders of magnitude fewer episodes. We show tradeoffs between sufficient progress in exploration of the environment and ensuring strict safety.

1 Introduction

In recent years, artificial intelligence (AI), and in particular *machine learning*, evolved from areas like game-playing or language-translation to critical domains such as health, energy, defense, or transportation. As a result, a major challenge is the *safety* of decision-making for systems employing AI [41, 15, 37, 35]. The area of *safe exploration* aims at restricting decision-making to adhere to safety requirements during the exploration of an environment [31, 2]. Such restrictions may cause *insufficient progress* in following the original objective of the decision-maker or even *deadlocks*. Thus, there is a trade-off between safety and progress which needs to be properly addressed. Take for instance a self-driving car. The main objective for the underlying AI is to follow a road, while in certain critical events the car has to brake in order to avoid an accident. However, making emergency brakes too often is not desirable in terms of many performance measures.

Reinforcement learning (RL) [42] lets an agent *explore* its environment by sequential decision-making. The objective is to (approximatively) optimize the expected reward for an agent in the underlying Markov decision process (MDPs) [33]. During the exploration of the MDP, the current policy may be unsafe in the sense that it harms the agent or the environment. This shortcoming restricts the application of RL mainly to application areas where safety is not a concern and has triggered the particular direction of safe exploration for RL, in short *safe RL* [18, 31].

We introduce the concept of *runtime enforcement* for MDPs. We assess safety by means of (probabilistic) *temporal logic constraints* [4], which restrict, for instance, the probability to reach a set of critical states in the MDP. Such constraints together with the original RL objective, optimizing

Preprint. Work in progress.

the expected reward, cause the aforementioned tradeoffs between safety and progress. We employ so-called *shields* [8] that prevent decisions during runtime that cause the violation of the temporal logic constraints. For an RL algorithm augmented with such shield, any intermediate or final policy provably adheres to such constraints. We identify the following requirements:

Guaranteed Safety: If sequential decision-making for an MDP is shielded, the resulting policy should satisfy the probabilistic temporal logic constraints. A user may provide a *safety-level* in form of an upper bound on the acceptable probability for decisions to be unsafe.

Adaptivity: In certain situations the shield may be too restrictive to allow for sufficient progress. In that case, it needs to dynamically adapt to allow more, potentially less safe, decisions. The predefined safety-level may bound such adaptivity. In case safety and progress prerequisites contradict each other, the shield will potentially prevent all possible decisions. If such *deadlocks* are not avoidable, it is not possible to synthesize a feasible shield.

In addition to these hard requirements, we establish the following desired properties of a shield:

Minimal Invasiveness: The shield should restrict potential decisions as little as possible while still ensuring safe decision-making.

Independence: While we focus on shielding RL algorithms, the concept of a shield should be amenable for any kind of decision-making algorithm for MDPs.

Related Work. Most approaches to safe RL [18, 31] rely on reward engineering and changing the learning objective. In contrast to ensuring temporal logic constraints, reward engineering designs or "tweaks" the reward functions attached to state observations such that a learning agent behaves in a desired, potentially safe, manner. As rewards are often specialized for particular environments, reward engineering runs the risk of triggering negative side effects or hiding potential bugs [38].

First approaches actually incorporating formal specifications tackle this problem with pre-computations making strong assumptions on the available information about the environment [44, 39, 21, 17, 28, 29], by employing PAC guarantees [16], or by an intermediate "correction" of policies [30]. Safe model-based RL for continuous state spaces with formal stability guarantees is considered in [7]. Most related is [1], which also introduces the concept of a shield for RL. In contrast to our work, however, the underlying stochastic behavior of the MDP is not exploited which circumvents the learning agent from taking any risks, preventing progress in sufficiently exploring the environment.

Our Contributions and Structure of the Paper. We present a novel method to shield decision-making for MDPs regarding temporal logic constraints. We employ a *separation of concerns*: For a large MDP setting with an potentially unknown reward function, we compute a smaller MDP that is only relevant for safety assessments. This restriction enables the usage of formal methods to compute a shield. Reinforcement learning for the full scenario under this shield is then guaranteed to be safe, while we never construct the full (large) MDP.

First, we state the formal problem and provide necessary background in Sect. 2. To construct an MDP in the first place, we build a behavior model for the environment, for example using model-based RL [13], in a training environment using data augmentation techniques. We plug this behavior model into a concrete scenario and obtain an MDP. In Sect. 3 we describe in detail how to actually construct a shield and provide optimizations towards a computationally tractable implemenation. Then, we demonstrate several concepts on how to maintain sufficient progress in exploring an environment while shielding decision-making. We show the correctness of these constructions. In Sect. 4, we demonstrate our implementation and experiments based on the arcade game PAC-MAN. We show that RL for PAC-MAN is safe and performs superior if it is augmented by a shield.

