Thesis!

by
Astrid Horn Brorholt

AALBORG UNIVERSITET

Main Supervisor:	Kim Guldstrand Larsen Aalborg University, Denmark
Co-supervisors:	Christian Schilling Aalborg University, Denmark
	Florian Lorber Silicon Labs, Austria
	Peter Gjøl Jensen Aalborg University, Denmark
Assessment:	Title and name Affiliation and country
Department:	Dept. of Computer Science, Aalborg Univesity

Use of AI: This thesis explores safe reinforcement learning. Large language models have not been used to produce any text presented in this thesis. Generative AI has been used to a limited extent for generating parts of figures, and for generation of source code in experiments.

Abstract

Hey bigsby, summarize this thesis in 200 words

Resumé

Det danske abstract. Frem med den tekniske ordbog.

Table of Contents

Introduction

Digital control of physical components enables time-saving automation and efficient use of available resources. This can range from a simple on/off switch to a complex neural network managing multiple processes. It is not uncommon for several digital components to be deployed in concert to serve complementary purposes. Such cyber-physical systems [1], [2] are becoming more ubiquitous and more advanced.

With applications such as autonomous vehicles, water management systems, industrial hydraulics, or power controllers, great care must be taken to ensure the safety of people, equipment, and resources that are directly or indirectly affected by the system.

This can be achieved through the field of formal methods, which has a wide variety of approaches that can provide proof that a given system restricts itself ot a safe subset of behaviours. [handbook of model checking (?)] This presumes an accurate model of the (cyber-physical) system under verification and techniques are most often subject to "state-space explosion," where the complexity of verification is highly sensitive to the size of the model.

Neural networks are notable for having achieved impressive performance in a wide variety of tasks [alphago, atari games, muzero, chatgpt]. This performance is achieved by controllers that use a high number of neurons, making direct formal verification infeasible.

TODO: Mention the term multi-objective optimization and the trade-off between safety and efficiency.

1 Reinforcement Learning

Reinforcement learning [3], [4] is a major class of machine learning techniques along with supervised learning, and unsupervised learning [5]. In supervised learning, models learn from labelled data, to predict the labels of unseen data. Unsupervised (or self-supervised) learning similarly trains the model on a set amount of unlabelled data, to discover relevant patterns and approximations. In contrast, reinforcement learning *agents* are actively interacting with a system, directing exploration and receiving observation data and reward, as the system responds to actions taken by the agent.

The interaction between an agent and a system is illustrated in Figure 1 as an unending loop. The agent observes its current state, and makes a decision on which action to take, based on its current policy and exploration strategy (e.g. ε-greedy).

Taking the action yields a reward that the agent can use to update its policy, and an observation of the updated state which it will use to pick the next action.

The reinforcement learning problem can be stated in many different ways – depending on the nature of the problem – but is perhaps most commonly defined in terms of a Markov decision process (MDP) [6]. MDPs describe stochastic systems, where the outcomes of actions only depend on the current (observable) state of the system, and not on which actions or states were seen previously. An MDP can be described by a tuple $(S, s_0 Act, P, R)$ where

- S is a set of states,
- $s_0 \in S$ is an initial state,
- Act is a set of actions,
- $P : S \times Act \times S \to [0; 1]$ with $\forall s \in S, a \in Act : \sum_{s' \in S} P(s, a, s') = 1$ gives the probability of reaching state s' from state s as a result of taking the action a,
- and $R : S \times Act \to \mathbb{R}$ gives the reward $R(s, a)$ for taking action a in state s.

A *policy* (or strategy) is any method – such as a reinforcement learning agent – for choosing the next action from a given state. Policies can either be

- *probabilistic* $S \times Act \to [0; 1]$, giving a probability distribution over actions,
- *deterministic* $S \to Act$, uniquely selecting one specific action for each state,
- or *nondeterministic* $S \to \mathscr{P}(Act)$, giving a subset $A \in Act$ of actions, one of which may be chosen according to some other mechanism.

Given an e.g. nondeterministic policy $\sigma : S \to [0; 1]$, a trace ξ of an MDP is an interleaved series of states and actions $\xi = s_0 a_0 s_1 a_1 s_2 a_2 \ldots$ such that $a_i \in \sigma(s_i)$ and $P(s_i, a_i, s_{i+1}) > 0$. Traces can be both finite or infinite. Given a set $\varphi \subset S$ of safe states, we say a trace is safe (with regards to φ) if for every s_i in ξ, $s_i \in \varphi$.

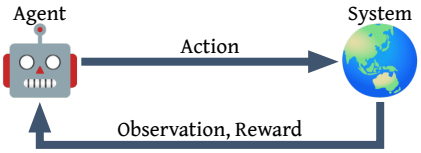


Figure 1: The reinforcement learning loop.

It can sometimes be useful to view machine learning as consisting of two different phases: Initial *training,* and subsequent *operation* as part of a real-life system. In the common view of reinforcement learning, the agent is continually exploring, learning, and improving, even when in operation. However, this is not always the case in practice. Legal requirements may warrant a costly re-certification every time changes are made to a controller, prohibiting the agent from adapting its behaviour during operation. Technical limitations during operations may also preclude learning, such as reductions applied to the model, in order to deploy it to an embedded platform.

2 Shielding

TODO: Introduce safety as a set of states, as opposed to liveness

Even when a controller cannot be verified directly, other approaches can be used to verify the safety of the system as a whole. In [7] it was shown how a safety property can be enforced through a maximally permissive, non-deterministic strategy. While acting within the constraints of this strategy, reinforcement learning was utilized to optimize for a second objective, achieving a near-optimal strategy within the safety constraints.

The term **shield** was coined in [8] to describe a component which would work in concert with a (mostly safe) controller, and intervene to prevent unsafe behaviour. Thus, the behaviour of the shield and controller together is verifiably safe, as long as the shield is safe. Contrary to runtime monitors [Citation Needed], which enforce a property by halting the system if the property is not satisfied, the shield will intervene by altering the actions of the controller without knowing future input/output. The authors proposed guarantees of minimal interference, and a property of k-stabilization, which states that the shield will at most intervene k times before control is handed back to the controller.

This concept was extended to a framework of **shielded reinforcement learning** in [9]. Here, a shield monitors and possibly corrects the actions of a learning agent, which enables safe exploration. This enables the safe use of complex learning agents that can achieve cost optimal behaviour. Approaches such as deep Q-learning or proximal policy optimization can be safely used in this framework, even though these methods cannot feasibly be verified directly. The paper also points out that a shield can be synthesized from an **abstract model** of the system, one which only models behaviour relevant to the safety property being enforced. Such an abstraction could be significantly simpler than the full system, allowing shielded reinforcement learning to scale to systems where other methods for safe and optimal control are infeasible.

Since this first article covering shielded reinforcement learning in finite MDPs, other shielding methods building upon the same framework have been described in the literature [Every shielded RL article I have on hand].

A shield can be viewed as a nondeterministic strategy $\shield : S \to \mathcal{P}(Act)$ for a safety property φ on an MDP $\mathcal{M}$ such that any trace ξ that is an outcome of $\shield$ is safe with regards to φ. A shield $\shield^*$ is said to be *maximally permissive* (or minimally interfering) [8], [9], [10] if for all shields $\shield$ that are safe wrt. $\mathcal{M}$ and φ,

TODO: An example around here somewhere would be helpful.

TODO: Define maximally permissive

Finite- and infinite-horizon shielding

In many domains, the controller will continue to function indefinitely, which warrants safety guarantees to match. Ideally, the shield should be able to ensure that as long as the shield is applied, the system is safe forever, such as with the shielding methods described in [all of them].

In some cases, it can make sense to only give guarantees k steps into the future. These finite horizon shields are often referred to as k-step lookahead shields [Citation Needed]. This may be desirable for models where infinite-horizon safety strategies do not exist, or are infeasible to compute.

Probabilistic shielding

When uncertainty is inherent to a system, it might not be possible for any strategy to guarantee safety for all traces starting in the initial state. Although such *absolute guarantees* are not available, it might still be possible to give guarantees relating to staying safe with a specific probability. One such guarantee could be a k-step lookahead shield which guarantees a maximum risk of safety violation to occur within those k steps. This leaves the possibility, that the state at step $k+1$ is unsafe for all actions that could be taken at k.

TODO: Expand this section with types of probabilistic guarantees.

Obtaining a Model

Many formal verification tools assume that an accurate model of the system is available. This model could be provided by a domain expert, but in other cases it might not be available. When no models are available, some things like erm automata learning or uncertain MDPs might be useful. [Citation Needed] [11]

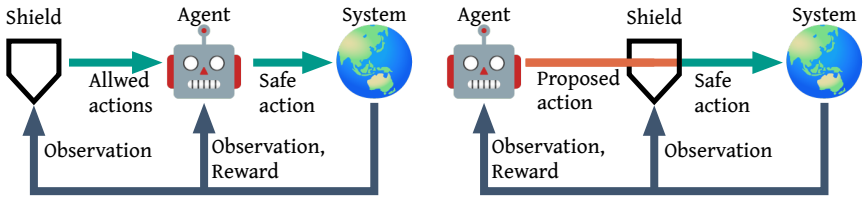
TODO: Expand this section.

2.1 Applying the Shield

The methods of corrective action taken by the shield can vary depending on the model and the application. The terms pre- and post-shielding have been used in the literature to describe a shield's relationship with the controller, but with two distinct sets of meaning:

1. In one part of the literature, pre- and post-shielding refer to **how** the shield ensures only safe actions reach environment [11], [12], [13], [14].
2. Alternatively the terms can refer to **when** a shield is applied, in the process of obtaining a controller [10], [15].

In the following, we shall use the terms pre- and post-shielding to mean the former, while we dub the latter meaning resp. end-to-end shielding and post-hoc shielding.

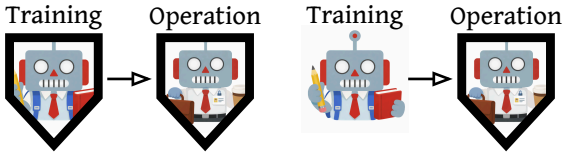


(a) Pre-shielding (b) Post-shielding

Figure 2: The shield can enforce safety in multiple ways

Pre-shielding This term refers to the shield restricting the behaviour the controller, by providing a set of actions $A \subseteq Act$ that are permitted for the given state. The controller must be set up in such a way as to only pick an action a if it is included in the set A it receives from the shield.

Post-shielding Contrary to pre-shielding, this configuration is transparent to the controller. In post-shielding the controller outputs an action a to the shield, rather than sending it directly to the environment. The shield then evaluates the action a, checking if it is in the set of permissible actions $a \in A$s. If the action is permitted, a is sent to the environment unaltered. Otherwise, the action a is replaced with an alternative, permissible action $a' \in A$.



(a) End-to-end shielding (b) Post-hoc shielding

Figure 3: The shield may or may not be in place during training.

End-to-end Shielding In the context of reinforcement learning, when the shield is in place during *both* the learning *and* operational phases, this is called end-to-end shielding. This is necessary if the agent is interacting with a real-life system where safety violations during training are to be avoided. End-to-end shielding was seen in [9] to lead to faster convergence, as the shield acts as a teacher guiding the agent away from undesirable behaviours.

More generally, end-to-end shielding describes the process of integrating the shield in the design process of the controller, omitting unsafe actions from consideration.

Post-hoc Shielding Alternatively, the shield can be applied only in the operational phase, allowing the agent to explore unsafe actions during learning. This can lead to slower convergence, since the agent spends time exploring unsafe states and (ideally) learning to avoid them. It is conceivable that the shield does not alter the behaviour of the agent. This can happen if the shield is maximally permissive while

the agent has learned to behave safely. However, in other cases the shield may interfere with the policy learned by the agent, which violates the key assumption of RL that the environment remains static [Citation Needed].

Outside the context of RL, the term describes applying the shield as a guardrail of a controller that has been designed without the shield as reference. Thus, an existing controller can be upgraded to give formal safety guarantees by applying a post-hoc shield in cases where altering the controller itself would be costly.

Method of correction	Time of correction
Pre-shielding	End-to-end shielding
Post-shielding	Post-hoc shielding

Table 1: A term from each column can be combined to describe how the shield works in tandem with the controller.

Terminology in Paper A

Note that what was referred to in Paper A [10] as post-shielding, we refer to here as post-hoc shielding. Conversely, what was referred to as pre-shielding was described in this section as end-to-end shielding.

Using the terminology of this section, the paper examines the difference between end-to-end- and post-hoc shielding. The paper utilizes post-shields, and examines different action selection strategies when the shield intervenes.

2.2 Effects on Convergence in RL

Convergence guarantees for reinforcement learning methods usually require the environment under learning to satisfy the Markov property [Q-learning, DQL, PPO]. I.e the state must be fully observable, and the probability distributions within the model do not change.

In reinforcement learning methods such as Q-learning [16], actions can be removed from consideration in states by omitting them in the Bellman equation update, or by initializing their Q-value as negative infinity. A similar approach works for gradient methods [4], [17], [18]. As such, it is straightforward to apply a pre-shield by restricting unsafe actions as provided by the shield.

As argued in [9], a post-shield $\triangledown$ applied to an environment $\mathcal{M}$ can be viewed as a new environment $\mathcal{M}^{\triangledown}$. Therefore, convergence guarantees are preserved for the combined $\mathcal{M}^{\triangledown}$ as long as the shield satisfies the same assumptions as the environment (i.e. Markov properties).

Some care must be taking when performing value updates using a post-shield. Say that in state s, the shield alters an unsafe action a to the safe alternative a', receiv-

ing reward r. The straightforward approach would be to treat the shield as part of the environment, and performing value update using the triple (s, a, r). It was shown in [19] through a series of deep Q-learning experiments, that the number of interventions of a shield was reduced changing the value update scheme: The reward r was applied to the learning agent as (s, a', r) while a penalty p was applied to the suggested action, (s, a, p). Other approaches may be unsound, such as applying only the update (s, a', r).

