Online Shielding for Stochastic Systems

Bettina Könighofer[1,3(✉)], Julian Rudolf[1], Alexander Palmisano[1], Martin Tappler[2,3,4], and Roderick Bloem[1]

[1] Institute IAIK, Graz University of Technology, Graz, Austria
bettina.konighofer@iaik.tugraz.at
[2] Institute IST, Graz University of Technology, Graz, Austria
[3] Silicon Austria Labs, TU-Graz SAL DES Lab, Graz, Austria
[4] Schaffhausen Institute of Technology, Schaffhausen, Switzerland

Abstract. We propose a method to develop trustworthy reinforcement learning systems. To ensure safety especially during exploration, we automatically synthesize a correct-by-construction runtime enforcer, called a shield, that blocks all actions of the agent that are unsafe with respect to a temporal logic specification. Our main contribution is a new synthesis algorithm for computing the shield online. Existing offline shielding approaches compute exhaustively the safety of all states-action combinations ahead-of-time, resulting in huge computation times, large memory consumption, and significant delays at runtime due to the look-ups in huge databases. The intuition behind online shielding is to compute at runtime the set of all states that could be reached in the near future. For each of these states, the safety of all available actions is analysed and used for shielding as soon as one of the considered states is reached. Our proposed method is general and can be applied to a wide range of planning problems with stochastic behaviour. For our evaluation, we selected a 2-player version of the classical computer game Snake. The game requires fast decisions and the multiplayer setting induces a large state space, computationally expensive to analyze exhaustively. The safety objective of collision avoidance is easily transferable to a variety of planning tasks.

1 Introduction

Reinforcement Learning (RL) proved successful in solving complex tasks that are difficult to solve using classic controller design, including applications in computer games [33], multi-agent planning [40], and robotics [37]. RL learns high-performance controllers by optimizing objectives expressed via rewards in unknown, stochastic environments. Although learning-enabled controllers (LECs) have the potential to outperform classical controllers, safety concerns prevent LECs from being widely used in real-world tasks [3]. In RL, optimal strategies are obtained without prior knowledge about the environment. Therefore, the safety of actions is not known before their executions. Even after training, there is no guarantee that no unsafe actions are part of the final policy. Having no safety guarantees is unacceptable for safety-critical areas, such as

A. Dutle et al. (Eds.): NFM 2021, LNCS 12673, pp. 231–248, 2021.
https://doi.org/10.1007/978-3-030-76384-8_15

autonomous driving. Safety guarantees take different forms. Especially safety-critical operations require the absence of all unsafe behaviour, while achieving absolute safety for all operations may be impossible due to uncertain, stochastic behaviour. In these cases, safety guarantees may limit the probability of unsafe events.

Shielding [7] is a runtime enforcement technique to ensure safe decision making. By augmenting an RL-agent with a shield unsafe actions are blocked by the shield and the learning agent can only pick a safe action to be sent to the environment. Shields are automatically constructed via correct-by-construction formal synthesis methods from a model of the environment dynamics and a safety specification. Consequently, an agent augmented with a shield is guaranteed to satisfy the safety objective as long as the shield is used. We model the environment via Markov decision processes (MDPs), a popular modelling formalism for decision-making under uncertainty [36,38]. We assess safety by means of probabilistic *temporal logic constraints* [5], which can express different forms of safety guarantees. In this paper, we generally limit the probability to reach critical states in the MDP. For each state and action, exact probabilities are computed on how likely it is that executing this action results in a safety violation from the current state. The shield then blocks all actions whose probability of leading to safety violations exceeds a threshold with respect to an optimally safe action.

The problem with offline shielding. The computation of an offline shield for discrete-event systems requires an exhaustive, ahead-of-time safety analysis for all possible state-action combinations. Therefore, the complexity of offline shield synthesis grows exponentially in the state and action dimension, which limits the application of offline shielding to small environments. Previous work that applied shields in complex, high-dimensional environments relied on over-approximations of the reachable states and domain-oriented abstractions [2,4]. However, this may result in imprecise safety computations of the shield. This way, the shield may become over-restrictive, hindering the learning agent in properly exploring the environment and finding its optimal policy [19].

Our Solution – Online Shielding. Our approach is based on the idea of computing the safety of actions on-the-fly during run time. In many applications, the learning agent does not have to take a decision at every time step. Instead, the learning agent only has to make a decision when reaching a *decision state*. As an example consider a service robot traversing a corridor. The agent has time until the service robot reaches the end of the corridor, i.e., the next decision state, to decide where the service robot should go next. Online shielding uses the time between two decision states to compute the safety of all possible actions in the next decision state. When reaching the next decision state, this information is used to block unsafe actions of the agent. While the online safety analysis incurs a runtime overhead, every single computation of the safety of an action is efficient and parallelisable. Thus, in many settings, expensive offline pre-computation and huge shielding databases with costly lookups are not necessary. Since the safety analysis is performed only for decision states that are actually reached, online shielding is applicable to large, changing, or unknown environments.

In this paper, we solve the problem of shielding a controllable RL-agent in an environment shared with other autonomous agents that perform tasks concurrently. Some combinations of agent positions are safety-critical, as they e.g., correspond to collisions. A safety property may describe the probability of reaching such positions (or other safety properties expressible in temporal logic). The task of the shield is to block actions with a too high risk of leading to such a state. In online shielding, the computation of the safety for any action in the next decision state starts as soon as the controllable agent leaves the current decision state. The tricky part of online shielding in the multi-agent setting is that during the time the RL agent has between two consecutive decisions, the other agents also change their positions. Therefore, online shielding needs to compute the safety of actions with respect to all possible movements of the other agents.

Technically, we use MDPs to formalize the dynamics of the agents operating within the environment. At runtime, we create a small MDP for each decision. These MDPs model the immediate future from the viewpoint of the RL. Via model checking, we determine for the next actions the minimal probability of violating safety. An action is blocked by the shield, if the action violates safety with a probability higher than a threshold relative to the minimal probability.

Contributions. The contributions of this paper comprise (1) the formalisation of online shielding, (2) its implementation via probabilistic model-checking including a demonstrator that is available online[1], and (3) an evaluation of online shielding. The implementation and the evaluation apply shielding to a two-player version of the classic computer game Snake. The evaluation demonstrates that shields can be efficiently computed at runtime, guarantee safety, and have the potential to positively influence learning performance.

Outline. The rest of the paper is structured as follows. Section 1.1 discusses related work. We discuss the relevant foundations in Sect. 2. In Sect. 3, we present the setting and formulate the problem that we address. We present online shielding in Sect. 4, by defining semantics for autonomous agents in the considered setting and defining online shield computations based on these semantics. In Sect. 5, we report on the evaluation of online shielding for the classic computer game Snake. Section 6 concludes the paper with a summary and an outlook on future work.