2 Problem statement

Background

A **probability distribution** over a finite or countably infinite set X is a function $\mu\colon X \to [0,1] \subseteq \mathbb{R}$ with $\sum_{x\in X} \mu(x) = 1$. The set of all distributions on X is denoted by $\mathit{Distr}(X)$, and the support of $\mu \in \mathit{Distr}(X)$ is the set $\mathit{supp}(\mu) = \{x \in X \mid \mu(x) > 0\}$.

A **Markov decision process (MDP)** $\mathcal{M} = (S, Act, \mathcal{P}, r)$ has a finite set S of *states*, a finite set Act of *actions*, a (partial) *probabilistic transition function* $\mathcal{P}\colon S \times Act \to Distr(S)$, and an *immediate reward function* $r\colon S \times Act \to \mathbb{R}_{\geq 0}$. The set $Act(s)$ denotes the actions at $s \in S$, $Act(s) = \{\alpha \in Act \mid \exists \mu \in Distr(S).\, \mu = \mathcal{P}(s, \alpha)\}$. We disallow deadlock states: $|Act(s)| \geq 1$ for all $s \in S$.

A **policy** is a function $\sigma\colon S^* \to Distr(Act)$ with $supp(\sigma(s_1 \dots s_n)) \subseteq Act(s_n)$, where S^* denotes a finite sequence of states. While for many specifications, it suffices to consider *stationary*, *deterministic* policies $\sigma\colon S \to Act$ [33], in the presence of multiple—possibly conflicting—specifications, more general policies (with randomization and finite memory) are necessary [10].

In formal methods, specifications are mostly given in variants of **temporal logic constraints** such as linear temporal logic (LTL) [32] or computation-tree logic (CTL) [11]. **Model checking** is a fully automatic formal verification technique [12, 4]. Based on a system model and a description of the desired properties in LTL or CTL, model checkers automatically assess whether the model satisfies the properties. Due to its rigor, the reliability of the results *only* depends on the quality of the model.

We consider a variant of temporal logic called **probabilistic computation tree logic** (PCTL) [19]. A specification in PCTL is for instance of the form $\mathbb{P}_{\geq \lambda}(\Diamond T)$, which is satisfied for an MDP $\mathcal{M}$, if the probability to "eventually" reach a set of target states $T \subseteq S$ is at least $\lambda \in [0, 1]$, when considering all possible policies. Probabilistic model checking [23, 26, 3] employs methods based on, e.g., value iteration or linear programming to verify such specifications for MDPs. Tool support is readily available in `PRISM` [27], `Storm` [14], or `Modest` [20]. Specifically, for an MDP $\mathcal{M}$ and a PCTL specification $\mathbb{P}_{\geq \lambda}(\varphi)$, we compute $\eta^{\max}_{\varphi,\mathcal{M}}\colon S \to [0, 1]$ or $\eta^{\min}_{\varphi,\mathcal{M}}\colon S \to [0, 1]$, where $\eta^{\min}_{\varphi,\mathcal{M}}(s)$ (or $\eta^{\max}_{\varphi,\mathcal{M}}(s)$) give which give the minimal (or maximal) probability over all possible policies to satisfy φ for all states of the MDP. Similar problems are considered in probabilistic planning [40, 24].

Setting

We consider problems in a broad multi-agent setting. An arena is a finite directed graph $G = (V, E, d)$ with a finite set V of nodes, a set of edges $E \subseteq V \times V$, and distances $d\colon E \to \mathbb{N}_{>0}$. The **position** of an agent encodes the current node v, a target node v', and the distance to the goal, i. e., the distance of the edge minus the steps already taken along the edge that the agent is moving along. Formally, a position $(v, v', n) \in Pos = V \times V \times \{0, \dots, \max_{e \in E} d(e)\}$ states that the agent is approaching v' from v and arrives there in $n \in \mathbb{N}$ steps. When $n = 0$, the agent **decides** on a new edge $(v, v') \in E$ where v' is the new goal, and the agent arrives there in $d(v, v')$ steps.

Specifically, we consider partially-controlled multi-agent systems [9], with one controllable agent which we call the **avatar**. All other agents are uncontrollable, and we call them **adversaries**. To edges we also associate a (partial) **token** function $\circ\colon E \to \{0, 1\}$, indicating the (scenario dependent) status of some edge. Tokens can be activated (structurally or randomly), and have an associated **reward** that is earned as long as the token is present, or upon visiting an edge.