Pre-shielding will likely converge faster than post-shielding in general. If a model has a state s, with one safe action a_1 and unsafe actions a_2, a_3, a post-shielded agent will have to explore actions a_1, a_2 and a_3 in order to learn the expected reward attainable in s. However, a post-shielded agent will only have to learn the expected reward of a_1, since the other actions are masked. Thus, it will likely gain a more precise estimate of the expected value of s from the same amount of visits to the state.

Multi-agent shielding

In many cyber-physical systems, several components are working together to achieve distinct goals in the entire system. Since formal verification methods are highly affected by issues of scalability, multi-agent approaches are useful. There is no silver bullet.

TODO: Expand this section.

3 State of the Art

3.1 Shielding of Hybrid Systems

3.2 Tools for Shielding

[uppaal] [storm] [tempest]

3.3 Multi-agent Shielding

...

4 Research Statement and Goals

...

4.1 Summary of Papers

...

Paper A: Shielded Reinforcement Learning for Hybrid systems

...

Paper B: Efficient Shield Synthesis via State-space Transformation

...

Paper C: Compositional Shielding and Reinforcement Learning for Multi-agent Systems

...

Paper D: UPPAAL COSHY - Automatic Synthesis of Compact Shields for Hybrid Systems

...

Paper A:

Shielded Reinforcement Learning for Hybrid Systems

Asger Horn Brorholt
Department of Computer Science
Aalborg University, Aalborg, Denmark

Peter Gjøl Jensen
Department of Computer Science
Aalborg University, Aalborg, Denmark

Kim Guldstrand Larsen
Department of Computer Science
Aalborg University, Aalborg, Denmark

Florian Lorber
Department of Computer Science
Aalborg University, Aalborg, Denmark

Christian Schilling
Department of Computer Science
Aalborg University, Aalborg, Denmark

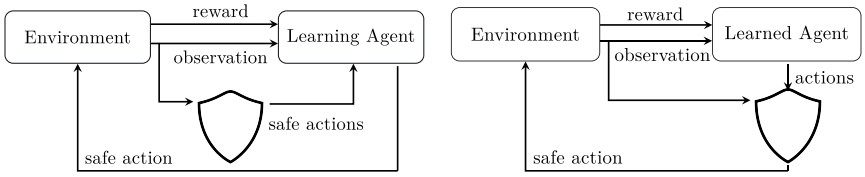
Abstract

Safe and optimal controller synthesis for switched-controlled hybrid systems, which combine differential equations and discrete changes of the system's state, is known to be intricately hard. Reinforcement learning has been leveraged to construct near-optimal controllers, but their behavior is not guaranteed to be safe, even when it is encouraged by reward engineering. One way of imposing safety to a learned controller is to use a emph{shield}, which is correct by design. However, obtaining a shield for non-linear and hybrid environments is itself intractable. In this paper, we propose the construction of a shield using the so-called emph{barbaric method}, where an approximate finite representation of an underlying partition-based two-player safety game is extracted via systematically picked samples of the true transition function. While hard safety guarantees are out of reach, we experimentally demonstrate strong statistical safety guarantees with a prototype implementation and uppaalstratego. Furthermore, we study the impact of the synthesized shield when applied as either a pre-shield (applied before learning a controller) or a post-shield (only applied after learning a controller). We experimentally demonstrate superiority of the pre-shielding approach. We apply our technique on a range of case studies, including two industrial examples, and further study post-optimization of the post-shielding approach.

1 Introduction

Digital controllers are key components of cyber-physical systems. Unfortunately, the algorithmic construction of controllers is intricate for any but the simplest systems [20], [21]. This motivates the usage of rl, which is a powerful machine-learning method applicable to systems with complex and stochastic dynamics [22].

However, while controllers obtained from rl provide near-optimal average-case performance, they do not provide guarantees about worst-case performance, which limits their application in many relevant but safety-critical domains, ranging from power converters to traffic control [23], [24]. A typical way to tackle this challenge is to integrate safety into the optimization objective via *reward shaping* during the learning phase, which punishes unsafe behavior [25]. This will make the controller more robust to a certain degree, but safety violations will still be possible, and the integration of safety into the optimization objective can reduce the performance, thus yielding a controller that is neither safe nor optimal.



(a) Pre-shield. (b) Post-shield.

Figure 4: Pre- and post-shielding in a reinforcement-learning setting.

A principled approach to obtain worst-case guarantees is to use a *shield* that restricts the available actions [26]. This makes it possible to construct correct-by-design and yet near-optimal controllers. Figure 4 depicts two ways of shielding RL agents: *pre-* and *post-shielding*. Pre-shielding is already applied during the learning phase, and the learning agent receives only safe actions to choose from. Post-shielding is only applied during deployment, where the trained agent is monitored and, if necessary, corrected. Such interventions to ensure safety interfere with the learned policy of the agent, potentially causing a loss in optimality.

In a nutshell, the algorithm to obtain a shield works as follows. First we compute a finite partitioning of the state space and approximate the transitions between the partitions. This results in a two-player safety game, and upon solving it, we obtain a strategy that represents the most permissive shield.

Cyber-physical systems exhibit behavior that is both continuous (the environment) and discrete (the control, and possibly the environment too). We are particularly interested in a class of systems we refer to as *hybrid Markov decision processes* (HMDPs). In short, these are control systems where the controller can choose an action in a periodic manner, to which the environment chooses a stochastic continuous trajectory modeled by a stochastic hybrid automaton [27]. While HMDPs represent many real-world systems, they are a rich extension of

hybrid automata, and thus their algorithmic analysis is intractable even under serious restrictions [28]. These complexity barriers unfortunately also carry over to the above problem of constructing a shield.

In this paper, we propose a new practical technique to automatically and robustly synthesize a shield for HMDPs. The intractability in the shield-synthesis algorithm is due to the rigorous computation of the transition relation in the abstract transition system, since that computation reduces to the (undecidable) reachability problem. Our key to get around this limitation is to approximate the transition relation through systematic sampling, in a way that is akin to the *barbaric method* (a term credited to Oded Maler [29], [30]).

We combine our technique with the tool UPPAAL STRATEGO to learn a shielded near-optimal controller, which we evaluate in a series of experiments on several models, including two real-world cases. In our experiments we also find that pre-shielding outperforms post-shielding. While the shield obtained through our technique is not guaranteed to be safe in general due to the approximation, we demonstrate that the controllers we obtain are statistically safe, and that a moderate number of samples is sufficient in practice.

Related work. Enforcing safety during RL by limiting the choices available to the agent is a known concept, which is for instance applied in the tool UPPAAL STRATEGO [31]. The term "shielding" was coined by Bloem et al. [26], who introduced special conditions on the enforcer like *shields with minimal interference* and *k-stabilizing shields* and later demonstrated shielding for RL agents [9], where they correct potentially unsafe actions chosen by the RL agent. Jansen et al. [32] introduced shielding in the context of RL for probabilistic systems. A concept similar to shielding has also been proposed for safe model predictive control [33], [34]. Carr et al. [35] show how to shield partially observable environments. In a related spirit, Maderbacher et al. start from a safe policy and switch to a learned policy if safe at run time [36].

(Pre-)Shielding requires a model of the environment in order to provide safety guarantees during learning. Orthogonal to shielding, several model-free approaches explore an rl environment in a *safer* way, but without any guarantees. Several works are based on barrier certificates and adversarial examples [37], [38] or Lyapunov functions [39]. Similarly, Berkenkamp et al. describe a method to provide a safe policy with high probability [40]. Chow et al. consider a relaxed version of safety based on expected cumulative cost [41]. In contrast to these model-free approaches, we assume a model of the environment, which allows us to safely synthesize a shield just from simulations before the learning phase. We believe that the assumption of a model, typically derived from first principles, is realistic, given that our formalism allows for probabilistic modeling of uncertainties. To the best of our knowledge, none of the above works can be used in practice for safe rl in the complex class of HMDPs.

Larsen et al. [42] used a set-based Euler method to overapproximate reachability for continuous systems. This method was used to obtain a safety strategy and a safe near-optimal controller. Contrary to that work, we apply both pre- and

post-shielding, and our method is applicable to more general hybrid systems. We employ state-space partitioning, which is common for control synthesis [43] and reachability analysis [44] and is also used in recent work on learning a safe controller for discrete stochastic systems in a teacher-learner framework [45]. Contemporary work by Badings et al. [46] also uses a finite state-space abstraction along with sample-based reachability estimation, to compute a reach-avoid controller. The method assumes linear dynamical systems with stochastic disturbances, to obtain upper and lower bounds on transition probabilities. In contrast, our method supports a very general hybrid simulation model, and provides a safety shield, which allows for further optimization of secondary objectives.

A special case of the HMDPs we consider is the class of stochastic hybrid systems (SHSs). Existing reachability approaches are based on state-space partitioning [47], [48], which we also employ in this work, or have a statistical angle [49]. We are not aware of any works that extended SHSs to HMDPs.

Outline. The remainder of the paper is structured as follows. In Section 2 we present the formalism we use. In Section 3 we present our synthesis method to obtain a safety strategy and explain how this strategy can be integrated into a shield. We demonstrate the performance of our pre- and post-shields in several cases in Section 4. Finally we conclude the paper in Section 5.

2 Euclidian and Hybrid Markov Decision Processes

In this section we introduce the system class we study in this paper: hybrid Markov decision processes (HMDPs). They combine Euclidean Markov decision processes and stochastic hybrid automata, which we introduce next. HMDPs model complex systems with continuous, discrete and stochastic dynamics.

2.1 Euclidean Markov Decision Processes

A Euclidean Markov decision process (EMDP) [50], [51] is a continuous-space extension of a Markov decision process (MDP). We recall its definition below.

A *Euclidean Markov decision process* of dimension k is a tuple $\mathcal{M} = (\mathcal{S}, s_0, Act, T, C, \mathcal{G})$ where

- $\mathcal{S} \subseteq \mathbb{R}^k$ is a bounded and closed part of k-dimensional Euclidean space,
- $s_0 \in \mathcal{S}$ is the initial state,
- Act is the finite set of actions,
- $T : \mathcal{S} \times Act \to (\mathcal{S} \to \mathbb{R}_{\geq 0})$ maps each state-action pair (s, a) to a probability density function over $\mathcal{S}$, i.e., we have $\int_{s' \in \mathcal{S}} T(s, a)(s')ds' = 1$,
- $C : \mathcal{S} \times Act \times \mathcal{S} \to \mathbb{R}$ is the cost function, and
- $G \subseteq \mathcal{S}$ is the set of goal states.

Example 1 *(Random Walk)* Figure 5 illustrates an EMDP of a (semi-)random walk on the state space $\mathcal{S} = [0, x_{max}] \times [0, t_{max}]$ (one-dimensional space plus time). The goal is to cross the $x = 1$ finishing line before $t = 1$. Two movement actions are available: fast and expensive (blue), or slow and cheap (brown). Both actions have uncertainty about the distance traveled and time taken. Given a state (x, t) and an action $a \in \{slow, fast\}$, the next-state density function $T((x, t), a)$ is a uniform distribution over the successor-state set $(x + d_x(a) \pm \epsilon) \times (t + d_t(a) \pm \epsilon)$, where $d_x(a)$ and $d_t(a)$ respectively represent the direction of movement in space and time given action a, while ϵ models the uncertainty.

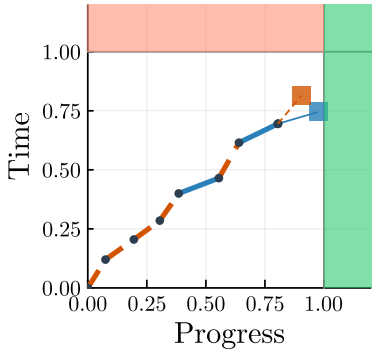


Figure 5: A random walk with action sequence *slow, slow, slow, slow, fast, slow, fast*.

A run π of an EMDP is an alternating sequence $s_0 a_0 s_1 a_1 \ldots$ of states and actions such that $T(s_i, a_i)(s_{i+1}) > 0$ for all $i \geq 0$. A (memoryless) strategy for an EMDP is a function $\sigma : \mathcal{S} \to (Act \to [0, 1])$, mapping a state to a probability distribution over Act. Given a strategy σ, the expected cost of reaching a goal state is defined as the solution to a Volterra integral equation as follows:

Let $\mathcal{M} = (\mathcal{S}, s_0, Act, T, C, \mathcal{G})$ be an EMDP and σ be a strategy. If a state s can reach the goal set $\mathcal{G}$, the *expected cost* is the solution to the following recursive equation:

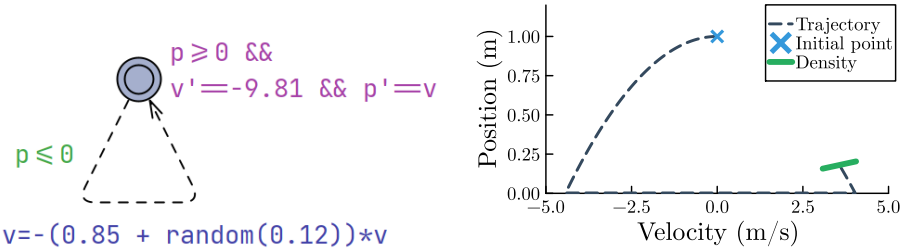
$$\mathbb{E}_\sigma^{\mathcal{M}}(s) = \begin{cases} 0 & \text{if } \ s \in \mathcal{G} \\ \sum_{a \in\ Act} \sigma(s)(a) \cdot \int_{s' \in \mathcal{S}} T(s, a)(s') \cdot (C(s, a, s') + \mathbb{E}_\sigma^{\mathcal{M}}(s'))\, ds' & \text{if } \ s \notin \mathcal{G} \end{cases} \quad (1)$$

A strategy σ^* is optimal if it minimizes $\mathbb{E}_\sigma^{\mathcal{M}}(s_0)$. We note that there exists an optimal strategy which is deterministic.