1.1 Related Work

Runtime enforcement (RE) [12,30,41] covers a wide range of techniques to enforce the correctness of a controller at run-time. The concept of a correct-by-construction *safety-shield* to enforce such correctness with respect to a temporal logic specification was first proposed in [7]. Shields are usually constructed offline by computing a maximally permissive policy containing all actions that will not violate the safety specification. Several extensions exist [4,6,29,39]. The shielding approach has been shown to be successful in combination with RL [2,21].

[1] http://www.onlineshielding.at, accessed: 2020-11-27.

Jansen et al. [19] introduced offline shielding with respect to probabilistic safety. Our work on online shielding directly extends their notion of shielding to the online setting. The offline approach was limited as every action for every state has to be analyzed ahead of time, making the offline approach infeasible for complex environments. Our proposed extension to perform the safety analysis online allows the application of shielding in large, high-dimensional environments.

Pranger et al. [29] proposed *adaptive shields* to enforce quantitative objectives at run-time. While the computation of their shields is performed offline, the authors deal with the consequences of an incorrect or incomplete model that is used for the computation of the shield. During runtime, the authors use abstraction refinement and online probability estimation to update the model and synthesize new shields from the updated models periodically.

Li et al. [24] proposed model predictive shielding (MPS). Given an optimal policy and a safe policy, MPS checks online for each visited state, whether safety will be maintained using the optimal policy. If not, MPS switches to the safe policy. In online shielding, we compute the safety of actions before a decision state is visited, thereby preventing delays at runtime. Furthermore, in online shielding, we do not switch between policies but evaluate all possible decisions to be maximally permissive to the shielded agent.

Safe RL [13,15,27] is concerned with providing safety guarantees for learned agents. Our work focusses on the safe exploration [26], we refer to [15] for other types of safe RL. Using their taxonomy, shielding is an instance of "teacher provides advice" [9], where a teacher with additional information about the system guides the RL agent to pick the right actions. Apprenticeship learning [1] is a closely related variant where the teacher gives (positive) examples and has been used in the context of verification [42]. Uppaal Stratego synthesizes safe, permissive policies that are optimized via learning to create controllers for real-time systems [10]. Some of the work does not assume a model for the environment, making the problem intrinsically harder—and often limiting safety during exploration. We refer to [8,14,16–18] for some interesting approaches.

2 Preliminaries

Sequence and Tuple Notation. We denote sequences of elements by $t = e_0 \cdots e_n$ with ϵ denoting the empty sequence. The length of t is denoted $|t| = n + 1$. We use $t[i] = e_i$ for 0-based indexed access on tuples and sequences. The notation $t[i \leftarrow e'_i]$ represents overwriting of the i^{th} element of t by e'_i, that is, $t[j] = t[i \leftarrow e'_i][j]$ for all $j \neq i$ and $t[i \leftarrow e'_i][i] = e'_i$.

A *probability distribution* over a countable set X is a function $\mu\colon X \to [0,1]$ with $\sum_{x \in X} \mu(x) = 1$. $Distr(X)$ denotes all distributions on X. The support of $\mu \in Distr(X)$ is $supp(\mu) = \{x \in X \mid \mu(x) > 0\}$.

A *Markov decision process* (MDP) $\mathcal{M} = (\mathcal{S}, s_0, \mathcal{A}, \mathcal{P})$ is a tuple with a finite set $\mathcal{S}$ of states, a unique initial state $s_0 \in \mathcal{S}$, a finite set $\mathcal{A} = \{a_1 \ldots, a_n\}$ of actions, and a (partial) *probabilistic transition function* $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to Distr(\mathcal{S})$, where $\mathcal{P}(s,a) = \bot$ denotes undefined behaviour. For all $s \in \mathcal{S}$ the available

actions are $\mathcal{A}(s) = \{a \in \mathcal{A} | \mathcal{P}(s,a) \neq \bot\}$ and we assume $|\mathcal{A}(s)| \geq 1$. A *path* in an MDP $\mathcal{M}$ is a finite (or infinite) sequence $\rho = s_0 a_0 s_1 a_1 \ldots$ with $\mathcal{P}(s_i, a_i)(s_{i+1}) > 0$ for all $i \geq 0$ unless otherwise noted.

Non-deterministic choices in an MDP are resolved by a so-called *policy*. For the properties considered in this paper, memoryless deterministic policies are sufficient [5]. These are functions $\pi : \mathcal{S} \to \mathcal{A}$ with $\pi(s) \in \mathcal{A}(s)$. We denote the set of all memoryless deterministic policies of an MDP by Π. Applying a policy π to an MDP yields an induced *Markov chain* $\mathcal{D} = (\mathcal{S}, s_I, P)$ with $\mathcal{P} : \mathcal{S} \to Distr(\mathcal{S})$ where all nondeterminism is resolved. A *reward function* $r \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$ for an MDP adds a reward to every state s and action a enabled in s.

In formal methods, safety properties are often specified as *linear temporal logic* (LTL) properties [28]. For an MDP $\mathcal{M}$, probabilistic model checking [20,22] employs value iteration or linear programming to compute the probabilities of *all states and actions of the MDP* to satisfy a safety property φ.

Specifically, we compute $\eta^{\max}_{\varphi,\mathcal{M}} \colon \mathcal{S} \to [0,1]$ or $\eta^{\min}_{\varphi,\mathcal{M}} \colon \mathcal{S} \to [0,1]$, which yields for all states the maximal (or minimal) probability over all possible policies to satisfy φ. For instance, for φ encoding to reach a set of states T, $\eta^{\max}_{\varphi,\mathcal{M}}(s)$ is the maximal probability to "eventually" reach a state in T from state $s \in \mathcal{S}$.

3 Setting and Problem Statement

Setting. We consider a setting similar to [19], where one controllable agent, called the *avatar*, and multiple uncontrollable agents, called *adversaries* operate within an *arena*. The arena is a compact, high-level description of the underlying model and captures the dynamic of the agents. Any information on rewards is neglected within the arena since it is not needed for safety computations.

From this arena, potential agent locations may be inferred. Within the arena, the agents perform *tasks* that are sequences of *activities* performed consecutively.