As an example, take a factory floor plan with several corridors. The nodes describe crossings, the edges the corridors with machines, and the distances the length of the corridors. The adversaries are (possibly autonomous) transporters moving parts within the factory. The avatar models a service unit moving around and inspecting machines where an issue has been raised (as indicated by a token). All agents follow the corridors and take another corridor upon reaching a crossing. Several notions of cost can be induced by these tokens, either as long as they are present (indicating the costs of a broken machine) or for removing the tokens (indicating costs for inspecting the machine).

Problem

We assume, the avatar selects edges based on an unknown decision procedure. Our aim is to provide a shield for the decision procedure of the avatar to avoid unsafe behavior. In particular, given a specification in PCTL, we aim to prevent decisions necessarily leading to behavior that with high probability violates these specifications. The probability threshold λ should be relative to the current position – some states are intrinsically more dangerous, and a shield that disables all potential decisions would lead to deadlocks in the system. Thus a notion of progress is essential. Progress may even be mandatory to ensure safety, i. e., stagnation may lead to violation of the specification.

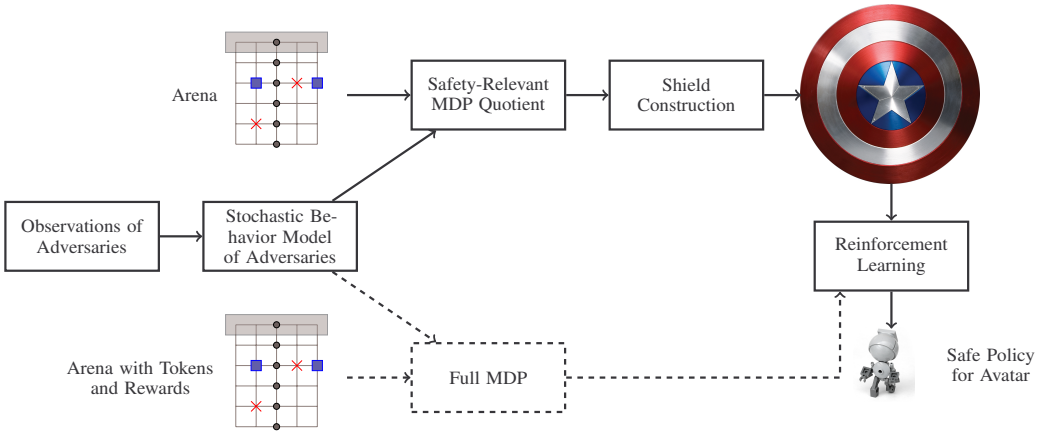


Figure 1: Workflow of the Shield Construction

3 Constructing Shields for MDPs

We outline the workflow of our approach in Figure 1. The starting point is the setting described in the previous section. First, based on observations in arbitrary arenas, we construct a general (stochastic) **behavior model** for each adversary. Combining these models with a concrete arena yields an MDP. At this point, we ignore the token function (and necessarily the potentially unknown reward function), so the MDP may be seen as a quotient of the real system within which we only assess safe behavior. We therefore call the MDP the **safety-relevant quotient**. The real scenario incorporates the **token function**. Rewards may be known or only be observed during learning. In any case, theoretically, the underlying **full MDP** constitutes an **exponential blowup of the safety-relevant quotient**, rendering probabilistic model checking or planning infeasible for realistic scenarios. First, using the safety-relevant MDP, we construct a **shield** using probabilistic model checking. RL now aims to maximize the reward according to the original scenario. As we augment the RL with the pre-computed shield, all decisions are **guaranteed to form an overall policy that adheres to safety requirements**.

Separation of Concerns: Probabilistic Model Checking and Reinforcement Learning. We have to separate the problem to make use of the individual strengths of the two disciplines of formal methods and machine learning. By learning adversary behavior for general scenarios, we enable to construct a compact MDP model in which we can assess safety. Only by neglecting the token and the reward structure, we are able to employ model checking (or probabilistic planning) in a feasible way. Then, with a shield for unsafe behavior precomputed, we determine optimal behavior using reinforcement learning for a large scenario with an a priori unknown performance criterion.

We now detail the individual technical steps to realize our proposed methods.

3.1 Learning the Adversary Model

We learn an adversary by observing its behavior in several arenas, until we gain a sufficient confidence that by obtaining more training data the behavior would not change significantly. An upper bound on the necessary data may be obtained using Hoeffding's inequality [46]. To reduce the size of the training set, we devise a data augmentation technique using **domain knowledge** of the arenas [45, 25].