2.2 Stochastic Hybrid Systems

In an EMDP, the environment responds instantaneously to an action proposed by the agent according to the next-state density function T. In a more refined

view, the agent proposes actions with some period P, and the response of the environment is a stochastic, time-bounded trajectory (bounded by the period P) over the state space. For this response, we use a stochastic hybrid system (SHS) [27], [52], which allows the environment to interleave continuous evolution and discrete jumps.



(a) SHA for the bouncing ball. (b) State density after one bounce.

Figure 6: An SHA for the bouncing ball and a visualization after one bounce.

A *stochastic hybrid system* of dimension k is a tuple $H = (\mathcal{S}, F, \mu, \eta)$ where

- $\mathcal{S} \subseteq \mathbb{R}^k$ is a bounded and closed part of k-dimensional Euclidean space,
- $F : \mathbb{R}_{\geq 0} \times \mathcal{S} \to \mathcal{S}$ is a flow function describing the evolution of the continuous state with respect to time, typically represented by differential equations,
- $\mu : \mathcal{S} \to (\mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0})$ maps each state s to a delay density function $\mu(s)$ determining the time point for the next discrete jump, and
- $\eta : \mathcal{S} \to (\mathcal{S} \to \mathbb{R}_{\geq 0})$ maps each state s to a density function $\eta(s)$ determining the next state.

Example 2 *(Bouncing ball)* To represent an SHS, we use a stochastic hybrid automaton (SHA) [27], which we only introduce informally here. Figure 6 shows an SHA of a bouncing ball, which we use as a running example. Here the state of the ball is given by a pair (p, v) of continuous variables, where $p \in \mathbb{R}_{\geq 0}$ represents the current height (position) and $v \in \mathbb{R}$ represents the current velocity of the ball. Initially (not visible in the figure) the value of v is zero while p is picked randomly in $[7.0; 10.0]$. The behavior of the ball is defined by two differential equations: $v' = -9.81m/s^2$ describing the velocity of a falling object and $p' = v$ stating that the rate of change of the height is the current velocity. The invariant $p \geq 0$ expresses that the height is always nonnegative. The single transition of the automaton triggers when $p \leq 0$, i.e., when the ball hits the ground. In this case the velocity reverts direction and is subject to a random dampening effect (here "" draws a random number from $[0, 0.12]$ uniformly). The state density after one bounce is illustrated in

Figure 6. The SHA induces the following SHS, where δ denotes the Dirac delta distribution:

- $\mathcal{S} = [0, 10] \times [-14, 14]$,
- $F((p, v), t) = ((-9.81/2)t^2 + vt + p, -9.81t + v)$
- $\mu((p, v)) = \delta\Big(\Big(v + \sqrt{v^2 + 2 \cdot 9.81 \cdot p}\Big)/9.81\Big)$
- $\eta((p, v)) = \Big(p, v \cdot U_{[-0.97, -0.85]}\Big)$, with uniform distribution $U_{[l,u]}$ over $[l, u]$.

A timed run ρ of an SHS H with n jumps from an initial state density ι is a sequence $\rho = s_0 {s'}_0 t_0 s_1 {s'}_1 t_1 s_2 {s'}_2 ... t_{n-1} s_n {s'}_n$ respecting the constraints of H, where each $t_i \in \mathbb{R}_{\geq 0}$. The total duration of ρ is $\sum_{i=0}^{n-1} t_i$, and the density of ρ is $\iota(s_0) \cdot \prod_{i=0}^{n-1} \mu({s'}_i)(t_i) \cdot \eta(s_{i+1})({s'}_{i+1})$.

Given an initial state density ι and a time bound T, we denote by $\Delta^T_{H,\iota}$ the density function on $\mathcal{S}$ determining the state after a total delay of T, when starting in a state given by ι. The following recursive equation defines $\Delta^T_{H,\iota}$:[1]

$$\Delta^T_{H,\iota}(s') = \begin{cases} \iota(s') & \text{if } \ T = 0 \\ \int_s \iota(s) \cdot \int_{t \leq T} \mu(s)(t) \cdot \Delta^{T-t}_{H,\eta(F(t,s))}(s') \, dt \, ds & \text{if } \ T > 0 \end{cases} \tag{2}$$

For $T = 0$, the density of reaching s' is given by the initial state density function ι. For $T > 0$, reaching s' at T first requires to start from an initial state s (chosen according to ι), followed by some delay t (chosen according to $\mu(s)$), leaving the system in the state $F(t, s)$. From this state it remains to reach s' within time $(T - t)$ using $\eta(F(t, s))$ as initial state density.

2.3 Hybrid Markov Decision Processes

A hybrid Markov decision process (HMDP) is essentially an EMDP where the actions of the agent are selected according to some time period $P \in \mathbb{R}_{\geq 0}$, and where the next-state probability density function T is obtained from an SHS.

A *hybrid Markov decision process* is a tuple $HM = (\mathcal{S}, s_0, Act, P, N, H, C, \mathcal{G})$ where $\mathcal{S}, s_0, Act, C, \mathcal{G}$ are defined the same way as for an EMDP, and

- $P \in \mathbb{R}_{\geq 0}$ is the period of the agent,
- $N : \mathcal{S} \times Act \to (\mathcal{S} \to \mathbb{R}_{\geq 0})$ maps each state s and action a to a probability density function determining the immediate next state under a, and

[1] For SHS with an upper bound on the number of discrete jumps up to a given time bound T, the equation is well-defined.

- $H = (\mathcal{S}, F, \mu, \eta)$ is a stochastic hybrid system describing the responses of the environment.

An HMDP $HM = (\mathcal{S}, s_0, Act, P, N, H, C, \mathcal{G})$ induces the EMDP $M_{HM} = (\mathcal{S}, s_0, Act, T, C, \mathcal{G})$, where T is given by $T(s,a) = \Delta^P_{H,N(s,a)}$. That is, the next-state probability density function of M_{HM} is given by the state density after a delay of P (the period) according to H with initial state density N.

Example

(Hitting the bouncing ball). Figure 7 shows an HMDP extending the SHS of the bouncing ball from Figure 6. Now a player has to keep the ball bouncing indefinitely by periodically choosing between the actions *hit* and *nohit*,

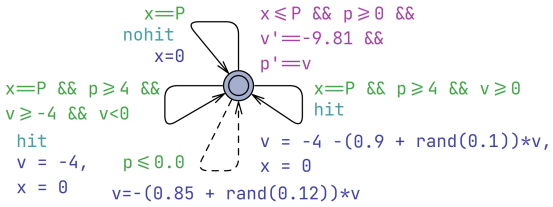
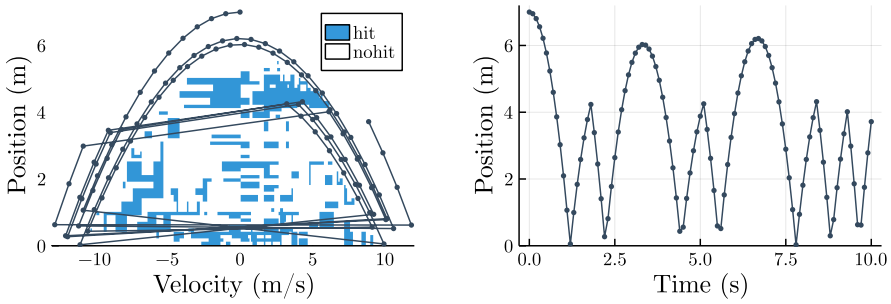


Figure 7: An HMDP for hitting a bouncing ball.

(three solid transitions). The period $P = 0.1$ is modeled by a clock x with suitable invariant, guards and updates. The top transition triggered by the *nohit* action has no effect on the state (but will have no cost). The *hit* action affects the state only if the height of the ball is at least 4m ($p \geq 4$). The left transition applies if the ball is falling with a speed not greater than -4m/s ($v \geq -4$) and accelerates to a velocity of -4m/s. The right transition applies if the ball is rising, and sets the velocity to a random value in $[-v-4, -0.9v-4]$. The bottom dashed transition represents the bounce of the ball as in Figure 6, which is part of the environment and outside the control of the agent.

A time-extended state (p, v, t) is in the goal set G if either $t \geq 120$ or $(p \leq 0.01 \wedge |v| \leq 1)$ (the ball is deemed dead). The cost (C) is 1 for the *hit* action and 0 for the *nohit* action, with an additional penalty of $1,000$ for transitions leading to a dead state. Figure 8 illustrates the near-optimal strategy σ^* obtained by the RL method implemented in UPPAAL STRATEGO and the prefix of a random run. The expected number of *hit* actions of σ^* within 120s is approximately 48.◁



(a) Strategy. (b) Example run for 10 seconds.

Figure 8: Near-optimal strategy learned by UPPAAL STRATEGO.

3 Safety, Partitioning, Synthesis and Shielding

3.1 Safety

In this section we are concerned with a strategy obtained for a given EMDP being *safe*. For example, a safety strategy for hitting the bouncing ball must ensure that the ball never reaches a dead state ($p \leq 0.01 \wedge |v| \leq 1$). In fact, although safety was encouraged by cost-tweaking, the strategy σ^* in Figure 8 is *not* safe. In the following we use symbolic techniques to synthesize safety strategies.

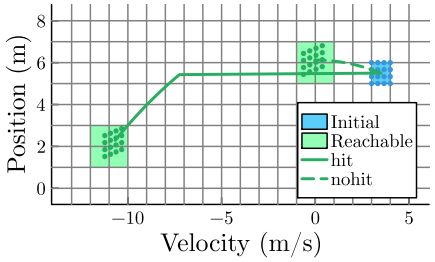
Let $\mathcal{M} = (\mathcal{S}, s_0, Act, T, C, \mathcal{G})$ be an EMDP. A safety property φ is a set of states $\varphi \subseteq \mathcal{S}$. A run $\pi = s_0 a_0 s_1 a_1 s_2 \ldots$ is safe with respect to φ if $s_i \in \varphi$ for all $i \geq 0$. Given a nondeterministic strategy $\sigma : \mathcal{S} \to 2^{Act}$, a run $\pi = s_0 a_0 s_1 a_1 s_2 \ldots$ of M is an outcome of σ if $a_i \in \sigma(s_i)$ for all i. We say that σ is a safety strategy with respect to φ if all runs that are outcomes of σ are safe.

3.2 Partitioning and Strategies

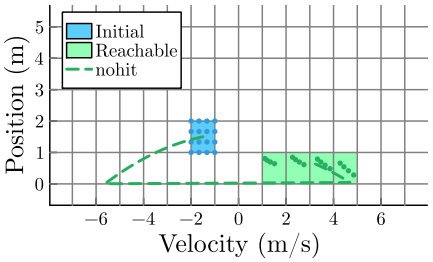
Given the infinite-state nature of the EMDP M, we will resort to finite partitioning (similar to [45]) of the state space in order to algorithmically synthesize nondeterministic safety strategies. Given a predefined granularity γ, we partition the state space into disjoint regions of equal size (we do this for simplicity; our method is independent of the particular choice of the partitioning). The partitioning along each dimension of $\mathcal{S}$ is a half-open interval belonging to the set $\mathcal{I}_\gamma = \{[k\gamma; k\gamma + \gamma[\mid k \in \mathbb{Z}\}$. For a bounded k-dimensional state space $\mathcal{S}$, $\mathcal{A} = \{\mu \in \mathcal{I}_\gamma^k \mid \mu \cap \mathcal{S} \neq \emptyset\}$ provides a finite partitioning of $\mathcal{S}$ with granularity γ. For each $s \in \mathcal{S}$ we denote by $[s]_A$ the unique region containing s.

Given an EMDP M, a partitioning A induces a finite labeled transition system $T_M^A = (\mathcal{A}, Act, \to)$, where

$$\mu \xrightarrow{a} \mu' \Leftrightarrow \exists s \in \mu.\, \exists s' \in \mu'.\, T(s,a)(s') > 0. \tag{3}$$



(a) Scenario where the ball is rising and high enough to be hit.



(b) Scenario where the ball is too low to be hit, but bounces off the ground.

Figure 9: State-space partitioning for Section . Starting in the blue region and depending on the action, the system can end up in the green regions within one time period, witnessed by simulations from 16 initial states.

Figure 9 shows a partitioning for the running example and displays some witnesses for transitions in the induced transition system.

Next, we view T_M^A as a 2-player game. For a region $\mu \in A$, Player 1 challenges with an action $a \in Act$. Player 2 responds with a region $\mu' \in A$ such that $\mu \xrightarrow{a} \mu'$.

Let $\varphi \subseteq \mathcal{S}$ be a safety property and A a partitioning. We denote by φ^A the set $\{\mu \in A \mid \mu \subseteq \varphi\}$. The set of safe regions with respect to φ is the maximal set of regions $\mathbb{S}_\varphi$ such that

$$\mathbb{S}_\varphi = \varphi^A \cap \{\mu \mid \exists a.\, \forall \mu'.\, \mu \xrightarrow{a} \mu' \implies \mu' \in \mathbb{S}_\varphi\}. \tag{4}$$

Given the finiteness of A and monotonicity of Equation 4, $\mathbb{S}_\varphi$ may be obtained in a finite number of interations using Tarski's fixed-point theorem [53].

A (nondeterministic) strategy for T_M^A is a function $\nu : A \to 2^{Act}$. The most permissive safety strategy ν_φ obtained from $\mathbb{S}_\varphi$ [54] is given by

$$\nu_\varphi(\mu) = \{a \mid \forall \mu'.\, \mu \xrightarrow{a} \mu' \implies \mu' \in \mathbb{S}_\varphi\}. \tag{5}$$

The following theorem states that we can obtain a safety strategy for the original EMDP M from a safety strategy ν for T_M^A.

Theorem 3 Given an EMDP M, safety property $\varphi \subseteq \mathcal{S}$ and partitioning A, if ν is a safety strategy for T_M^A, then $\sigma(s) = \nu([s]_A)$ is a safety strategy for M.