Formally, an *arena* is a pair $G = (V, E)$, where V is a set of nodes and E is a finite set of $E \subseteq V \times V$. An agent's *location* is defined via the current node $v \in V$. An edge $(v, v') \in E$ represents an *activity* of an agent. By executing an activity, the agent moves to its next location v'. A *task* is defined as a non-empty sequence $(v_1, v_2) \cdot (v_2, v_3) \cdot (v_3, v_4) \cdots (v_{n-1}, v_n) \in E^*$ of connected edges. To ease representation, we denote tasks also as sequences of locations $v_1 \cdot v_2 \cdots v_n$.

The set of tasks available in a location $v \in V$ is given by the function $Task(v)$. The set of all tasks of an arena G is denoted by $Task(G)$. The avatar is only able to select a next task at a *decision location* in $V_D \subseteq V$. To avoid deadlocks, we require for any decision location $v \in V_D$ that $Task(v) \neq \emptyset$ and for all $v \cdots v' \in Task(v)$ that $v' \in V_D$, i.e., any task ends in another decision location from which the agent is able to decide on a new task. A safety property may describe that some combinations of agent positions are safety-critical and should not be reached (or any other safety property from the safety fragment of LTL).

Example 1 (Gridworld). Figure 1 shows a simple gridworld with corridors represented by white tiles and walls represented by black tiles. A tile is defined via its (x, y) position. We model this gridworld with an arena $G = (V, E)$ by

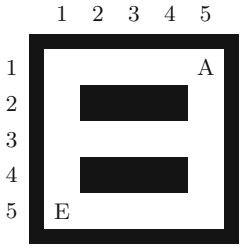

Fig. 1. Gridworld with avatar A (top right) and an adversary E (bottom left).

associating each white tile with a location in V and creating an edge in E for each pair of adjacent white tiles. Corners and crossings are decision locations, i.e., $V_d = \{(1,1),(1,3),(1,5),(5,1),(5,3),(5,5)\}$. At each decision location, tasks define sequences of activities needed to traverse adjoining corridors, e.g., $Task((1,3)) = \{(1,3)\cdot(2,3)\cdot(3,3)\cdot(4,3)\cdot(5,3),\ (1,3)\cdot(1,2)\cdot(1,1),\ (1,3)\cdot(1,4)\cdot(1,5)\}$.

Problem. Consider an environment described by an arena as above and a safety specification. We assume stochastic behaviours for the adversaries, e.g., obtained using RL [31,32]. In fact, this stochastic behaviour determines all actions of the adversaries via probabilities. The underlying model is then an MDP: the avatar executes an action, and upon this execution, the next exact positions (the state of the system) are determined stochastically.

Our aim is to *shield* the decision procedure of the avatar to avoid unsafe behaviour regarding the stochastic movements of the adversaries. *The problem is to compute a shield that prevents the avatar to violate the given safety specification by more than a threshold δ with respect to the optimal safety probability.* The safety analysis of actions is performed on-the-fly allowing the avatar to operate within large arenas.

Example 2 (Gridworld). In Fig. 1, the tile labelled `A` denotes the location of the avatar and the tile labelled `E` denotes the position of an adversary. Let (x_A, y_A) and (x_E, y_E) be the positions of the avatar and the adversary, respectively. A safety property in this scenario is $\neg\mathbf{F}(x_A = x_E \wedge y_A = y_E)$. The negated "eventually" operator $\mathbf{F}$ states that we must not eventually reach a state where the agents collide. Thus, the property specifies safety by requiring that unsafe states must not be reached. We give more details in Sect. 4.3 on how to construct a shield for this setting.

4 Online Shielding for MDPs

In this section, we outline the workflow of online shielding in Fig. 2 and describe it below. Given an arena and behaviour models for adversaries, we define an MDP $\mathcal{M}$ that captures all safety-relevant information. At runtime, we use current

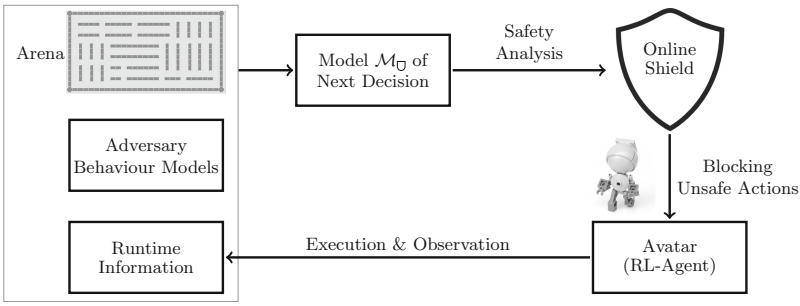

Fig. 2. Workflow of the shield construction.

runtime information to create sub-MDPs $\mathcal{M}_{\cup}$ of $\mathcal{M}$ that model the immediate future of the agents up to some finite horizon. Given such a sub-MDP $\mathcal{M}_{\cup}$ and a safety property φ, we compute via model checking the probability to violate φ within the finite horizon for each task available. The shield then blocks tasks involving a too large risk from the avatar.

4.1 Behaviour Models for Adversaries

The adversaries and the avatar operate within a shared environment, which is represented by an arena $G = (V, E)$, and perform tasks independently. We assume that we are given a stochastic behaviour model of each adversary that determines all task choices of the respective adversary via probabilities. The behaviour of an adversary is formally defined as follows.

Definition 1 (Adversary Behaviour). *For an arena $G = (V, E)$, we define the behaviour B_i of an adversary i as a function $B_i \colon V_D \to Distr(Task(G))$ from decision locations to distributions over tasks, with $supp(B_i(v)) \subseteq Task(v)$.*

Behaviour models of adversaries may be derived using domain knowledge or generalised from observations using machine learning or automata learning [25,35]. A potential approach is to observe adversaries in smaller arenas and transfer knowledge gained in this way to larger arenas [19]. Cooperative and truly adverse behaviour of adversaries may require considering additional aspects in the adversary behaviour, such as the arena state at a specific point in time. Such considerations are beyond the scope of this paper, since complex adversary behaviour generally makes the creation of behaviour models more difficult, whereas the online shield computations are hardly affected.

4.2 Safety-Relevant MDP $\mathcal{M}$

In the following, we describe the safety-relevant MDP $\mathcal{M}$ underlying the agents operating within an arena. This MDP includes non-deterministic choices of the avatar and stochastic behaviour of the adversaries. Note that the safety-relevant

MDP $\mathcal{M}$ is never explicitly created for online shielding, but is explored on-the-fly for the safety analysis of tasks.