Intuitively, we abstract away from the precise configuration of the arena by partitioning the graph into zones relative to the view-point of the adversary (e. g., close or far, north or south). For an arena $G = (V, E, d)$, **zones relative to a node** $v \in V$ are functions $z_v \colon V \to C$ where C is a finite set of **colors**. For nodes $x, y \in V$, with $z_v(x) = z_v(y)$, the assumption is that the adversary in v behaves similar regardless wether the avatar is in x or y. From our observations, we extract a **histogram** $E \times C \to \mathbb{N}$ describing how often the adversary takes an edge, depending on the color. Then, we translate these likelihoods based on our observations into distributions over the possible edges. The **adversary behavior** becomes a function $B \colon V \times C \to \mathit{Distr}(E)$. While we employ

a simple normalization of likelihoods, alternatively one may also utilize, e. g., a softmax function which is adjustable to favor more or less likely decisions [42].

3.2 Constructing the Shield

Safety-Relevant MDP. We describe how to construct the MDP that is the basis for our safety analysis. For an arena (V, E, d), we have an avatar and m adversaries, i. e., agents $0, \ldots, m$. For each adversary, we have corresponding behaviors $B_1, \ldots, B_m$. We build the safety-relevant MDP $\mathcal{M}$ as follows. The **states** $S = Pos^{m+1} \times \{0, \ldots, m\}$ encode the positions for all agents. The **actions** $Act = \{\alpha_0\} \cup \{\alpha_e \mid e \in E\}$ determine the movements. If agent i moves next and its position is (v, v', n) with $n > 0$, there is one unique action α_0 at the corresponding state. That action has one successor state where n is decremented by one and i incremented modulo m, i. e., agent $i + 1$ will move next. Note that if $n > 0$, there is no **decision** to be made. If $n = 0$, the agent needs to decide which edge (v', v'') to select. Intuitively, this selection means that the position of agent i is set to $(v'', w(v', v''))$ and again i is incremented modulo m.

Edge selection has different types: If $i > 0$ (an adversary moves next), there is a unique action α_0 where the successor state is randomly determined according to the behavior B_i for the current position of the adversary and the avatar. If $n = 0$ and $i = 0$ (the avatar moves next), there is an action α_e reflecting every outgoing edge $e \in E$. Observe that the only states in which the avatar has to make choices are of the form $s_d = (v, v', 0, \ldots, 0)$. We call these states s_d the **decision states**, from which the underlying decision procedure selects action α_e leading to a state $s_e = (v', v'', n, \ldots, 1)$.

Shield Construction using Probabilistic Model Checking. For an arena (V, E, d) and the corresponding safety-relevant MDP $\mathcal{M}$, we have a PCTL specification $\mathbb{P}_{\geq\lambda}(\varphi)$ with a lower bound λ on the probability to satisfy φ. The task is to evaluate the decision states s_d with respect to the probability of satisfying φ. In particular, we compute $\eta^{\max}_{\varphi,\mathcal{M}}(s_e)$, which is the maximal probability to satisfy φ from state s_e after taking action α_e in state s_d. Using this information, we construct an **action-valuation** for each action α_e at each decision state s_d:

$$val^{\mathcal{M}}_{s_d} \colon Act(s_d) \to [0,1], \text{ with } val^{\mathcal{M}}_{s_d}(\alpha_e) = \eta^{\max}_{\varphi,\mathcal{M}}(s_e)\ .$$

The optimal action-value for s_d is $optval^{\mathcal{M}}_{s_d} = \max_{\alpha' \in Act} val^{\mathcal{M}}_{s_d}(\alpha')$.

We are now ready to define a shield for the safety-relevant MDP $\mathcal{M}$. Specifically, a **δ-shield** for $\delta \in [0,1]$ determines a set of safe actions for each decision state s_d. These actions are δ-optimal for the specification φ. All other actions are "shielded" and cannot be chosen by a decision-maker.

Definition 1 (Shield)**.** For an action-valuation $val^{\mathcal{M}}_{s_d}$ and $\delta \in [0,1]$, a *δ-shield for decision state s_d* is

$$shield^{s_d}_{\delta} \colon (Act(s_d) \to [0,1]) \to 2^{Act(s_d)}$$

with $shield^{s_d}_{\delta}(val^{\mathcal{M}}_{s_d}) = \{\alpha \in Act(s_d) \mid val^{\mathcal{M}}_{s_d}(\alpha) \geq \delta \cdot optval^{\mathcal{M}}_{s_d}\}$.

Thus, the shield is **adaptive** with respect to δ, as a high value for δ yields a stricter shield, a smaller value a less strict shield. A user may enforce a strict threshold, like in the specification $\mathbb{P}_{\geq\lambda}(\varphi)$. If for all s_d it holds that $\delta \cdot optval^{\mathcal{M}}_{s_d} \geq \lambda$, the shield is *realizable* and ensures **guaranteed safety**.