3.3 Approximating the 2-player Game

Let M be an EMDP and φ be a safety property. To algorithmically compute the set of safe regions $\mathbb{S}_\varphi$ for a given partitioning A, and subsequently the most permissive safety strategy ν_φ, the transition relation $\xrightarrow{a}$ needs to be a decidable predicate.

If M is derived from an HMDP $HM = (\mathcal{S}, s_0, Act, P, N, H, C, \mathcal{G})$, this requires decidability of the predicate $\Delta^P_{H,N(s,a)}(s') > 0$. Consider the bouncing ball from Section . The regions are of the form $\mu = \{(p, v) \mid l_p \leq p < u_p \wedge l_v \leq v < u_v\}$. For given regions μ, μ', the predicate $\mu \overset{nohit}{\rightarrow} \mu'$ is equivalent to the following first-order predicate over the reals (note that $F((p, v), t)$ is a pair of polynomials in p, v and t):[2]

$$\exists (p, v) \in \mu.\ F((p, v), P) \in \mu' \vee \exists \beta \in [0.85, 0.97].\ \exists t' \leq P.\ \exists v'.\ F((p, v), t') = (0, v') \wedge F((0, -\beta \cdot v'), P - t') \in \mu' \tag{6}$$

For this simple example, the validity of the formula can be decided [55], which may however require doubly exponential time [56], and worse, when considering nonlinear dynamics with, e.g., trigonometric functions, the problem becomes undecidable [57]. One can obtain a conservative answer via over-approximate reachability analysis [58] in Section 4 we compare to such an approach and demonstrate that, while effective, that approach also does not scale. This motivates to use an efficient and robust alternative. We propose to approximate the transition relation using equally spaced samples, which are simulated according to the SHS $\mathcal{H}$ underlying the given HMDP $\mathcal{HM}$.

Algorithm 1 describes how to compute an approximation $\mu \overset{a}{\underset{app}{\rightarrow}} \mu'$ of $\mu \overset{a}{\rightarrow} \mu'$. The algorithm draws from a finite set of n evenly distributed supporting points per dimension $app[\mu] = \{s_1, ..., s_{n^k}\} \subseteq \mu$ and simulates H for P time units. A region μ' is declared reachable from μ under action a if it is reached in at least one simulation. When stochasticity is involved in a simulation, additional care must be taken. The random variables can be considered an additional dimension to be sampled from; alternatively, a worst-case value can be used if available, such as the bouncing ball with the highest velocity damping. Figure 9 illustrates 16 ($n =$

Algorithm 1: Approximation of $\overset{a}{\rightarrow}$

Input: $\mu \in \mathcal{A}, a \in Act$
Output: $\mu \overset{a}{\underset{app}{\rightarrow}} \mu'$ iff $\mu' \in R$

```
R = ∅
for all s_i ∈ app[μ] do
    select s'_i ~ N(s_i, a)
    simulate H from s'_i for P time units
    let s''_i be the resulting state
    add [s''_i]_A to R
end for
```

[2] We assume that at most one bounce can take place within the period P.

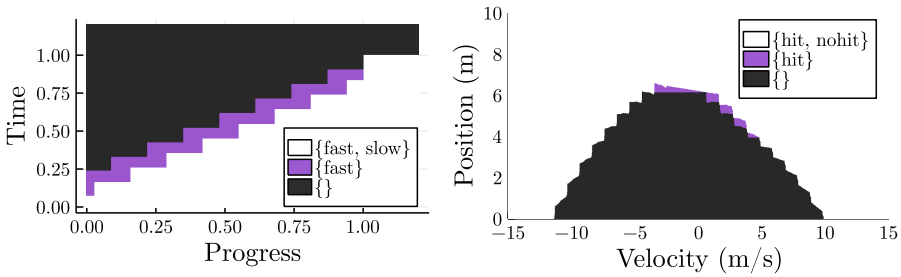


Figure 10: Synthesized nondeterministic strategies for random walk (left) and bouncing ball (right).

4) possible starting points for the bouncing ball together with most pessimistic outcomes, depending on the action taken.

The result $\xrightarrow[app]{a}$ is an underapproximation of the transition relation $\xrightarrow{a}$, with a corresponding transition system $\hat{T}_M^A = \left(\mathcal{A}, Act, \xrightarrow[app]{}\right)$. Thus if we compute a safety strategy ν from $\xrightarrow[app]{a}$, then the strategy $\sigma(s) = \nu([s]_\mathcal{A})$ from Theorem 3 is not necessarily safe. However, in Section 4 we will see that this strategy is statistically safe in practice. We attribute this to two reasons. 1) The underapproximation of $\xrightarrow[app]{a}$ can be made accurate. 2) Since $\xrightarrow{a}$ is defined over an abstraction, it is often robust against small approximation errors.

3.4 Shielding

As argued above, we can obtain the most permissive safety strategy ν_φ from $\xrightarrow[app]{a}$ over $\mathcal{A}$ and then use $\sigma_\varphi(s) = \nu_\varphi([s]_\mathcal{A})$ as an approximation of the most permissive safety strategy over the original HMDP. We can employ σ_φ to build a shield. As discussed in the introduction, we focus on two ways of shielding: *pre-shielding* and *post-shielding* (recall Figure 4). In pre-shielding, the shield is already active during the learning phase of the agent, which hence only trains on sets of safe actions. In post-shielding, the shield is only applied after the learning phase, and unsafe actions chosen by the agent are corrected (which is possibly detrimental to the performance of the agent).

Figure 10 shows examples of such strategies for the random walk (Example 1) and the bouncing ball. As can be seen, most regions of the state space are either unsafe (black) or both actions are safe (white). Only in a small area (purple) will the strategy enforce walking fast or hitting the ball, respectively. In the white area, the agent can learn the action that leads to the highest performance.

We use UPPAAL STRATEGO [31] to train a shielded agent based on σ_φ. The complete workflow of pre-shielding and learning is depicted in Figure 11. Starting from the EMDP, we partition the state space, obtain the transition system using Algorithm 1 and solve the game according to a safety property φ, as described above. The produced strategy is then conjoined with the original EMDP to form the shielded

EMDP, and reinforcement learning is used to produce a near-optimal deterministic strategy σ^*. This strategy can then be used in the real world, or get evaluated via statistical model checking. The only difference in the workflow in post-shielding is that the strategy σ_φ is not applied to the EMDP, but on top of the deterministic strategy σ^*.

4 Experiments

In this section we study our proposed approach with regard to different aspects of our shields. In addition to the random walk (Example 1) and bouncing ball (Section), we consider three benchmark cases:

- *Cruise control* [42], [59], [60]: A car is controlled to follow another car as closely as possible without crashing. Either car can accelerate, keep its speed, or decelerate freely, which makes finding a strategy challenging. This model was subject to several previous studies where a safety strategy was carefully designed, while our method can be directly applied without human effort.
- *DC-DC converter* [61]: This industrial DC-DC boost converter transforms input voltage of 10V to output voltage of 15V. The controller switches between storing energy in an inductor and releasing it. The output must stay in ± 0.5V around 15 V, and the amount of switching should be minimized.
- *Oil pump* [62]: In this industrial case, flow of oil into an accumulator is controlled to satisfy minimum and maximum volume constraints, given a consumption pattern that is piecewise-constant and repeats every 20 seconds. Since the exact consumption is unknown, a random perturbation is added to the reference value. To reduce wear, the volume should be kept low.

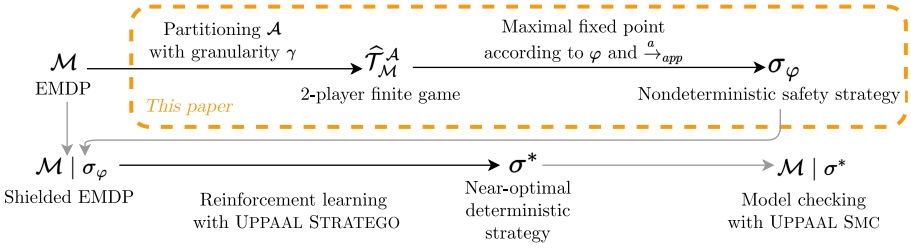
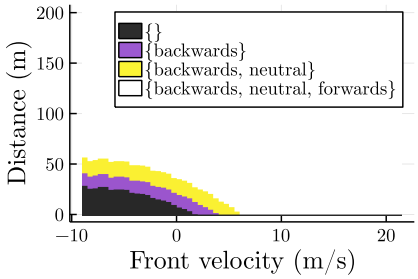
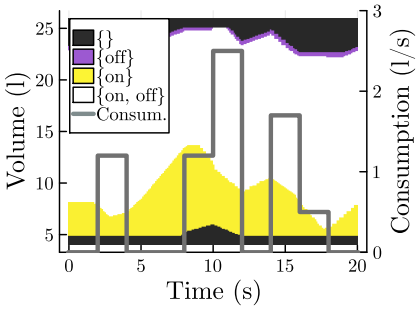


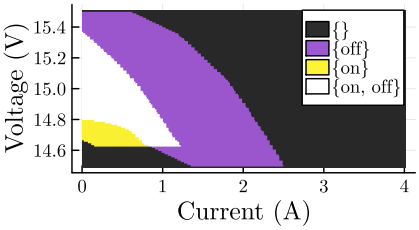
Figure 11: Complete method for pre-shielding and statistical model checking (SMC).



(a) Cruise control ($n = 4, \gamma = 0.5$) when the car's velocity is $0m/s$



(b) Oil pump ($n = 4, \gamma = 0.1$) when the pump is *on*. The periodic piecewise consumption pattern has been overlaid. Turning off the pump requires it to stay off for two seconds, which could cause an underflow in the yellow area. Conversely, the purple area shows the states where the pump *must* be turned off to avoid overflow. Since the pump is on in this projection, this can wait until the last moment.



(c) DC-DC boost converter ($n = 4$, $\gamma = 0.01$) when the output resistance is 30Ω.

Figure 12: Projected views of synthesized most permissive safety strategies.

Figure 12 shows the synthesized most permissive safety strategies. For instance, in Figure 12a we see the strategy for the cruise-control example when the controlled car is standing still. If the car in front is either close or reverses at high speed, the controlled car must also reverse (purple area). The yellow area shows states where it is safe to stand still but accelerating may lead to a collision.

We conduct four series of experiments to study different aspects of our approach.

The quality of our approximation of the transition relation $\xrightarrow[app]{a}$,

the computational performance of our approximation in comparison with a fully symbolic approach,

the performance in terms of reward and safety of the pre- and post-shields synthesized with our method, and

the potential of post-optimization for post-shielding.

All experiments are conducted on an AMD Ryzen 7 5700x with 33 GiB RAM. Our implementation is written in Julia, and we use UPPAAL STRATEGO [31] for learning and statistical model checking. The experiments are available online [63].

4.1 Quality of the Approximated Transition System

In the first experiment we statistically assess the approximation quality of $\xrightarrow[app]{a}$ wrt. the underlying infinite transition system. For varying granularity γ of $\mathcal{A}$ and numbers of supporting points n per dimension (see Section 3) we first compute $\xrightarrow[app]{a}$ with Algorithm 1. Then we uniformly sample 10^8 states s and compute their successor states s' under a random action a. Finally we count how often $[s]_{\mathcal{A}} \xrightarrow[app]{a} [s']_{\mathcal{A}}$ holds.

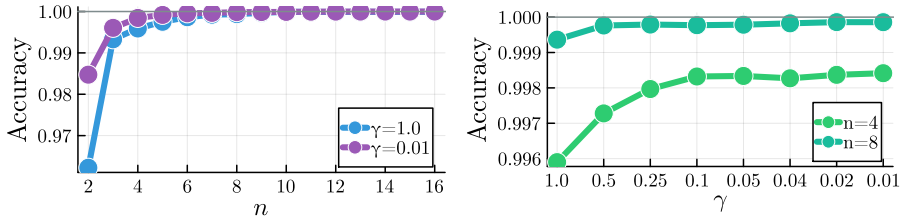


Figure 13: Accuracy of the approximation $\xrightarrow[app]{a}$ under different granularity γ and number of supporting points n per dimension.

Here we consider the bouncing-ball model, where we limit the domain to $p \in [0, 15], v \in [-15, 15]$. The results are shown in Figure 13. An increase in the number of supporting points n correlates with increased accuracy. For $\gamma \leq 1$, using $n = 3$ supporting points already yields accuracy above 99%. Finer partition granularity γ increases accuracy, but less so compared to increasing n.

4.2 Comparison with Fully Symbolic Approach

As described in Section 3, as an alternative to Algorithm 1 one can use a reachability algorithm to obtain an overapproximation of the transition relation $\xrightarrow{a}$. Here we analyze the performance of such an approach based on the reachability tool JuliaReach [64]. Given a set of initial states of a hybrid automaton where we have substituted probabilities by nondeterminism, JuliaReach can compute an overapproximation of the successor states. In JuliaReach, we select the reachability algorithm from [65]. This algorithm uses time discretization, which requires a small time step to give precise answers. This makes the approach expensive. For instance, for the bouncing-ball system, the time period is $P = 0.1$ time units, and a time step of 0.001 time units is required, which corresponds to 100 iterations.

The shield obtained with JuliaReach is safe by construction. To assess the safety of the shield obtained with Algorithm 1, we choose an agent that selects an action at random and let it act under the post-shield for 10^6 episodes. (We use a random agent because a learned agent may have learned to act safely most of the time and thus not challenge the shield as much.) If no safety violation was detected, we compute 99% confidence intervals for the statistical safety.