Let $G = (V, E)$ be an arena, let *Task* be a task function for G, let B_i with $i \in \{1 \dots m\}$ be the behaviour functions of m adversaries, and let the avatar be the zeroth agent. The safety-relevant MDP $\mathcal{M} = (\mathcal{S}, s_0, \mathcal{A}, \mathcal{P})$ models the arena and agents' dynamics as follows. Each agent has a *position* and a *task queue* containing the activities to be performed from the last chosen task. The agents take turns performing activities from their respective task queue. If the task queue of an agent is empty, a new task has to be selected. The avatar chooses non-deterministically, whereas the adversaries choose probabilistically.

Hence, $\mathcal{M}$ has three types of states: (1) states where the avatar's task queue is empty and the avatar makes a *non-deterministic* decision on its next task, (2) states where an adversary's task queue is empty and the adversary selects its next task *probabilistically*, and (3) states where the currently active agent has a non-empty task queue and the agent processes its task queue *deterministically*.

Formally, the *states* $\mathcal{S} = V^{m+1} \times (E^*)^{m+1} \times \{0, \dots, m\}$ are triples $s = (v, q, t)$ where v encodes the agent positions, q encodes the task queue states of all agents, and t encodes whose turn it is. To enhance readability, we use $pos(s) = s[0] = v$, $task(s) = s[1] = q$, and $turn(s) = s[2] = t$ to access the elements of a state s. We additionally define $ava = 0$, thus $pos(s)[ava]$ and $task(s)[ava]$ are the position and task of the avatar, whereas $turn(s) = ava$ specifies that it is the turn of the avatar. There is a unique action α_{adv} representing adversary decisions, there is a unique action α_{e} representing individual activities (movement along edges), and there are actions for each task available to the avatar, thus $\mathcal{A} = \{\alpha_{\text{adv}}, \alpha_{\text{e}}\} \cup Task(G)$.

Definition 2 (Decision State). *Given a safety-relevant MDP $\mathcal{M}$. We define the set of* decision states $\mathcal{S}_D \subseteq \mathcal{S}$ *via* $\mathcal{S}_D = \{s_D \in \mathcal{S} \mid task(s_D)[ava] = \epsilon \wedge turn(s_D) = ava\}$*, i.e., it is the turn of the avatar and its task queue is empty.*

This implies that if $s_D \in \mathcal{S}_D$, then $pos(s_D)[ava]$ is a decision location in V_D. A policy for $\mathcal{M}$ needs to define actions only for states in $\mathcal{S}_D$, thereby defining the decisions for the avatar. All other task decisions in states s, where $turn(s) \neq ava$, are performed stochastically by adversaries and cannot be controlled.

At run-time, in each turn each agent performs two steps:

(1) If its task queue is empty, the agent has to select its next task.
(2) The agent performs the next activity of its current task queue.

Selecting a new task. A new task has to be selected in all states s with $turn(s) = i$ and $task(s)[i] = \epsilon$, i.e., it is the turn of agent i and agent i's task queue is empty.

If $i = ava$, the avatar is in a decision state $s \in \mathcal{S}_D$, with actions $\mathcal{A}(s) = Task(pos(s)[ava])$. For each task $t \in \mathcal{A}(s)$, there is a successor state s' with $task(s') = task(s)[ava \leftarrow t]$, $pos(s') = pos(s)$, $turn(s') = turn(s)$, and $\mathcal{P}(s, t, s') = 1$. Thus, there is a transition that updates the avatar's task queue with the edges of task t with probability one. Other than that, there are no changes.

If $i \neq ava$, an adversary makes a decision, thus $\mathcal{A}(s) = \alpha_{\text{adv}}$. For each $t \in Task(pos(s)[i])$, there is a state s' with $task(s') = task(s)[i \leftarrow t]$, $pos(s') = pos(s)$,

$turn(s') = turn(s)$, and $\mathcal{P}(s, \alpha_{\text{adv}}, s') = B_i(pos(s)[i])(t)$. There is a single action with a stochastic outcome determined according to the adversary behaviour B_i.

Performing Activities. After potentially selecting a new task, the task queue of agent i is non-empty. We are in a state s', where $task(s')[i] = t = (v_i, v_i') \cdot t'$ with $pos(s')[i] = v_i$. Agent i moves along the edge (v_i, v_i') deterministically and we increment the turn counter modulo $m+1$, i.e., $\mathcal{A}(s') = \{\alpha_{\text{e}}\}$ and $\mathcal{P}(s', \alpha_{\text{e}}, s'') = 1$ with $s''[0] = pos(s')[i \leftarrow v_i']$, $task(s'') = task(s')[i \leftarrow t']$, and $turn(s'') = turn(s') + 1 \mod m+1$.

4.3 Sub-MDP $\mathcal{M}_{\cup}$ for Next Decision

The idea of online shielding is to compute the safety value of actions in the decision states on the fly and block actions that are too risky. For infinite horizon properties, the probability to violate safety, in the long run, is often one and errors stemming from modelling uncertainties may sum up over time [19]. Therefore, we consider safety relative to a *finite horizon* such that the action values (and consequently, a policy for the avatar) carry guarantees for the next steps. Explicitly constructing an MDP $\mathcal{M}$ as outlined above yields a very large number of decision states that may be infeasible to check. The finite horizon assumption allows us to prune the safety-relevant MDP and construct small sub-MDPs $\mathcal{M}_{\cup}$ capturing the immediate future of individual decision states.

More concretely, we consider runtime situations of being in a state s_t, the state visited immediately after the avatar decided to perform a task t. In such situations, we can use the time required to perform t for shield computations for the next decision. We create a sub-MDP $\mathcal{M}_{\cup}$ by determining all states reachable within a finite horizon and use $\mathcal{M}_{\cup}$ to check the safety probability of each action (task) available in the next decision and block unsafe actions.

Construction of $\mathcal{M}_{\cup}$. Online shielding relies on the insight that after deciding on a task t, the time required to complete t can be used to compute a shield for the next decision. Thus, we start the construction of the sub-MDP $\mathcal{M}_{\cup}$ for the next decision location v_D' from the state s_t that immediately follows a decision state s_D, where the avatar has chosen a task $t \in \mathcal{A}(s_D)$. The MDP $\mathcal{M}_{\cup}$ is computed with respect to a finite horizon h for v_D'.

By construction, the task is of the form $t = v_D \cdots v_D'$, where v_D is the avatar's current location and v_D' is the next decision location. While the avatar performs t to reach v_D', the adversaries perform arbitrary tasks and traverse $|t|$ edges, i.e., until v_D' is reached only adversaries take decisions. This leads to a set of possible next decision states. We call these states the *first decision states* $S_{FD} \subseteq \mathcal{S}_{\mathcal{D}}$. After reaching v_D', both avatar and adversaries decide on arbitrary tasks and all agents traverse h edges. This behaviour defines the structure of $\mathcal{M}_{\cup}$.