A δ-shield for the whole MDP $\mathcal{M}$ is built by constructing δ-shields for every (decision) state in $\mathcal{M}$. Formally, a **shielded MDP** $\mathcal{M}_{\cup}$ has the transition probability function

$$\mathcal{P}_{\cup}(s,\alpha) = \begin{cases} \mathcal{P}(s,\alpha) & \alpha \in shield^{s}_{\delta}(val^{\mathcal{M}}_{s}) \\ \bot & \text{otherwise.} \end{cases}$$

Lemma 1. *For an MDP $\mathcal{M}$ and a δ-shield, $\mathcal{M}_{\cup}$ is deadlock-free.*

We assume that the original MDP $\mathcal{M}$ is deadlock-free. As we compute the shield relative to the optimal values $optval^{\mathcal{M}}_{s_d}$, at least the optimal action will always be allowed, even if $\delta = 1$.

More importantly, we can compute the function $val^{\mathcal{M}_{\cup}}_{s}$ *for the shielded MDP*.

Theorem 1. *For an MDP $\mathcal{M}$ and a δ-shield, it holds for any state s that $val^{\mathcal{M}}_{s} = val^{\mathcal{M}_{\cup}}_{s}$.*

As the optimal actions are not removed, optimality is preserved in the shielded MDP. In particular, computing a shield for a state is **independent from the application of the shield to other states**.

Constructing the full MDP. In theory, one could build the full MDP for the arena (V, E, d), the token function $\circ\colon E \to \{0, 1\}$, and the associated rewards. Under the assumption that the reward function is known, one would then be able to compute the reward-optimal and safe policy without need for further learning techniques. As there are 2^E token configurations, the state space would blow up exponentially. Thus, any model checking or planning technique would be infeasible even for small applications.

3.3 Faster Construction of Shields

Even with the restriction to the safety-relevant MDP, automatic analysis is challenging for realistic applications. We utilize the following optimizations to render the problem computationally tractable.

Finite Horizon. First, we may restrict the shield computations to finite horizons to increase the scalability of model checking. A policy for the avatar in the shielded MDP is then only guaranteed to be safe for the next steps. Additionally, our learned MDP model is inherently an approximation of the real world (the arena). For infinite horizon properties, errors stemming from this approximation may grow too large. Moreover, the probability of reaching critical states may be 1 for an infinite horizon.

Piecewise Construction. As witnessed by Theorem 1, we can compute the shield for each state independently, enabling multi-threaded computations. Solving many small problems introduces redundancy, as states are considered multiple times. Yet, memory consumption is often the bottleneck for (probabilistic) model checkers. To remove parts of the redundancies, we pre-analyze that in some states all actions allow for a high probability to satisfy the safety specification. The shield will not block any actions for these states—we omit model checking there. In combination with finite horizon properties the shield will then guarantee safety in each state exactly for the specified horizon.

Independent Agents. The explosion of state spaces stems to a large part from the number of agents. Here, an important observation is that we can consider agents independently. For instance, the probability for the avatar to crash with an adversary is stochastically independent from crashing with the other adversaries. Instead of determining the shield for all adversaries at once, we perform computations for each agent individually, and combine them via the inclusion-exclusion principle. Afterwards, the shield is composed from the shields dedicated to individual adversaries.

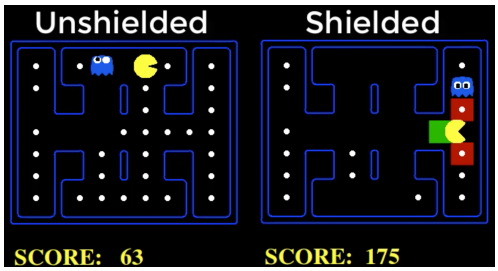
Abstractions. For finite horizon properties, adversaries may be far away—beyond the horizon—without a chance to reach the avatar. In our computation, we neglect all adversary positions that are not relevant at the current state. Such states that in fact have probability 0 to violate the safety specification, are excluded from the state space prior to model checking. For finite-horizon properties, they may not even be part of the MDP we build.

3.4 Shielding versus Progress

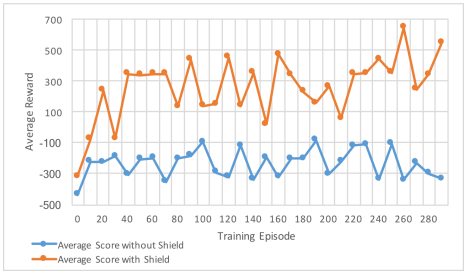
The shield needs to be **minimally invasive** regarding the objective of the decision procedure. We propose three methods to alleviate the invasiveness, all of them assume **domain knowledge** of the rationale behind the decision procedure, for instance in the form of a measure *progress*$\colon S \to [0, 1]$.