γ	$\underset{app}{\overset{a}{\rightarrow}}$ method	Parameters	Time	Probability safe
0.02	Algorithm 1	$n = 2$	1m 50s	unsafe run found
0.02		$n = 4$	2m 14s	$[99.9999\%; 100\%]$
0.02		$n = 8$	4m 02s	$[99.9999\%; 100\%]$
0.02		$n = 16$	11m 03s	$[99.9999\%; 100\%]$
0.01	Algorithm 1	$n = 2$	16m 49s	$[99.9999\%; 100\%]$
0.01		$n = 4$	19m 00s	$[99.9999\%; 100\%]$
0.01		$n = 8$	27m 21s	$[99.9999\%; 100\%]$
0.01		$n = 16$	56m 32s	$[99.9999\%; 100\%]$
0.01	JuliaReach	time step 0.002	24h 30m	considers s_0 unsafe
0.01		time step 0.001	41h 05m	safe by construction

Table 2: Synthesis results for the bouncing ball under varying granularity (γ) and supporting points (n) using Algorithm 1 (top) and two choices of the time-step parameter using JuliaReach (bottom). The safety probability is computed for a 99% confidence interval. $\gamma = 0.02$ corresponds to $9.0 \cdot 10^5$ partitions, and $\gamma = 0.01$ quadruples the number of partitions to $3.6 \cdot 10^6$.

We consider again the bouncing-ball model. JuliaReach requires a low partition granularity $\gamma = 0.01$; for $\gamma = 0.02$ it cannot prove that a safety strategy exists, which may be due to conservatism, while our method is able to synthesize a shield that, for $n \geq 4$, is statistically safe. Table 2 shows the results obtained from the two approaches. In addition, the reachability algorithm uses time discretization, and a small time step is required to find a safety strategy.

We remark that the bouncing-ball model has linear dynamics, for which reachability analysis is relatively efficient compared to nonlinear dynamics, and thus this model works in favor of the symbolic method. However, the hybrid nature of the model and the large number of queries (one for each partition-action pair) still make the symbolic approach expensive. Considering the case $\gamma = 0.01$ and $n = 4$, our method can synthesize a strategy in 19 minutes, while the approach based on JuliaReach takes 41 hours.

Figure 14 visualizes the two strategies and shows how the two approaches largely agree on the synthesized shield – but also the slightly more pessimistic nature of the transition relation computed with JuliaReach.

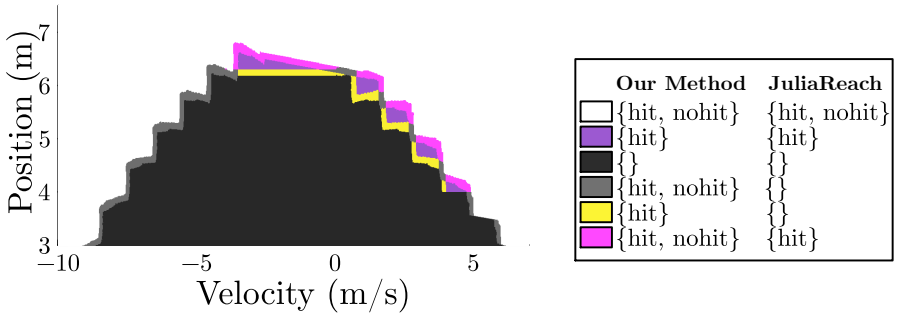


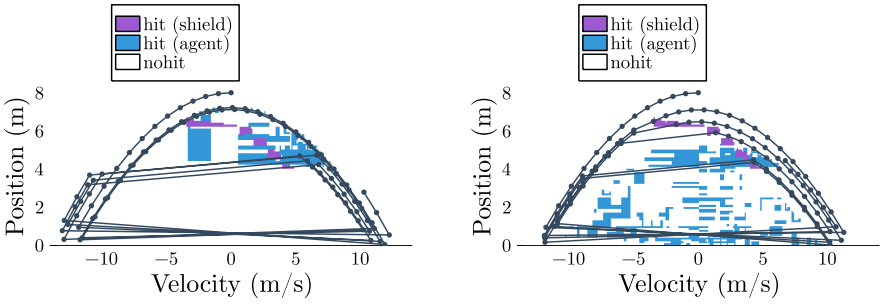
Figure 14: Superimposed strategies of our method and JULIAREACH.

4.3 Evaluation of Pre- and Post-shields

In the next series of experiments, we evaluate the full method of obtaining a shielded agent. The first step is to approximate $\xrightarrow[app]{a}$ using Algorithm 1 and extract the most permissive safety strategy σ_φ to be used as a shield. For the second step we have two options: pre- or post-shielding. Recall from Figure 4 that a pre-shield is applied to the agent during training while a post-shield is applied after training.

In the case of the bouncing ball, the post-shielded agent's strategy is shown in Figure 15. It consists of the unshielded strategy from Figure 8 plus the purple regions of the safety strategy in Figure 10. Correspondingly, Figure 15 shows the pre-shielded strategy, which is significantly simpler because it does not explore unsafe regions of the state space. This also leads to faster convergence.

Tables 3—5 report the same data as in Table 2 for the other models. Overall, we see a similar trend in all tables. For a low number of supporting points (say, $n = 3$) we can obtain a safety strategy that we find to be statistically safe. In all cases, no unsafe run was detected in the statistical evaluation. The synthesis time varies depending on the model and is generally feasible. The longest computation times



(a) Pre-shield. (b) Post-shield.

Figure 15: Learned shielded strategies for the bouncing ball.

are seen for the oil-pump example, which has the most dimensions. Still, times are well below JULIAREACH for the comparatively simple bouncing ball.

γ	n	Time	Probability safe
1	2	1m 50s	Considers s_0 unsafe
0.5	2	13m 16s	$[99.9995\%; 100\%]$
0.5	3	23m 03s	$[99.9995\%; 100\%]$
0.5	4	35m 55s	$[99.9995\%; 100\%]$

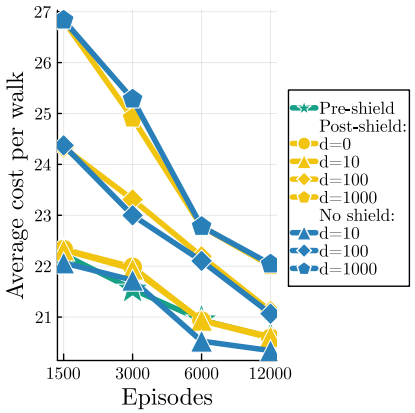
Table 3: Cruise control. $\gamma = 1$ corresponds to 1.9×10^5 partitions, and $\gamma = 0.5$ to 1.5×10^6.

γ	n	Time	Probability safe
0.2	2	3m 07s	considers s_0 unsafe
0.1	2	32m 15s	$[99.9995\%; 100\%]$
0.1	3	1h 37m	$[99.9995\%; 100\%]$
0.1	4	5h 23m	$[99.9995\%; 100\%]$

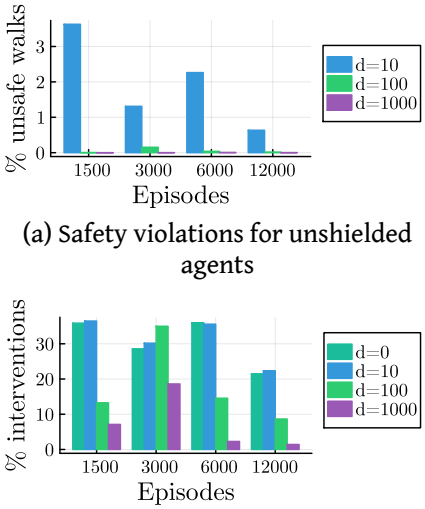
Table 4: Oil pump. $\gamma = 0.2$ corresponds to 2.8×10^5 partitions, and $\gamma = 0.1$ to 1.1×10^6.

γ	n	Time	Probability safe
0.05	2	41s	$[99.9995\%; 100\%]$
0.05	3	1m 50s	considers s_0 unsafe
0.05	4	3m 30s	considers s_0 unsafe
0.02	2	3m 43s	$[99.9995\%; 100\%]$
0.02	3	8m 59s	$[99.9995\%; 100\%]$
0.02	4	18m 11s	$[99.9995\%; 100\%]$
0.01	2	15m 48s	$[99.9995\%; 100\%]$
0.01	3	38m 26s	$[99.9995\%; 100\%]$
0.01	4	1h 19m	$[99.9995\%; 100\%]$

Table 5: DC-DC boost converter. $\gamma = 0.05$ corresponds to $3.1 \cdot 10^5$ partitions, $\gamma = 0.02$ to $1.7 \cdot 10^6$ and $\gamma = 0.01$ to $7.0 \cdot 10^6$.



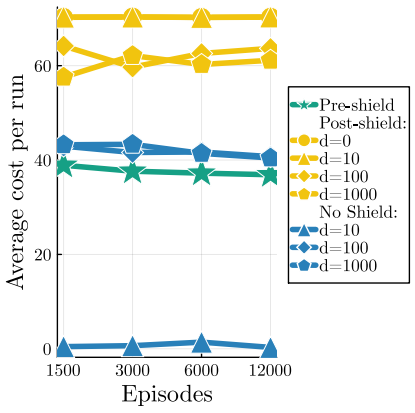
(b) Average cost per run.



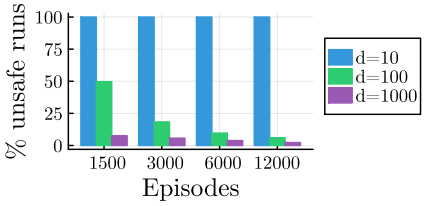
(a) Safety violations for unshielded agents

(c) Interventions for post-shielded agents.

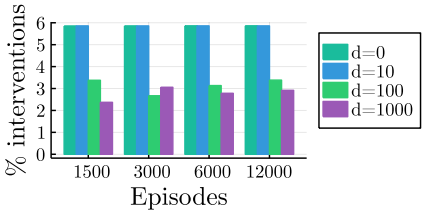
Figure 16: Results of shielding the random walk using $\gamma = 0.005$.



(b) Average *hit* actions per 120s.

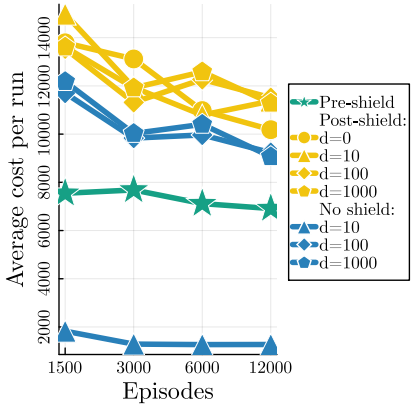


(a) Safety violations for unshielded agents.

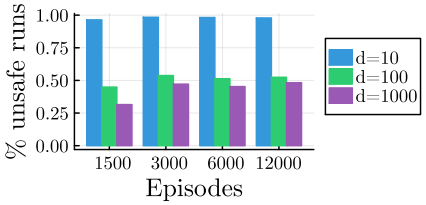


(c) Interventions for post-shielded agents.

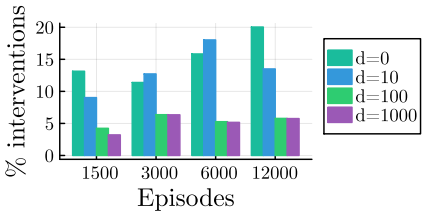
Figure 17: Results of shielding the bouncing ball using $n = 16$, $\gamma = 0.01$.



(b) Accumulated distance per 120s.

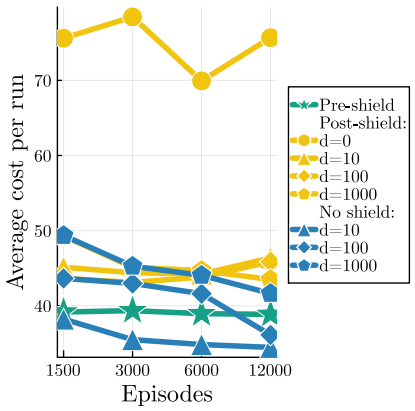


(a) Safety violations for unshielded agents.

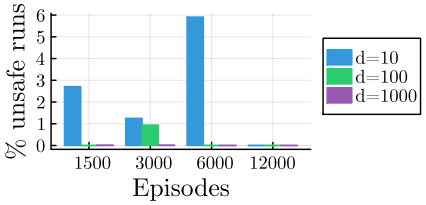


(c) Interventions for post-shielded agents.

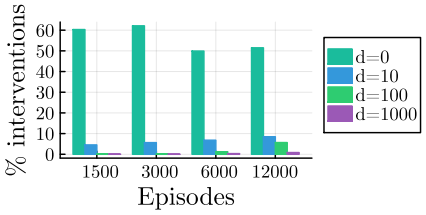
Figure 18: Results of shielding the cruise control using $n = 4, \gamma = 0.5$.



(b) Accumulated error plus number of switches per 120s.

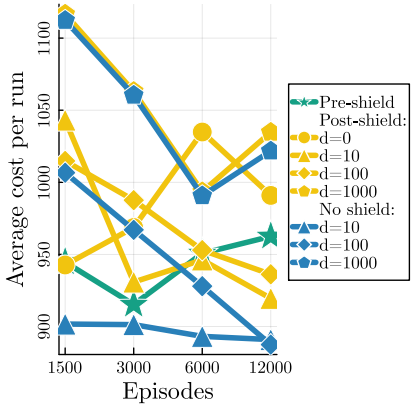


(a) Safety violations for unshielded agents.

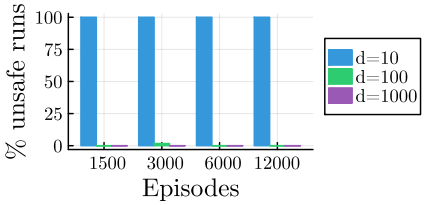


(c) Interventions for post-shielded agents.

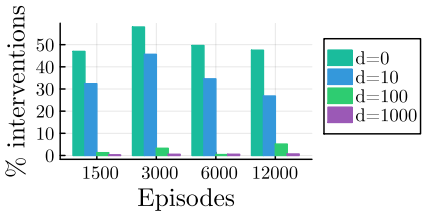
Figure 19: Results of shielding the DC-DC boost converter using $n = 4, \gamma = 0.01$.



(b) Accumulated oil volume per 120s.



(a) Safety violations for unshielded agents.



(c) Interventions for post-shielded agents.