Given a safety-relevant MDP $\mathcal{M} = (\mathcal{S}, s_0, \mathcal{A}, \mathcal{P})$, a decision state s_D and its successor s_t with $task(s_t)[ava] = t$, and a finite horizon $h \in \mathbb{N}$ representing a number of turns taken by all agents following the next decision. These turns and the (stochastic) agent behaviour leading to the next decision are modelled by the

sub-MDP $\mathcal{M}_{\cup}$. $\mathcal{M}_{\cup} = (\mathcal{S}_{\cup}, s_{0\,\cup}, \mathcal{A}_{\cup}, \mathcal{P}_{\cup})$ is formally constructed as follows. The actions are the same as for $\mathcal{M}$, i.e., $\mathcal{A}_{\cup} = \mathcal{A}$. The initial state is given by $s_{0\,\cup} = (s_t, 0)$. The states of $\mathcal{M}_{\cup}$ are a subset of $\mathcal{M}$'s states augmented with the distance from $s_{0\,\cup}$, i.e., $\mathcal{S}_{\cup} \subseteq \mathcal{S} \times \mathbb{N}_0$. The distance is measured in terms of the number of turns taken by all agents.

We define transitions and states inductively by:

(1) *Decision Actions.* If $(s, d) \in \mathcal{S}_{\cup}$, $d < |t| + h$, and there is an $s' \in \mathcal{S}$ such that $\mathcal{P}(s, \alpha, s') > 0$ and $\alpha \in \{\alpha_{\text{adv}}\} \cup \mathit{Task}(G)$ then $(s', d) \in \mathcal{S}_{\cup}$ and $\mathcal{P}_{\cup}((s, d), \alpha, (s', d)) = \mathcal{P}(s, \alpha, s')$.
(2) *Movement Actions.* If $(s, d) \in \mathcal{S}_{\cup}$, $d < |t|+h$, and there is an $s' \in \mathcal{S}$ such that $\mathcal{P}(s, \alpha_{\text{e}}, s') > 0$, then $(s', d') \in \mathcal{S}_{\cup}$ and $\mathcal{P}_{\cup}((s, d), \alpha_{\text{e}}, (s', d')) = \mathcal{P}(s, \alpha_{\text{e}}, s')$, where $d' = d + 1$ if $\mathit{turn}(s) = m$ and $d' = d$ otherwise.

Movements of the last of $m+1$ agents increase the distance from the initial state. Combined with the fact that every movement action increases the agent index and every decision changes a task queue, we can infer that the structure of $\mathcal{M}_{\cup}$ is a directed acyclic graph. This enables an efficient probabilistic analysis.

By construction, it holds for every state $(s, d) \in \mathcal{S}_{\cup}$ with $d < |t|$, s is not a decision state of $\mathcal{M}$. The set of first decision states S_{FD} consists of all states $s_{FD} = (s, |t|)$ such that $s_{FD} \in \mathcal{S}_{\cup}$ with $\mathit{task}(s)[\mathit{ava}] = \epsilon$ and $\mathit{turn}(s) = \mathit{ava}$, i.e., all first decision states reachable from the initial state of $\mathcal{M}_{\cup}$. We use $\mathit{Task}(S_{FD}) = \{t \mid s \in S_{FD}, t \in \mathcal{A}(s)\}$ to denote the tasks available in these states. $\mathcal{M}_{\cup}$ does not define actions and transitions from states $(s, |t| + h) \in \mathcal{S}_{\cup}$, as their successor states are beyond the considered horizon h. We have $\mathcal{A}((s, d)) \neq \emptyset$ for all states at distance $d < |t| + h$ from the initial state.

4.4 Shield Construction

The probability of reaching a set of unsafe states $T \in \mathcal{S}$ from any state in the safety-relevant MDP should be low. In the finite horizon setting, we are interested in bounded reachability from decision states $s_D \in \mathcal{S}_D$ within the finite horizon h. We concretely evaluate reachability on sub-MDPs $\mathcal{M}_{\cup}$ and use $T_{\cup} = \{(s, d) \in S_{\cup} \mid s \in T\}$ to denote the unsafe states that may be reached within the horizon covered by $\mathcal{M}_{\cup}$. The property $\varphi = \Diamond T_{\cup}$ encodes the **violation** of the safety constraint, i.e., eventually reaching $T_{\cup}$ within $\mathcal{M}_{\cup}$. The shield needs to limit the probability to *satisfy* φ.

Given a sub-MDP $\mathcal{M}_{\cup}$ and a set of first decision states S_{FD}. For each task $t \in \mathit{Task}(S_{FD})$, we evaluate t with respect to the minimal probability to satisfy φ from the initial state $s_{0\,\cup}$ when executing t by computing $\eta^{\min}_{\varphi, \mathcal{M}_{\cup}}(s_{0\,\cup})$. This is formalised with the notion of task-valuations below.

Definition 3 (Task-valuation). *A* task-valuation *for a task t in a sub-MDP $\mathcal{M}_{\cup}$ with initial state $s_{0\,\cup}$ and first decision states S_{FD} is given by*

$$\mathit{val}_{\mathcal{M}_{\cup}} \colon \mathit{Task}(S_{FD}) \to [0, 1], \text{ with } \mathit{val}_{\mathcal{M}_{\cup}}(t) = \eta^{\min}_{\varphi, \mathcal{M}_{\cup}}(s_{0\,\cup}),$$
$$\text{and } \mathcal{A}(s_{FD}) = \{t\} \text{ for each } s_{FD} \in S_{FD}.$$

The optimal task-value *for* $\mathcal{M}_{\sqcup}$ *is* $optval_{\mathcal{M}_{\sqcup}} = \min_{t' \in Task(S_{\mathrm{FD}})} val_{\mathcal{M}_{\sqcup}}(t')$.

A task-valuation is the minimal probability to reach an unsafe state in T from each immediately reachable decision state $s_{FD} \in S_{\mathrm{FD}}$ weighted by the probability to reach s_{FD}. When the avatar chooses an optimal task t (with $val_{\mathcal{M}_{\sqcup}}(t) = optval_{\mathcal{M}_{\sqcup}}$) as next task in a state s_{FD}, $optval_{\mathcal{M}_{\sqcup}}$ can be achieved if all subsequent decisions are optimal as well.

We now define a shield for the decision states S_{FD} in a sub-MDP $\mathcal{M}_{\sqcup}$ using the task-valuations. Specifically, a *shield* for a threshold $\delta \in [0,1]$ determines a set of tasks available in S_{FD} that are δ-optimal for the specification φ. All other tasks are "shielded" or "blocked".