Iterative Weakening. During runtime, we may observe that the progress of the avater is decreasing. In that case, we weaken the shield by $\delta - \varepsilon$, allowing additional actions. As soon as progress is made, we can reset δ to its former value. We still guarantee $\delta - \varepsilon$ safety overall. Note that the adaption of $\mathit{shield}^{s}_{\delta}$ to $\mathit{shield}^{s}_{\delta-\varepsilon}$ can be done on the fly, without new computations.

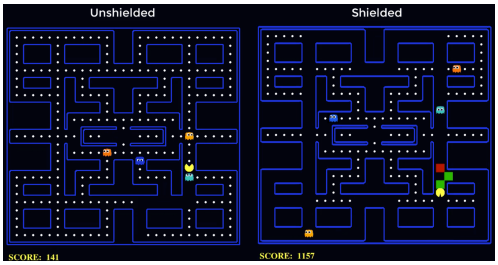
Adapted Specifications. If the goal of the decision maker is known *and* can be captured in temporal logic, we may adapt the original specification accordingly. There are often natural trade-offs between safety and performance. These trade-offs might be resolved via weights, but this process is often undesirable [34] and similar to reward engineering. Instead, optimization of the conditional performance under the assumption of staying sufficiently (nearly-optimal) safe (cf. [5, 43]), avoids side-effects of attaching some weights to the safety specification.



(a) Small PAC-MAN



(b) Resulting Scores for small PAC-MAN



(c) Classic PAC-MAN

(d) Resulting Scores for Classic PAC-MAN

Side Constraints. Side constraints can be deduced and formulated in many fashions. We propose the use of sets of actions, which we refer to as *progress sets*. Consider an MDP $\mathcal{M}$ where some states T are reachable from a state s, and this reachability is desirable. For instance, $T_s = \{s' \in S \mid \mathit{progress}(s') > \mathit{progress}(s)\}$. Assume that in $\mathcal{M}_{\lightning}$, states in T_s are not reachable anymore. Thus, there is at least one action along every path from s to T in the original $\mathcal{M}$ that is blocked by the shield and prevents progress. We put these actions into a progress set. Then, a side constraint to the shield computation states that **from each progress set, one action needs to be allowed**. Computing shields independently is thus possible, but might lead to suboptimal results. In fact, we propose to do a form of regret minimization. We compute the regret for adding actions, and sum over all these actions. The following optimization problem describes the regret minimization, with variables $t_\alpha \in \{0, 1\}$ for each $\alpha \in \mathit{Act}$.

$$\text{minimize} \quad \sum_{s \in S} \sum_{\alpha \in \mathit{Act}(s)} \left(\delta \cdot \mathit{optval}_s^{\mathcal{M}} - \mathit{val}_s^{\mathcal{M}}(a)\right) \cdot t_\alpha \tag{1}$$

$$\text{subject to} \tag{2}$$

$$\forall \text{progress sets } X \quad \sum_{\alpha \in X} t_\alpha \geq 1 \tag{3}$$

The problem may be encoded as a mixed integer linear program [36].

4 Implementation and Experiments

In the previous sections, we discussed a theoretical framework (1) to learn stochastic behavior models for adversaries, (2) to construct a shield for an MDP, (3) to enable the computationally tractable construction of such a shield, and (4) to provide sufficient progress for a shielded learning algorithm.

A Shield for PAC-MAN. To demonstrate our methods, we consider the arcade game PAC-MAN. The task is to eat *food* in a *maze*. Vice versa, *ghosts* want to eat PAC-MAN. PAC-MAN achieves a high *score* if it eats all the food as quickly as possible while minimizing the number of times to get eaten by the ghosts. RL approaches exist [6], but they suffer largely from the fact that during the exploration phase PAC-MAN is eaten by the ghosts many times and thus achieves very poor scores.

We model each instance of the game as an arena, where PAC-MAN is the avatar and the ghosts are adversaries. Tokens represent the food at each position in the maze, such that food is either present or already eaten. Food earns reward, while each step causes a small penalty. Large penalties are imposed when PAC-MAN is eaten by a ghost and the game is restarted. We learn the ghost behavior from

Table 1: Experimental Results

			Total Results - Training				Total Results - Execution			
Size, #Ghosts	**#Model Checking**	**time (s)**	**Score w/o Shield**	**Score with Shield**	**Win Rate w/o Shield**	**Win Rate with Shield**	**Score w/o Shield**	**Score with Shield**	**Win Rate w/o Shield**	**Win Rate with Shield**
6x5,1	780	6	94,12	391	0,46	0,85	-62,8	462,2	0,3	1
11x6,1	5821	61,75	-140,5	613,05	0,22	0,83	325,2	715,7	0,6	0,9
9x7,1	5912	73,5	-236,1	259,24	0,08	0,52	-227,7	547,2	0,1	0,8
17x5,1	5841	61	114,14	798,9	0,31	0,89	76,3	903,9	0,6	1
17x10,3	51732	1227	-220,79	-40,52	0,01	0,07	-412	84,3	0,00	0,1
27x25,4	269426	6647	-129,25	339,89	0,00	0,00	-166,6	397	0,00	0,00

the original PAC-MAN game in small arenas individually for each ghost. Transferring the resulting stochastic behavior to any arena (without tokens) yields the safety-relevant MDP.