Figure 20: Results of shielding the oil pump using $n = 4, \gamma = 0.1$.

Next, we compare our method to other options to make an agent safe(r). As the baseline, we use the classic RL approach, where safety is encouraged using reward shaping. We experiment with a deterrence $d \in \{0, 10, 100, 1000\}$ (negative reward) as a penalty for safety violations for the learning agent. Note that this penalty is only applied during training, and not included in the total cost when we evaluate the agent below. As the second option, we use a post-shielded agent, to which the deterrence also applies. The third option is a pre-shielded agent. In all cases, training and evaluation is repeated 10 times, and the mean value is reported. The evaluation is based on 1000 traces for each repetition.

Figures 16, to 20 report the results for the different models. Each subfigure shows the following content: (a) shows the average cost of the final agent, (b) shows the amount of safety violations of the unshielded agents and (c) shows the number of times the post-shielded agents were intervened by the shield.

Overall, we observe similar tendencies. The unshielded agent has lowest average cost at deployment time under low deterrence, but it also violates safety. Higher deterrence values improve safety, but do not guarantee it.

The pre-shielded agents outperform the post-shielded agents. This is because they learn a near-optimal strategy subject to the shield, while the post-shielded agents may be based on a learned unsafe strategy that contradicts the shield, and thus the shield interference can be more detrimental.

Configuration	Cost	Interventions		
Baseline with uniform random choice	11371	13.50		
Minimizing interventions	11791	$(+3.7\%)$	11.43	(-15.3%)
Minimizing cost	10768	(-5.3%)	17.43	$(+29.1\%)$
Agent preference	11493	(-1.1%)	14.55	$(+7.8\%)$
Pre-shielded agent	6912	(-39.2%)	–	–

Table 6: Change of post-optimization relative to the uniform-choice strategy. The strategy was trained for $12,000$ episodes with $d = 10$ and post-optimized for $4,000$ episodes. Performance of the pre-shielded agent is included for comparison, but interventions are not applicable (because the shield was in place during training).

4.4 Post-Shielding Optimization

When a post-shield intervenes, more than one action may be valid. This leaves room for further optimization, for which we can use UPPAAL STRATEGO. Compared to a uniform baseline, we assess three ways to resolve nondeterminism: 1) minimizing interventions, 2) minimizing cost and 3) at the preference of the shielded agent (the so-called Q-value [66]).

Table 6 shows the effect of post-optimization on the cost and the number of interventions for the cruise-control example. Notably, cost is only marginally affected, but the number of shield interventions can get significantly higher. The pre-shielded agent has lower cost than all post-optimized alternatives.

5 Conclusion

We presented a practical approach to synthesize a near-optimal safety strategy via finite (2-player) abstractions of hybrid Markov decision processes, which are systems of complex probabilistic and hybrid nature. In particular, we deploy a simulation-based technique for inferring the 2-player abstraction, from which a safety shield can then be constructed. We show with high statistical confidence that the shields avoid unsafe outcomes in the case studies, and are significantly faster to construct than when deploying symbolic techniques for computing a correct 2-player abstraction. In particular, our method demonstrates statistical safety on several case studies, two of which are industrial. Furthermore, we study the difference between pre- and post-shielding, reward engineering and a post-shielding optimization. In general, we observe that reward engineering is insufficient to enforce safety, and secondarily observe that pre-shielding provides better controller performance compared to post-shielding.

Future work includes applying the method to more complex systems, and using formal methods to verify the resulting safety strategies, maybe based on [67].

Paper B:

Efficient Shield Synthesis via State-space Transformation

Asger Horn Brorholt
Department of Computer Science
Aalborg University, Aalborg, Denmark

Andreas Holck Høeg-Petersen
Department of Computer Science
Aalborg University, Copenhagen, Denmark

Kim Guldstrand Larsen
Department of Computer Science
Aalborg University, Aalborg, Denmark

Christian Schilling
Department of Computer Science
Aalborg University, Aalborg, Denmark

Abstract

We consider the problem of synthesizing safety strategies for control systems, also known as shields. Since the state space is infinite, shields are typically computed over a finite-state abstraction, with the most common abstraction being a rectangular grid. However, for many systems, such a grid does not align well with the safety property or the system dynamics. That is why a coarse grid is rarely sufficient, but a fine grid is typically computationally infeasible to obtain. In this paper, we show that appropriate state-space transformations can still allow to use a coarse grid at almost no computational overhead. We demonstrate in three case studies that our transformation-based synthesis outperforms a standard synthesis by several orders of magnitude. In the first two case studies, we use domain knowledge to select a suitable transformation. In the third case study, we instead report on results in engineering a transformation without domain knowledge.

1 Introduction

Cyber-physical systems are ubiquitous in the modern world. A key component in these systems is the digital controller. Many of these systems are safety critical, which motivates the use of methods for the automatic construction of controllers. Unfortunately, this problem is intricate for any but the simplest systems [68], [69].

Two main methods have emerged. The first method is *reinforcement learning* (RL) [22], which provides convergence to an optimal solution. However, the solution lacks formal guarantees about safety. The second method is *reactive synthesis*, which constructs a nondeterministic control strategy that is guaranteed to be safe. However, the solution lacks optimality for secondary objectives.

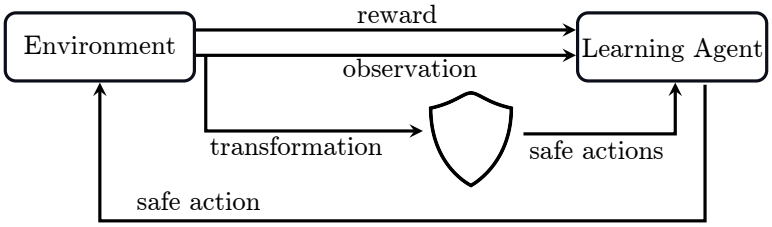


Figure 21: Reinforcement learning under a shield with a transformation f.

Due to their complementary strengths and drawbacks, these two methods have been successfully combined in the framework of *shielding* [7], [8] (cf. Figure 21). Through reactive synthesis, one first computes a nondeterministic control strategy called a *shield*, which is then integrated in the learning process to prevent unsafe actions. This way, safety is guaranteed and, at the same time, RL can still provide optimality with respect to the secondary objectives.

In this work, we focus on the first step: synthesis of a shield. For infinite state spaces, we employ an abstraction technique based on state-space partitioning, where we consider the common case of a *hyperrectangular grid* [70], [71]. This grid induces a finite-state two-player game, from which we can then construct the most permissive shield with standard algorithms. The downside of a grid-based approach is that a grid often does not align well with the dynamics of the system, which causes the game to have many transitions and thus results in an overly conservative control strategy. To counter this effect, one can refine the partition, but this has severe computational cost and quickly becomes infeasible.

The key insight we follow in this work is that a *state-space transformation* can yield a new state space where the grid aligns much better with the system dynamics. As we show in three case studies, the transformation allows to reduce the grid significantly, often by several orders of magnitude. In extreme cases, no grid may exist for synthesizing a control strategy in the original state space, while a simple grid suffices in the transformed state space.

We show that our transformation-based shield synthesis is sound, i.e., the guarantees of the shield transfer to the original system. Moreover, our experiments demonstrate that the transformation does not reduce the performance of the final controller (in fact, the performance slightly increases).

Our implementation is based on our previous work on sampling-based shield synthesis [10]. The present work integrates nicely with such a sampling-based method, but also generalizes to set-based methods.

For the first two case studies, we employ domain knowledge to derive a suitable transformation. For the third case study, we instead apply a new heuristic method to synthesize a suitable transformation.

1.1 Related Work

Abstraction-based controller synthesis is a popular approach that automatically constructs a controller from a system and a specification [70], [71]. The continuous dynamics are discretized and abstracted by a symbolic transition system, for which then a controller is found. The most common abstraction is a regular hyperrectangular (or even hypercubic) grid. The success of this approach depends on the choice of the grid cells' size. If too small, the state space becomes intractably large, while if too large, the abstraction becomes imprecise and a controller may not be found. While the cell size can be optimized [72], a fixed-sized grid is often bound to fail. Instead, several works employ multiple layers of different-sized grids in the hope that coarser cells can be used most of the time but higher precision can still be used when necessary [73], [74]. In this paper, we follow an orthogonal approach. We argue that a hyperrectangular grid is often inappropriate to capture the system dynamics and specification. Nevertheless, we demonstrate that, often, a (coarse) grid is still sufficient when applied in a different, more suitable state space.

In this work, we do not synthesize a full controller but only a nondeterministic safety strategy, which is known as a shield. We then employ this shield as a guardrail in reinforcement learning (cf. Figure 21) to limit the choices available to the agent so that the specification is guaranteed. This is a known concept, which is for instance applied in the tool UPPAAL STRATEGO [31] and was popularized by Bloem et al. [8], [9], [75]. A similar concept is safe model predictive control [33], [34].

Our motivation for applying a state-space transformation is to better align with a grid, and ultimately to make the synthesis more scalable. In that sense, our work shares the goal with some other influential concepts. In abstract interpretation, the transformation is the abstraction function and its inverse is the concretization function, which together form a Galois connection [76]. In our approach, the grid introduces an abstraction, but our additional transformation preserves information unless it is not injective. Another related concept is model order reduction, where a system is transformed to another system of lower dimensionality to simplify the analysis [77]. This reduction is typically approximate, which loses any formal guarantees. However, approaches based on (probabilistic) bisimulation [78] still allow to preserve a subspace and transfer the results to the original system. These approaches, also called lumpability, use linear transformations and have

been successfully applied to Markov chains [79], differential equations [80], and quantum circuits [81]. In contrast, while we do not put any restrictions on our transformations, we advocate for injective transformations in our context; this is because we also need to compute the preimage under the transformation, which otherwise incur additional approximation error.

Outline.
The remainder of the paper is structured as follows. In Section 2, we recall central concepts underlying partition-based shielding. In Section 3, we discuss how state-space transformations can be used for shield synthesis over grid-based partitions. In Section 4, we show experimental results in three case studies. Finally, we conclude the paper in Section 5.

2 Preliminaries

Intervals.
Given bounds $\ell, u \in \mathbb{R}$ with $\ell \leq u$, we write $\mathcal{I} = [\ell; u[\subseteq \mathbb{R}$ for the corresponding (half-open) *interval* with *diameter* $u - \ell$.

Set extension.
Given a function $f : S \to T$, the *set extension* is $f : 2^S \to 2^T$ with $f(X) = \bigcup_{s \in X} \{f(s)\}$ for any subset $X \subseteq S$. The extension generalizes to functions with further arguments, e.g., $g(X, y) = \bigcup_{s \in X} \{g(s, y)\}$. Set extension is monotonic, i.e., $f(X) \subseteq f(X')$ whenever $X \subseteq X'$.

Control systems.
In this work, we consider discrete-time control systems. Formally, a control system (S, Act, δ) is characterized by a bounded d-dimensional state space $S \subseteq \mathbb{R}^d$, a finite set of (control) actions Act, and a *successor function* $\delta : S \times Act \to 2^S$, which maps a *state* $s \in S$ and a *control action* $a \in Act$ to a set of successor states (i.e., δ may be nondeterministic). Often, δ is the solution of an underlying continuous-time system, measured after a fixed control period, as exemplified next.

Example 1 Consider a bivariate harmonic oscillator over the state space $S = [-2; 2[\times [-2; 2[\subseteq \mathbb{R}^2$, whose vector field is shown in Figure 22a. The continuous-time dynamics are given by the following system of differential equations: $\dot{s}(t) = As(t)$, where $A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$. The solution is $s(t) = e^{At} s_0$ for some initial state s_0. For this system, we only have a single (dummy) control action, $Act = \{a\}$. Fixing the control period $t = 1.2$ yields the discrete-time system (S, Act, δ) where $\delta(s, a) \approx \begin{pmatrix} 0.36 & 0.93 \\ -0.93 & 0.36 \end{pmatrix} s$.

Partitioning.

A *partition* $\mathcal{G} \subseteq 2^S$ of S is a set of pairwise disjoint sets of states (i.e., $\forall C_1 \neq C_2 \in \mathcal{G}.\ C_1 \cap C_2 = \emptyset$) whose union is S (i.e., $S = \bigcup_{C \in \mathcal{G}} C$). We call the elements C of $\mathcal{G}$ *cells*. For a state $s \in S$, $[s]_\mathcal{G}$ is the unique cell C such that $s \in C$. We furthermore define two helper functions $\lfloor \cdot \rfloor_\mathcal{G} : 2^S \to 2^\mathcal{G}$ and $\lceil \cdot \rceil_\mathcal{G} : 2^S \to 2^\mathcal{G}$ to under- and over-approximate a set of states $X \subseteq S$ with cells: $\lfloor X \rfloor_\mathcal{G} = \{C \in \mathcal{G} \mid C \subseteq X\}$ maps X to all its cells that are contained in X, and $\lceil X \rceil_\mathcal{G} = \{C \in \mathcal{G} \mid C \cap X \neq \emptyset\}$ maps X to all cells that intersect with X.

A cell C is *axis-aligned* if there exist intervals $\mathcal{I}_1, ..., \mathcal{I}_d$ such that $C = \mathcal{I}_1 \times ... \times \mathcal{I}_d$. A partition $\mathcal{G}$ is axis-aligned if all cells are axis-aligned. Moreover, $\mathcal{G}$ is a *regular grid* if for any two cells $C_1 \neq C_2$ in $\mathcal{G}$ and any dimension $i = 1, ..., d$, the diameters in dimension i are identical. In what follows, we consider axis-aligned regular grid partitions, or *grids* for short. Grids enjoy properties such as easy representation by just storing the bounds and diameters.

Strategies and safety.