Definition 4 (Shield). *For task-valuation* $val_{\mathcal{M}_{\sqcup}}$ *and a threshold* $\delta \in [0,1]$*, a* shield for S_{FD} *in* $\mathcal{M}_{\sqcup}$ *is given by*

$$shield_{\delta}^{\mathcal{M}_{\sqcup}} \in 2^{Task(S_{\mathrm{FD}})} \text{ with}$$
$$shield_{\delta}^{\mathcal{M}_{\sqcup}} = \{t \in Task(S_{\mathrm{FD}}) \mid \delta \cdot val_{\mathcal{M}_{\sqcup}}(t) \leq optval_{\mathcal{M}_{\sqcup}}\}.$$

Intuitively, δ enforces a constraint on tasks that are acceptable w.r.t. the optimal probability. The shield is *adaptive* with respect to δ, as a high value for δ yields a stricter shield, a smaller value a more permissive shield. In particularly critical situations, the shield can enforce the decision maker to resort to (only) the optimal actions w.r.t. the safety objective. This can be achieved by temporarily setting $\delta = 1$. Online shielding creates shields on-the-fly by constructing sub-MDPs $\mathcal{M}_{\sqcup}$ and computing task-valuations for all available tasks.

Through online shielding, we transform the safety-relevant MDP $\mathcal{M}$ into a *shielded MDP* with which the avatar interacts (which is never explicitly created) that is obtained from the composition of all sub-MDP$\mathcal{M}_{\sqcup}$. Due to the assumption on the task functions that requires a non-empty set of available tasks in all decision locations and due to the fact that every decision for shielding is defined w.r.t. an optimal task, the shielded MDP is deadlock-free. Hence, our notion of online shielding guarantees deadlock-freedom and optimality w.r.t. safety. By using the minimal probability as task valuation $val_{\mathcal{M}_{\sqcup}}(t)$, we assume that the avatar performs optimally with respect to safety in upcoming decisions. Alternatively, we could use the maximal probability in combination with a fixed threshold $\lambda \in [0,1]$ such that only tasks t with $val_{\mathcal{M}_{\sqcup}}(t) \leq \lambda$ are allowed. This would place weaker assumptions on the avatar behaviour, but it may induce deadlocks in case there are no sufficiently safe actions.

4.5 Optimisation – Updating Shields After Adversary Decisions

After the avatar decides on a task, we use the time to complete the task to compute shields based on task-valuations (see Definition 3 and Definition 4). Such shield computations are inherently affected by uncertainties stemming from stochastic adversary behaviour. These uncertainties consequently decrease

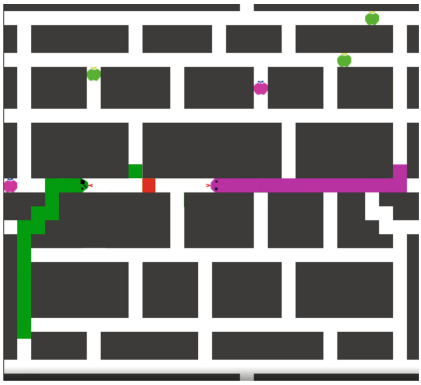

Fig. 3. A screenshot of the SNAKE game with colour-coded shield display. (Color figure online)

whenever we observe a concrete decision from an adversary that we considered stochastic in the initial shield computation.

An optimisation of the online shielding approach is to compute a new shield after any decision of an adversary, if there is enough remaining time until the next decision location. Suppose that after visiting a decision state, we computed a shield based on $\mathcal{M}_{\cup}$. While moving to the next decision state, an adversary decides on a new task and we observe the concrete state s. We can now construct a new sub-MDP $\mathcal{M}'_{\cup}$ using $s'_{0\,\cup} = s$ as initial state, thereby resolving a stochastic decision between the original initial state $s_{0\,\cup}$ and $s'_{0\,\cup}$. Using $\mathcal{M}'_{\cup}$, we compute a new shield for the next decision location.

The facts that the probabilistic transition function of $\mathcal{M}_{\cup}$ does not change during updates and that we consider safety properties enable a very efficient implementation of updates. For instance, if value iteration is used to compute task-valuations, we can simply change the initial state and reuse computations from the initial shield computation. Note that if a task is completely safe, i.e., tasks with a valuation of zero, the value of this task will not change under a re-computation, since the task is safe under any sequence of adversary decisions.

5 Implementation and Experiments

2-Player SNAKE. We implemented online shielding for a two-player version of the classic computer game SNAKE. We picked the game because it requires fast decision making during runtime, and provides in an intuitive and fun setting to show the potential of shielding such that it can potentially be used for teaching formal methods. Figure 3 shows a screenshot of the 2-Player SNAKE game on the map that was used for the experiments. In the game, each player controls a snake of a different colour. Here, the green snake is controlled by the avatar (the RL-agent) and the purple snake by the adversary. The goal for each player is to eat five randomly positioned apples of their own colour. The score for the

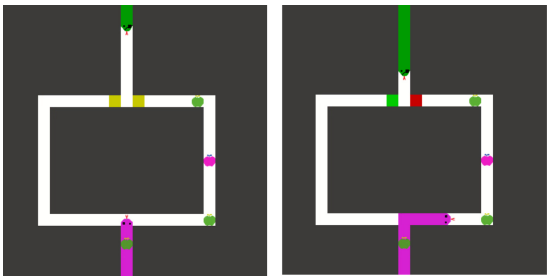

Fig. 4. Screenshots from the SNAKE game to demonstrate recalculation.

green snake (the avatar) is positively affected (+10) by collecting a green apple and by wins of the avatar (+50), i.e., if it collects all green apples before the adversary snake collects all purple apples. If a snake has a collision, the snake loses. In case that the heads of both snakes collide, the avatar loses.

We implemented a shield to protect the avatar snake from collisions with the adversary snake. The shield computes online the minimal probability that taking the next corridor will lead to a collision. The game, as shown in Fig. 3, indicates the risk of taking a corridor from low to high by the colours green, yellow, orange, red.

We also implemented the optimisation to recalculate the shield after a decision of the adversary snake. Figure 4 contains two screenshots of the game on a simple map to demonstrate the effect of a shield update. In the left figure, the available tasks of the green snake are picking the corridor to the left or the corridor to the right. Both choices induce a risk of a collision with the purple snake. After the decision of the purple snake to take the corridor to its right-hand-side, the shield is updated and the safety values of the corridors change.