For that MDP, we compute a δ-shield (with iterative weakening) via the probabilistic model checker `Storm` [14] for a finite horizon of 10 steps. We use a multi-threaded architecture that lets us construct the shields for very large instances of the example. Approximate Q-learning [42] is the particular RL technique. We compare RL to shielded RL on a number of different instances. The safety constraint for the shield is to **not get eaten** with a high probability. The key comparison criterion is the **score** which is largely affected by the imposed penalties.

Results. Figures 2(a) and 2(c) show screenshots of a series of **videos** we created, which **are uploaded with this submission as supplementary material**. Each video compares how RL performs either shielded or unshielded on a PAC-MAN instance. In the shielded version, at each decision state in the underlying MDP, we indicate the risk of potential decisions by the colors green (low), orange (medium), and red (high). This feature also enables safe game-playing for humans.

We run experiments using an Intel Core i7-4790K CPU with 16 GB of RAM using 4 cores. The piecewise construction of a shield for this large instance takes about 2 hours, while memory is not an issue due to the piecewise shield construction. Figures 2(b) and 2(d) depict the scores obtained during RL. In particular, the curves (blue: unshielded, orange: shielded) show the average scores for every 10 training episodes. One episode lasts until either the game is won (all food eaten) or lost (ghost eats PAC-MAN). Table 1 shows results for several instances in increasing size. We list the number of (multi-threaded) model checking calls and the total time to construct the shield. We distinguish results for the training (300 episodes) and the execution (10 episodes) phases for RL. For both, we list the scores with and without shield, and the *winning rate* which captures the number of episodes PAC-MAN finished without ever being eaten.

For all instances, we see a large difference in scores due to the fact that PAC-MAN is often saved by the shield. The winning rates differ for most benchmarks, favoring shielded RL. For the two largest instances with 3 and 4 ghosts, there are many situations where a shield that plans only 10 steps ahead is not enough to save PAC-MAN from being encircled by the ghosts. Therefore, the wining rate is 0 even in the shielded case. Nevertheless, the shield still safes PAC-MAN in many situations as indicated by the superior scores. Moreover, the shield also helps to learn an optimal policy much faster because less restarts are needed.

In general, learning for an arcade game like PAC-MAN is difficult to perform according to safety constraints if no knowledge about future events if available. Given our relatively loose assumptions about the setting, the shield proved a feasible means to ensure an appropriate measure of safety. Moreover, as in the classic PAC-MAN instance, 100% safety is not possible.

5 Conclusion and Future Work

We developed the concept of shields for MDPs. Utilizing probabilistic model checking, we were able to guarantee probabilistic safety measures during reinforcement learning. We addressed inherent scalability issues and provided means to deal with typical trade-off between safety and performance. Our experiments on a well-known arcade game showed that we improved the state-of-the-art in safe reinforcement learning.

For future work, we will extend shields to richer models such as partially-observable MDPs. Moreover, we will extend the applications to more arcade games and employ deep recurrent neural networks as further means of decision-making [22].

References

[1] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe reinforcement learning via shielding. In *AAAI*. AAAI Press, 2018.

[2] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in ai safety. *CoRR*, abs/1606.06565, 2016.

[3] C. Baier. Probabilistic model checking. In *Dependable Software Systems Engineering*, volume 45, pages 1–23. IOS Press, 2016.

[4] C. Baier and J. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[5] C. Baier, J. Klein, S. Klüppelholz, and S. Wunderlich. Maximizing the conditional expected reward for reaching the goal. In *TACAS (2)*, volume 10206 of *Lecture Notes in Computer Science*, pages 269–285, 2017.

[6] U. Berkeley. Intro to AI – Reinforcement Learning , 2018. http://ai.berkeley.edu/reinforcement.html.

[7] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in Neural Information Processing Systems*, pages 908–919, 2017.

[8] R. Bloem, B. Könighofer, R. Könighofer, and C. Wang. Shield synthesis: - runtime enforcement for reactive systems. In *TACAS*, volume 9035 of *LNCS*, pages 533–548. Springer, 2015.