Given a control system (S, Act, δ), a *strategy* $\sigma : S \to 2^{Act}$ maps a state s to a set of (allowed) actions a. (In the special case where $\sigma : S \to Act$ uniquely determines the next action a, we call σ a *controller*.) A sequence $\xi = s_0 a_0 s_1 a_1 ...$ is a *trajectory* of σ if $a_i \in \sigma(s_i)$ and $s_{i+1} \in \delta(s_i, a_i)$ for all i. A safety property $\varphi \subseteq S$ is characterized by the set of safe states. We call σ^X a *safety strategy*, or *shield*, with respect to a set X if all trajectories starting from any initial state $s_0 \in X$ are safe, i.e., only visit safe states. We often omit the set X.

In general, a safety strategy for infinite state spaces S cannot be effectively computed. The typical mitigation is to instead compute a safety strategy for a finite-state abstraction. One common such abstraction is a grid of finitely many cells. The grid induces a two-player game. Given a cell C, Player 1 challenges with an action $a \in Act$. Player 2 responds with a cell C' such that $C \xrightarrow{a} C'$. Player 1 wins if the game continues indefinitely, and Player 2 wins if $C' \not\subseteq \varphi$. Solving this game yields a safety strategy over cells, which then induces a safety strategy over the (concrete) states in S that uses the same behavior for all states in the same cell. We formalize this idea next.

Labeled transition system.

Given a control system (S, Act, δ), a grid $\mathcal{G} \subseteq 2^S$ induces a finite labeled transition system $(\mathcal{G}, Act, \to)$ that connects cells via control actions if they can be reached in one step:

$$C \xrightarrow{a} C' \Leftrightarrow \exists s \in C.\ \delta(s, a) \cap C' \neq \emptyset \tag{7}$$

Grid extension.

Given a grid $\mathcal{G}$ and a safety property φ, the set of *controllable cells*, or simply *safe cells*, is the maximal set of cells $\mathcal{C}_\varphi$ such that

$$\mathcal{C}_\varphi = \lfloor \varphi \rfloor_\mathcal{G} \cap \{C \in \mathcal{G} \mid \exists a \in Act.\, \forall C'.\, C \xrightarrow{a} C' \implies C' \in \mathcal{C}_\varphi\} \tag{8}$$

It is straightforward to compute $\mathcal{C}_\varphi$ with a finite fixpoint iteration. If $\mathcal{C}_\varphi$ is nonempty, there exists a safety strategy $\sigma^\mathcal{G} : \mathcal{G} \to 2^{Act}$ at the level of cells (instead of concrete states) with respect to the set $\mathcal{C}_\varphi$, where the most permissive such strategy [54] is

$$\sigma^\mathcal{G}(C) = \{a \in Act \mid \forall C'.\, C \xrightarrow{a} C' \implies C' \in \mathcal{C}_\varphi\} \tag{9}$$

A safety strategy $\sigma^\mathcal{G}$ over the grid $\mathcal{G}$ induces the safety strategy $\sigma^X(s) = \sigma^\mathcal{G}([s]_\mathcal{G})$ over the original state space S, with respect to the set $X = \bigcup_{C \in \mathcal{C}_\varphi} C$. The converse does not hold, i.e., a safety strategy may exist over S but not over $\mathcal{G}$, because the grid introduces an abstraction, as demonstrated next.

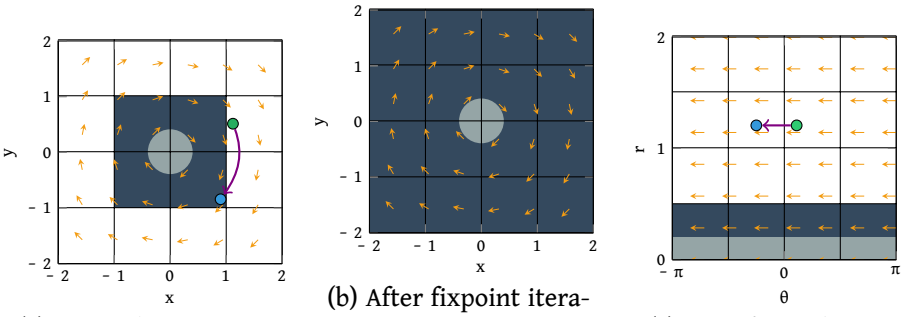
Lemma 2 Let (S, Act, δ) be a control system, $\varphi \subseteq S$ be a safety property, and $\mathcal{G} \subseteq 2^S$ be a partition. If $\sigma^\mathcal{G}$ is a safety strategy over cells, then $\sigma^X(s) = \sigma^\mathcal{G}([s]_\mathcal{G})$ is a safety strategy over states $s \in X \subseteq S$, where $X = \bigcup_{C \in \mathcal{C}_\varphi} C$.

Recall that a grid is an abstraction. The precision of this abstraction is controlled by the size of the grid cells, which we also refer to as the granularity.

Example 3 Consider again the harmonic oscillator from Example 1. To add a safety constraint, we place a disc-shaped obstacle, i.e., a set of states $O \subseteq S$, in the center. Since this system only has a dummy action a, there is a unique strategy σ that always selects this action. Clearly, σ is safe for all states that do not intersect with the obstacle O because all trajectories circle the origin.

With a rectangular grid in the x/y state space (with cell diameter 1 in each dimension), we face two fundamental problems. The first problem is that, in order to obtain a tight approximation of a disc with a rectangular grid, one requires a fine-grained partition. Thus, precision comes with a significant computational overhead. Recall that the cells that are initially marked unsafe are given by $\lceil O \rceil_\mathcal{G}$, which are drawn black in Figure 22a.

The second problem is similar in nature, but refers to the system dynamics instead. Since the trajectories of most systems do not travel parallel to the coordinate axes, a rectangular grid cannot capture the successor relation (and, hence, the required decision boundaries for the strategy) well. Consider the state highlighted in green in Figure 22a. Its trajectory leads to an unsafe cell, witnessing that its own cell is also unsafe. By iteratively applying this argument, the fixpoint is $\mathcal{C}_\varphi = \emptyset$, i.e., no cell is considered safe (Figure 22b). This means that, for the chosen grid granularity, no safety strategy $\sigma^\mathcal{G}$ at the level of cells exists.



(a) Original state space.

(b) After fixpoint iteration.

(c) Transformed state space.

Figure 22: Harmonic oscillator in x/y state space with an obstacle O (gray). Initially, the black cells $\lceil O \rceil_{\mathcal{G}}$ are unsafe. The example state (green) leads to an unsafe cell, rendering its cell unsafe too. In the fixpoint, all cells are unsafe. Transformation to polar coordinates. The initial marking is also the fixpoint.

We remark that, since we are only interested in safety at discrete points in time, finer partitions could still yield safety strategies for this example.

3 Shielding in Transformed State Spaces

In this section, we show how a transformation of the state space can be used for grid-based shield synthesis, and demonstrate that it can be instrumental.

3.1 State-Space Transformations

We recall the principle of state-space transformations. Consider a state space $S \subseteq \mathbb{R}^d$. A transformation to another state space $T \subseteq \mathbb{R}^{d'}$ is any function $f : S \to T$.

For our application, some transformations are better than others. We call these transformations *grid-friendly*, where, intuitively, cells in the transformed state space T are better able to separate the controllable from the uncontrollable states, i.e., capture the decision boundaries well. This is for instance the case if there is an invariant property and f maps this property to a single dimension.

Example 4 Consider again the harmonic oscillator from Example 3. We transform the system to a new state space, with the goal of circumventing the two problems identified above. Recall that we want to be able to represent a disc shape as well as circular trajectories in a grid-friendly way. Observe that the radius of the circle described by a trajectory is an invariant of the trajectory. This motivates to choose a transformation from Cartesian coordinates to polar coordinates. In polar coordinates, instead of x and y, we have the dimensions θ (angle) and r (radius). The transformation is $f(x, y) = (\theta, r)^\top =$

$\left(\text{atan2}\ (y,x), \sqrt{x^2+y^2}\right)^\top$, and the transformed state space is $T = [-\pi;\pi[\times [0;\sqrt{8}[$. The result after transforming the system, including the obstacle and the two example states, is shown in Figure 22c. As can be seen, the grid boundaries are parallel to both the obstacle boundaries as well as the dynamics, which is the best-case scenario. Observe that the radius dimension (r) is invariant. Hence, no white cell reaches a black cell and no further cells need to be marked unsafe.

3.2 Shield Synthesis in a Transformed State Space

In the following, we assume to be given a control system (S, Act, δ_S), a safety property φ, another state space T, a transformation $f : S \to T$, and a grid $\mathcal{G} \subseteq 2^T$. Our goal is to compute the controllable cells similar to Equation 8. However, since the grid is defined over T, we need to adapt the definition. The set of controllable cells is the maximal set of cells $\mathcal{C}^f_\varphi$ such that

$$\mathcal{C}^f_\varphi = \lfloor f(\varphi) \rfloor_\mathcal{G} \cap \{C \in \mathcal{G} \mid \exists a \in Act.\, \forall C'.\, C \xrightarrow{a} C' \implies C' \in \mathcal{C}^f_\varphi\} \tag{10}$$

The first change is to map φ to cells over T. Next, it is convenient to define a new control system (T, Act, δ_T) that imitates the original system in the new state space. The new successor function $\delta_T : T \times Act \to 2^T$ is given indirectly as

$$\delta_T(f(s), a) = f(\delta_S(s, a)) \tag{11}$$

The second change in Equation 10 is implicit in the transition relation $C \xrightarrow{a} C'$ of the labeled transition system $(\mathcal{G}, Act, \to)$. Recall from Equation 7 that the transitions are defined in terms of the successor function δ_T:

$$C \xrightarrow{a} C' \Leftrightarrow \exists t \in C.\, \delta_T(t, a) \cap C' \neq \emptyset \tag{12}$$

State-based successor computation.

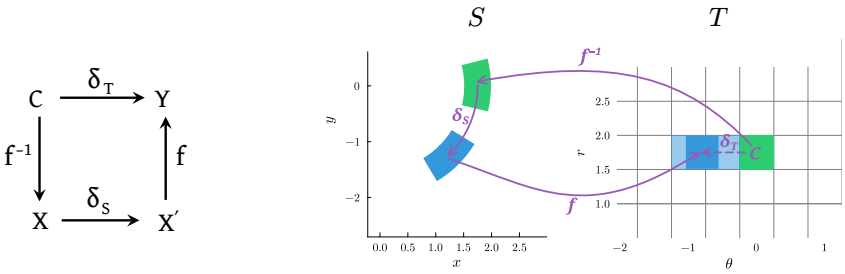
To simplify the presentation, for the moment, we only consider a single state $t \in T$. To effectively compute its successors, we cannot directly use Equation 11 because it starts from a state $s \in S$ instead. Hence, we first need to map t back to S using the *inverse transformation* $f^{-1} : T \to 2^S$, defined as $f^{-1}(t) = \{s \in S \mid f(s) = t\}$. The resulting set is called the *preimage*.

Now we are ready to compute $\delta_T(t, a)$ for any state $t \in T$ and action $a \in Act$. First, we map t back to its preimage $X = f^{-1}(t)$. Second, we apply the original successor function δ_S to obtain $X' = \delta_S(X, a)$. Finally, we obtain the corresponding transformed states $Y = f(X')$. In summary, we have

$$\delta_T(t, a) = f(\delta_S(f^{-1}(t), a)) \tag{13}$$

Note that, if the transformation f is bijective, its inverse f^{-1} is deterministic and we have $f^{-1}(f(s)) = s$ and $f(f^{-1}(t)) = t$ for all $s \in S$ and $t \in T$.

Consider again the harmonic oscillator from Example 3. The inverse transformation is $f^{-1}(\theta, r) = \begin{pmatrix} r\cos(\theta) \\ r\sin(\theta) \end{pmatrix}$. The blue successor state of the green state in Figure 22c is computed by mapping to the green state in Figure 22a via f^{-1}, computing the blue successor state via δ, and mapping back via f.



(a) Commutative diagram. (b) Illustration of $\delta_T(C, a) = f(\delta_S(f^{-1}(C), a))$.

Figure 23: The successor function δ_T for the cell C (green) in the transformed state space T is computed in three steps. First, we map to the original state space S via f^{-1}. Second, we compute the successors via δ_S. Third, we map back to the transformed state space T via f (dark blue). Finally, we can identify all cells intersecting with this set via $\lceil \cdot \rceil_{\mathcal{G}}$ (light blue).

Grid-based successor computation.

Equation 13 is directly applicable to cells via set extension, and no further modification is required. We provide illustrations of the construction in Figure 23.

The construction allows us to compute sound shields, both in the transformed and in the original state space.

Theorem 5 Let (S, Act, δ) be a control system, $\varphi \subseteq S$ be a safety property, $f : S \to T$ be a transformation with inverse $f^{-1} : T \to 2^S$, and $\mathcal{G} \subseteq 2^T$ be a partition of T. Define the control system (T, Act, δ_T) with δ_T according to Equation 13. Let $\mathcal{C}_\varphi^f$ be according to Equation 10 and $\sigma^{\mathcal{G}}$ be a safety strategy over cells. Then the following are safety strategies over states:

- $\sigma^Y(t) = \sigma^{\mathcal{G}}([t]_{\mathcal{G}})$ over states in $t \in Y \subseteq T$, where $Y = \bigcup_{C \in \mathcal{C}_\varphi^f} C$
- $\sigma^X(s) = \sigma^{\mathcal{G}}([f(s)]_{\mathcal{G}})$ over states in $s \in X \subseteq S$, where $X = f^{-1}\left(\bigcup_{C \in \mathcal{C}_\varphi^f} C\right)$

Proof We first need to argue that δ_T is well-defined. If f is not injective, then f^{-1} is nondeterministic, i.e., generally yields a set of states, but the set extension of δ_S treats this case. If f is not surjective, its inverse is undefined for some states $t \in T$. Note that the set extension ignores these states: for any set $Y \subseteq T$ we

have $f^{-1}(Y) = \{s \in S \mid f(s) \in Y\}$. In particular, if no state in C has a preimage, $\delta_T(C, a) = \emptyset$. Thus, δ_T is well-defined.