Experimental Set-up. The Python-based implementation can be found at http://onlineshielding.at along with videos, evaluation data and a Docker image that enables easy experimentation. For shield computations, we use the probabilistic model checker `Storm` [11] and its Python interface. We use the `PRISM` [23] language to represent MDPs and domain-specific optimisations to efficiently encode agents and tasks, that is, snakes and their movements. Reinforcement learning is implemented via approximate Q-learning [34] with the feature vector denoting the distance to the next apple. The Q-learning uses the learning rate $\alpha = 0.1$ and the discount factor $\gamma = 0.5$ for the Q-update and an ϵ-greedy exploration policy with $\epsilon = 0.6$. The pygame[2] library is used to implement the game's interface and logic. All experiments have been performed on a computer with an Intel®Core™ i7-4700MQ CPU with 2.4 GHz, 8 cores and 16 GB RAM.

[2] https://www.pygame.org/, accessed 2020-11-27.

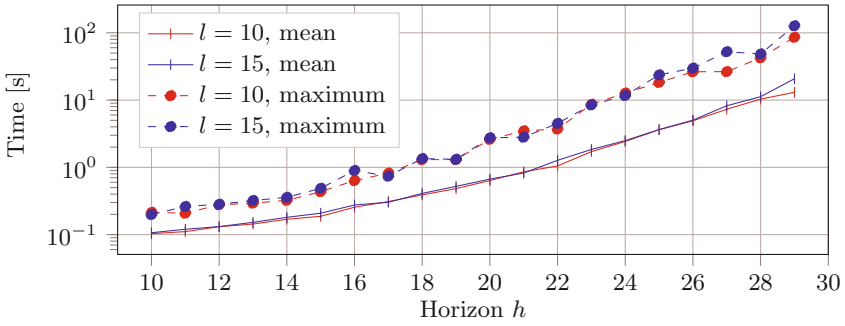

Fig. 5. Shield computation time for varying horizon values and snake lengths.

Evaluation Criteria. We report on two types of experiments: (1) the time required to compute shields relative to the computation horizon and (2) the performance of shielded reinforcement learning compared to unshielded reinforcement learning measured in terms of gained reward. The experiments on computation time indicate how many steps shielding can look ahead within some given time. The experiments on learning performance demonstrate the effect of shielding.

Computation Time Measurements. When playing the game on the map illustrated in Fig. 3, we measured the time to compute shields, i.e., the time to construct sub-MDPs $\mathcal{M}_{\cup}$ and to compute the safety values. We measured the time of 200 such shield computations and report the maximum computation times and the mean computation times. Figure 5 presents the results for two different snake lengths $l \in \{10, 15\}$ and different computation horizons $h \in \{10, 11, \ldots, 29\}$. The x-axis displays the computation horizon h and the y-axis displays the computation time in seconds in logarithmic scale.

We can observe that up to a horizon h of 17, all computations take less than one second, even in the worst case. Assuming that every task takes at least one second, we can plan ahead by taking into account safety hazards within the next 17 steps. A computation horizon of 20 still requires less than one second on average and about 3 seconds in the worst case. Horizons in this range are often sufficient, as we demonstrate in the next experiment by using $h = 15$.

We compare our timing results with a similar case study presented by Jansen et al. [19]. In a similar multi-agent setting on a comparably large map, the decisions of the avatar were shielded using an offline shield with a finite horizon of 10. The computation time to compute the offline shield was about 6 h on a standard notebook. Note, that although the setting has four adversaries, the offline computation was performed for one adversary and the results were combined for several adversaries online.

Furthermore, Fig. 5 shows that the snake length affects the computation time only slightly. This observation supports our claim that online shielding scales

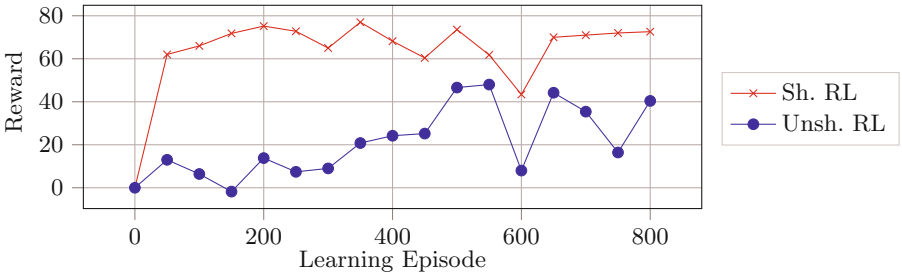

Fig. 6. Reward gained throughout learning for shielded and unshielded RL.

well to large arenas, i.e., scenarios where the safety-relevant MDP $\mathcal{M}$ is large. Note that the number of game configurations grows exponentially with the snake length (assuming a sufficiently large map), as the snake's tail may bend in different directions at each crossing.

The experiments further show that the computation time grows exponentially with the horizon. Horizons close to 30 may be advantageous in especially safety-critical settings, such as factories with industrial robots acting as agents. Since individual tasks in a factory may take minutes, online shielding would be feasible, as worst-case computation times are in the range of minutes. However, offline shielding would be infeasible due to the average computation time of more than 10 s that would be required for all decision states.

RL with Online Shielding. Figure 6 shows plots of the reward gained during learning in the shielded and the unshielded case. The online shield uses a horizon of $h = 15$. The y-axis displays the reward and the x-axis displays the learning episodes, where one episode corresponds to one play of the Snake game. The reward has been averaged over 50 episodes for each data point.

The plot demonstrates that shielding improves the gained reward significantly. By blocking unsafe actions, the avatar did not encounter a single loss due to a collision. For this reason, we see a consistently high reward right from the start of the learning phase. In the execution phase, shielded RL manages to win about 96% of all plays, whereas unshielded RL wins only about 54%.

6 Conclusion and Future Work

Online shielding is an efficient approach to enforce safe behaviour of autonomous agents operating within a stochastic environment. The approach exploits the time required to complete tasks to model and analyse the immediate future w.r.t. a safety property. For every decision at runtime, we create MDPs to model the current state of the environment and the behaviour of the agents. Given these MDP models, we employ probabilistic model-checking to evaluate every action possible in the next decision. In particular, we determine the probability of unsafe

behaviour following every possible choice. This information is used by shields to block unsafe actions, i.e., actions leading to safety violations with a probability exceeding a threshold relative to the minimal probability of safety violations.

For future work, we plan to investigate the application of online shielding in other settings, such as decision making in robotics and control. Another interesting extension would be to incorporate quantitative performance measures in the form of rewards and costs into the computation of the online shield, as previously demonstrated in an offline manner [4] and in a hybrid approach [29], where runtime information was used to learn the environment dynamics.