[9] R. I. Brafman and M. Tennenholtz. On partially controlled multi-agent systems. *Journal of Artificial Intelligence Research*, 4:477–507, 1996.

[10] K. Chatterjee, R. Majumdar, and T. A. Henzinger. Markov decision processes with multiple objectives. In *STACS*, volume 3884 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2006.

[11] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logic of Programs*, pages 52–71. Springer, 1981.

[12] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, 2001.

[13] P. Dayan and Y. Niv. Reinforcement learning: the good, the bad and the ugly. *Current opinion in neurobiology*, 18(2):185–196, 2008.

[14] C. Dehnert, S. Junges, J. Katoen, and M. Volk. A storm is coming: A modern probabilistic model checker. In *CAV (2)*, volume 10427 of *LNCS*, pages 592–600. Springer, 2017.

[15] R. G. Freedman and S. Zilberstein. Safety in ai-hri: Challenges complementing user experience quality. In *AAAI Fall Symposium Series*, 2016.

[16] J. Fu and U. Topcu. Probably approximately correct mdp learning and control with temporal logic constraints. In *RSS*, 2014.

[17] N. Fulton and A. Platzer. Safe reinforcement learning via formal methods. 2018.

[18] J. Garcıa and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[19] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

[20] A. Hartmanns and H. Hermanns. The modest toolset: An integrated environment for quantitative modelling and verification. In *TACAS*, volume 8413 of *Lecture Notes in Computer Science*, pages 593–598. Springer, 2014.

[21] M. Hasanbeig, A. Abate, and D. Kroening. Logically-correct reinforcement learning. *arXiv preprint arXiv:1801.08099*, 2018.

[22] M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*, 2015.

[23] J.-P. Katoen. The probabilistic model checking landscape. In *Proc. of LICS*, pages 31–45. ACM, 2016.

[24] A. Kolobov. Planning with markov decision processes: An ai perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–210, 2012.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[26] M. Z. Kwiatkowska. Model checking for probability and time: from theory to practice. In *LICS*, page 351. IEEE Computer Society, 2003.

[27] M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.

[28] G. Mason, R. Calinescu, D. Kudenko, and A. Banks. Assured reinforcement learning with formally verified abstract policies. In *ICAART (2)*, pages 105–117. SciTePress, 2017.

[29] T. M. Moldovan and P. Abbeel. Safe exploration in markov decision processes. *arXiv preprint arXiv:1205.4810*, 2012.

[30] S. Pathak, E. Ábrahám, N. Jansen, A. Tacchella, and J. Katoen. A greedy approach for the efficient repair of stochastic models. In *NFM*, volume 9058 of *Lecture Notes in Computer Science*, pages 295–309. Springer, 2015.

[31] M. Pecka and T. Svoboda. Safe exploration techniques for reinforcement learning–an overview. In *International Workshop on Modelling and Simulation for Autonomous Systems*, pages 357–375. Springer, 2014.

[32] A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[33] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[34] D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. A survey of multi-objective sequential decision-making. *J. Artif. Intell. Res.*, 48:67–113, 2013.

[35] S. J. Russell, D. Dewey, and M. Tegmark. Research priorities for robust and beneficial artificial intelligence. *CoRR*, abs/1602.03506, 2016.

[36] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[37] N. Science and T. C. (NSTC). *Preparing for the Future of Artificial Intelligence*. 2016.

[38] D. Sculley, T. Phillips, D. Ebner, V. Chaudhary, and M. Young. Machine learning: The high-interest credit card of technical debt. 2014.

[39] Sebastian Junges, Nils Jansen, C. Dehnert, U. Topcu, and J. Katoen. Safety-constrained reinforcement learning for MDPs. In *TACAS*, volume 9636 of *LNCS*, pages 130–146. Springer, 2016.

[40] M. Steinmetz, J. Hoffmann, and O. Buffet. Goal probability analysis in probabilistic planning: Exploring and enhancing the state of the art. *J. Artif. Intell. Res.*, 57:229–271, 2016.

[41] I. Stoica, D. Song, R. A. Popa, D. Patterson, M. W. Mahoney, R. Katz, A. D. Joseph, M. Jordan, J. M. Hellerstein, J. E. Gonzalez, et al. A berkeley view of systems challenges for ai. *CoRR*, abs/1712.05855, 2017.

[42] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[43] F. Teichteil-Königsbuch. Stochastic safest and shortest path problems. In *AAAI*. AAAI Press, 2012.

[44] M. Wen, R. Ehlers, and U. Topcu. Correct-by-synthesis reinforcement learning with temporal logic constraints. In *IROS*, 2015.

[45] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[46] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, pages 1433–1438. AAAI Press, 2008.