Acknowledgments. This work has been supported by the "University SAL Labs" initiative of Silicon Austria Labs (SAL) and its Austrian partner universities for applied fundamental research for electronic based systems.

References

1. Abbeel, P., Ng, A.Y.: Exploration and apprenticeship learning in reinforcement learning. In: ICML. ACM International Conference Proceeding Series, vol. 119, pp. 1–8. ACM (2005)
2. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding. In: AAAI. AAAI Press (2018)
3. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., Mané, D.: Concrete problems in AI safety. CoRR, abs/1606.06565 (2016)
4. Avni, G., Bloem, R., Chatterjee, K., Henzinger, T.A., Könighofer, B., Pranger, S.: Run-time optimization for learned controllers through quantitative games. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 630–649. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_36
5. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
6. Bharadwaj, S., Bloem, R., Dimitrova, R., Könighofer, B., Topcu, U.: Synthesis of minimum-cost shields for multi-agent systems. In: ACC, pp. 1048–1055. IEEE (2019)
7. Bloem, R., Könighofer, B., Könighofer, R., Wang, C.: Shield synthesis: runtime enforcement for reactive systems. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 533–548. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_51
8. Cheng, R., Orosz, G., Murray, R.M., Burdick, J.W.: End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In: AAAI (2019)
9. Clouse, J.A., Utgoff, P.E.: A teaching method for reinforcement learning. In: ML, pp. 92–110. Morgan Kaufmann (1992)
10. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: Uppaal Stratego. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 206–211. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_16
11. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A STORM is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_31

12. Falcone, Y., Pinisetty, S.: On the runtime enforcement of timed properties. In: Finkbeiner, B., Mariani, L. (eds.) RV 2019. LNCS, vol. 11757, pp. 48–69. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32079-9_4
13. Fulton, N., Platzer, A.: Verifiably safe off-model reinforcement learning. In: TACAS, pp. 413–430 (2019)
14. Fulton, N., Platzer, A.: Verifiably safe off-model reinforcement learning. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 413–430. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17462-0_28
15. Garcıa, J., Fernández, F.: A comprehensive survey on safe reinforcement learning. J. Mach. Learn. Res. **16**(1), 1437–1480 (2015)
16. Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Omega-regular objectives in model-free reinforcement learning. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 395–412. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17462-0_27
17. Hasanbeig, M., Abate, A., Kroening, D.: Logically-correct reinforcement learning. CoRR, abs/1801.08099 (2018)
18. Hasanbeig, M., Kantaros, Y., Abate, A., Kroening, D., Pappas, G.J., Lee, I.: Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In: CDC, pp. 5338–5343. IEEE (2019)
19. Jansen, N., Könighofer, B., Junges, S., Serban, A., Bloem, R.: Safe reinforcement learning using probabilistic shields (invited paper). In: Konnov, I., Kovács, L. (eds.) CONCUR, volume 171 of LIPIcs, pp. 3:1–3:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
20. Katoen, J.-P.: The probabilistic model checking landscape. In: LICS, pp. 31–45. ACM (2016)
21. Könighofer, B., Lorber, F., Jansen, N., Bloem, R.: Shield synthesis for reinforcement learning. In: Margaria, T., Steffen, B. (eds.) ISoLA 2020. LNCS, vol. 12476, pp. 290–306. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61362-4_16
22. Kwiatkowska, M.Z.: Model checking for probability and time: from theory to practice. In: LICS, pp. 351. IEEE CS (2003)
23. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
24. Li, S., Bastani, O.: Robust model predictive shielding for safe reinforcement learning with stochastic dynamics. In: ICRA, pp. 7166–7172. IEEE (2020)
25. Mao, H., Chen, Y., Jaeger, M., Nielsen, T.D., Larsen, K.G., Nielsen, B.: Learning deterministic probabilistic automata from a model checking perspective. Mach. Learn. **105**(2), 255–299 (2016). https://doi.org/10.1007/s10994-016-5565-9
26. Moldovan, T.M., Abbeel, P.: Safe exploration in Markov decision processes. In: ICML. icml.cc/Omnipress (2012)
27. Pecka, M., Svoboda, T.: Safe exploration techniques for reinforcement learning - an overview. In: Hodicky, J. (ed.) MESAS 2014. LNCS, vol. 8906. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13823-7_31
28. Pnueli, A.: The temporal logic of programs. In: Foundations of Computer Science, pp. 46–57. IEEE (1977)
29. Pranger, S., Könighofer, B., Tappler, M., Deixelberger, M., Jansen, N., Bloem, R.: Adaptive shielding under uncertainty. CoRR, abs/2010.03842 (2020)

30. Renard, M., Falcone, Y., Rollet, A., Jéron, T., Marchand, H.: Optimal enforcement of (timed) properties with uncontrollable events. Math. Struct. Comput. Sci. **29**(1), 169–214 (2019)
31. Sadigh, D., Landolfi, N., Sastry, S.S., Seshia, S.A., Dragan, A.D.: Planning for cars that coordinate with people: leveraging effects on human actions for planning and active information gathering over human internal state. Auton. Robot. **42**(7), 1405–1426 (2018). https://doi.org/10.1007/s10514-018-9746-1
32. Sadigh, D., Sastry, S., Seshia, S.A., Dragan, A.D.: Planning for autonomous cars that leverage effects on human actions. Science and Systems. In: Robotics (2016)
33. Silver, D., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484 (2016)
34. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
35. Tappler, M., Aichernig, B.K., Bacci, G., Eichlseder, M., Larsen, K.G.: L^*-based learning of Markov decision processes. In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) FM 2019. LNCS, vol. 11800, pp. 651–669. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30942-8_38
36. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. The MIT Press, Cambridge (2005)
37. Wang, A., Kurutach, T., Liu, K., Abbeel, P., Tamar, A.: Learning robotic manipulation through visual planning and acting. arXiv preprint arXiv:1905.04411 (2019)
38. White, D.J.: Real applications of Markov decision processes. Interfaces **15**(6), 73–83 (1985)
39. Wu, M., Wang, J., Deshmukh, J., Wang, C.: Shield synthesis for real: enforcing safety in cyber-physical systems. In: FMCAD, pp. 129–137. IEEE (2019)
40. Zhang, W., Bastani, O.: MAMPS: safe multi-agent reinforcement learning via model predictive shielding. CoRR, abs/1910.12639 (2019)
41. Zhou, W., Gao, R., Kim, B., Kang, E., Li, W.: Runtime-safety-guided policy repair. In: RV, pp. 131–150 (2020)
42. Zhou, W., Li, W.: Safety-aware apprenticeship learning. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 662–680. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_38