The first claim follows directly from Lemma 2. For the second claim, fix any state $s \in X$ and $a \in \sigma^X(s)$. We need to show that all states in $\delta_S(s, a)$ are safe, i.e., in X as well. By construction, $f(s) \in \bigcup_{C \in \mathcal{C}_\varphi^f} C$ and $a \in \sigma^{\mathcal{G}}([f(s)]_{\mathcal{G}})$. Hence,

$$\delta_T(f(s), a) \subseteq \bigcup_{C \in \mathcal{C}_\varphi^f} C \tag{14}$$

We also use the following simple lemma:

$$\forall s' \in S.\, s \in f^{-1}(f(s')) \tag{15}$$

Finally, we get (applying monotonicity in the last inclusion):

$$\begin{aligned} \delta_S(s, a) &\overset{(15)}{\subseteq} \delta_S(f^{-1}(f(s)), a) \overset{(15)}{\subseteq} f^{-1}(f(\delta_S(f^{-1}(f(s)), a))) \\ &\overset{(13)}{\subseteq} f^{-1}(\delta_T(f(s), a)) \overset{(14)}{\subseteq} f^{-1}\left(\bigcup_{C \in \mathcal{C}_\varphi^f} C\right) = X \end{aligned} \tag{16}$$

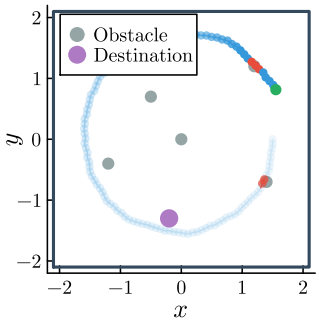
□

Note that we obtain Lemma 2 as the special case where f is the identity.

3.3 Shielding and Learning

We assume the reader is familiar with the principles of reinforcement learning. Here we shortly recall from [10] how to employ $\sigma^{\mathcal{G}}$ for safe reinforcement learning. The input is a Markov decision process (MDP) and a reward function, and the output is a controller maximizing the expected cumulative return. The MDP is a model with probabilistic successor function $\delta_P : S \times Act \times S \to [0, 1]$. An MDP induces a control system (S, Act, δ_S) with nondeterministic successor function $\delta_S(s, a) = \{s' \in S \mid \delta_P(s, a, s') > 0\}$ as an abstraction where the distribution has been replaced by its support.

Now consider Figure 21, which integrates a transformed shield into the learning process. In each iteration, the shield removes all unsafe actions (according to $\sigma^{\mathcal{G}}$) from the agent's choice. By construction, when starting in a controllable state, at least one action is available, and all available actions are guaranteed to lead to a controllable state again. Thus, by induction, all possible trajectories are infinite and never visit an unsafe state. Furthermore, filtering unsafe actions typically improves learning convergence because fewer options need to be explored.



State Space

$(x, y) \in S = [-2; 2[\times [-2; 2[$
$(\theta, r) \in T = [-\pi; \pi[\times [0; 2[$
$f(x, y) = \left(\text{atan2}\ (y, x), \sqrt{x^2, y^2}\right)^\top$

Figure 24: Satellite model.

Learning in S and T.
Recall from Theorem 5 that we can apply the shield both in the transformed state space and in the original state space by using the transformation function f. This allows us to also perform the learning in either state space. We consider the following setup the default: learning in the original state space S under a shield computed in the transformed state space T.

An alternative is to directly learn in T. A potential motivation could be that learning, in particular agent representation, may also be easier in T. For instance, the learning method implemented in UPPAAL STRATEGO represents an agent by axis-aligned hyperrectangles [82]. Thus, a grid-friendly transformation may also be beneficial for learning, independent of the shield synthesis. We will investigate the effect in our experiments.

4 Experiments

In this section, we demonstrate the benefits of state-space transformations for three models.[3] For the first two models, we use domain knowledge to select a suitable transformation. For the third model, we instead derive a transformation experimentally. The implementation builds on our synthesis method [10].

4.1 Satellite Model

For the first case study, we extend the harmonic oscillator with two more control actions to also move inward and outward: $Act = \{ahead, out, in\}$. The box to the side shows the relevant information about the transformation. Compared to Example 4, beside the actions, we modify two parts. First, the transformed state space T is reduced in the radius dimension to $r \in [0; 2[$ because values outside the disc with radius 2 are not considered safe (see below). Second, the successor function still uses matrix A from Example 1 but with a control period of $t = 0.05$.

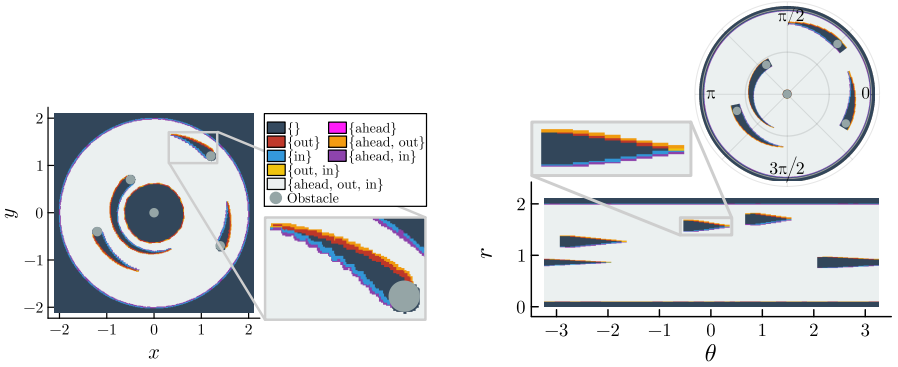
[3] A repeatability package is available here:
https://github.com/AsgerHB/state-space-transformation-shielding.

The successor function thus becomes $\delta(s,a) = e^{At}(\begin{pmatrix} rc\cos(\theta) \\ rc\sin(\theta) \end{pmatrix} HELLOOO$, where for $s = (x,y)^\top$ and f as in Example 4 we have

$$\begin{pmatrix} \theta \\ r \end{pmatrix} = f(s), \quad c = \begin{cases} 0.99 \text{ if } a = in \\ 1.01 \text{ if } a = out, \\ 1 \quad \text{otherwise.} \end{cases} \quad e^{At} \approx \begin{pmatrix} 1.00 & 0.05 \\ -0.05 & 1.00 \end{pmatrix} \tag{17}$$

Instead of one large obstacle, we add several smaller stationary (disc-shaped) obstacles. The shield has two goals: first, the agent must avoid a collision with the obstacles; second, the agent's distance to the center must not exceed 2. Figure 24 shows the size and position of the obstacles (gray). Overlaid is a trajectory (blue) produced by a random agent that selects actions uniformly. Some states of the trajectory collide with obstacles (red).

Additionally, we add an optimization component to the system. A disc-shaped *destination* area (purple) spawns at a random initial position (inside the 2-unit circle). Colliding with this area grants a reward and causes it to reappear at a new position. The optimization criterion for the agent is thus to visit as many destinations as possible during an episode.



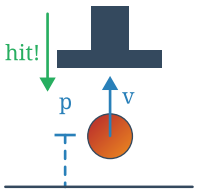
(a) Shield in S (176,400 cells).

(b) Shield in T (27,300 cells), including a transformation back to S.

Figure 25: Shields for the satellite model. The legend applies to both figures.

Figure 25a shows a shield obtained in the original state space. First, we note that a fine grid granularity is required to accurately capture the decision boundaries. In particular, the "tails" behind the obstacles split into regions where moving *ahead* is no longer possible. There is a small region at the tip of the tail (yellow) where the agent may either move *in* or *out*, but not *ahead* anymore.

Moreover, despite this high precision, the obstacle in the center causes a large set of cells around it to be marked unsafe, although we know that the *ahead* (and also *out*) action is safe. This is a consequence of the abstraction in the grid.



State Space

$(v, p) \in S = [-13; 13[\times [0; 8[$
$(E_m, v) \in T = [0; 100[\times [-13; 13[$
$f(v, p) = \left(mgp + \frac{1}{2}mv^2, v\right)^\top$

Figure 26: Bouncing-ball model.

Now we transform the system, for which we choose polar coordinates again. Figure 25b shows a shield obtained in this transformed state space. As we saw for the harmonic oscillator, the boundary condition is well captured by a grid. The obstacles also produce "tails" in this transformation, which require relatively high precision in the grid to be accurately captured. Still, since the shapes are axis-aligned, and the size of the transformed state space is different, the number of cells can be reduced by one order of magnitude. The grid over the original state space had $176,400$ cells, compared to $27,300$ cells in the transformed state space. Computing the original shield took 2 minutes and 41 seconds, while computing the transformed shield only took 10 seconds. Finally, the region marked unsafe at the bottom of Figure 25b, which corresponds to the central obstacle in the original state space, is tight, unlike in Figure 25a. In summary, the transformed shield is both easier to compute and more precise.

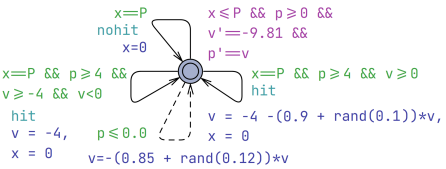
4.2 Bouncing-Ball Model

For the second case study, we consider the model of a bouncing ball from [10]. Figure 26 shows an illustration of the system, while Figure 27a shows the hybrid-automaton model. The state space consists of the velocity v and the position p of the ball. When the ball hits the ground, it loses energy subject to a stochastic dampening (dashed transition). The periodic controller is modeled with a clock x with implicit dynamics $\dot{x} = 1$ and control period $P = 0.1$. The available actions are $Act = \{nohit, hit\}$, where the *nohit* action has no effect and the *hit* action pushes the ball downward subject to its velocity, but only provided it is high enough ($p \geq 4$).

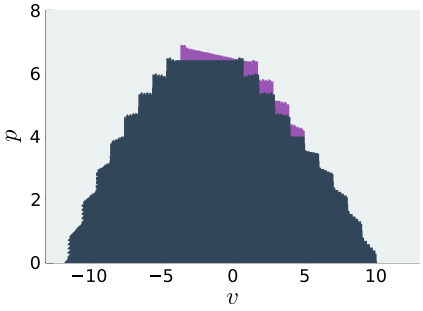
The goal of the shield is to keep the ball bouncing indefinitely, which is modeled as nonreachability of the set of states $p \leq 0.01 \wedge |v| \leq 1$.

The optimization task is to use the *hit* action as rarely as possible, which is modeled by assigning it with a cost and minimizing the total cost.

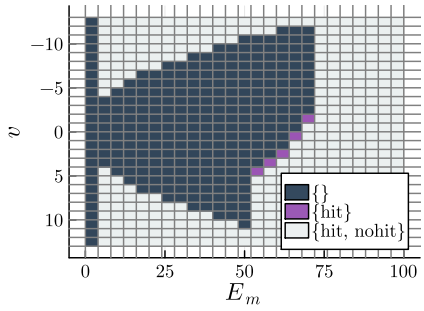
Despite its simple nature, this model has quite intricate dynamics, including stochastic and hybrid events that require zero-crossing detection, which makes determining reachability challenging. It was shown in [10] that a sampling-based shield synthesis is much more scalable than an approach based on guaranteed reachability analysis (19 minutes compared to 41 hours). The grid needs to be quite fine-grained to obtain a fixpoint where not every cell is marked unsafe. This corresponds to $520,000$ cells, and the corresponding shield is shown in Figure 27b.



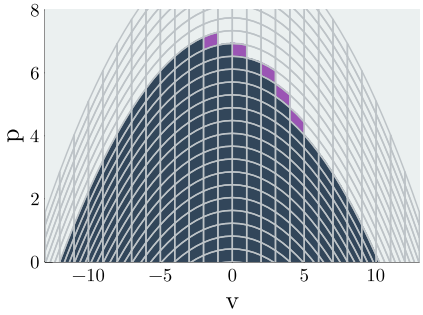
(a) Hybrid automaton.



(b) Shield in S (520,000 cells).



(c) Shield in T (650 cells).



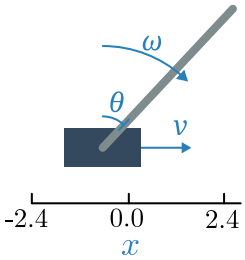
(d) Shield in T transformed back to S.

Figure 27: Hybrid automaton and shields for the bouncing-ball model.

Now we use a transformation to make the shield synthesis more efficient. The mechanical energy E_m stored in a moving object is the sum of its potential energy and its kinetic energy, respectively. Formally, $E_m(p, v) = mgp + \frac{1}{2}mv^2$, where $m = 1$ is the mass and $g = 9.81$ is gravity. Thus, the mechanical energy of a ball in free fall (both with positive or negative velocity) remains invariant. Hence, E_m is a good candidate for a transformation.

However, only knowing E_m is not sufficient to obtain a permissive shield because states with the same value of E_m may be below or above $p = 4$ and hence may or may not be hit. The equation for E_m depends on both p and v. In this case, it is sufficient to know only one of them. Here, we choose the transformed state space T with just E_m and v. The transformation function is $f(v, p) = \left(mgp + \frac{1}{2}mv^2, v\right)^\top$ and its inverse is $f^{-1}(E_m, v) = \left(\left(E_m - \frac{1}{2}mv^2\right)/(mg), v\right)^\top$. We note that using E_m and p instead yields a shield that marks all cells unsafe. This is because v is quadratic in E_m and, thus, we cannot determine its sign.

Figure 27c shows the shield obtained in this transformed state space. It can be seen that this results in a very low number of just 650 cells in total, which is a reduction by three orders of magnitude. This shield can be computed in just 1.3 seconds, which compared to 19 minutes is again a reduction by three orders of magnitude.



State Space

$(\theta, \omega) \in S = [-2.095; 2.095[\times [-3; 3[$
$(\theta, p(\theta, \omega)) \in T = [-2.095; 2.095[\times [-3; 3[$
$f(\theta, \omega) = (\theta, \omega - p(\theta))^\top$

Figure 28: Cart-pole model.

To provide more intuition about how precise this shield still is, we project the shield back to the original state space in Figure 27d. While a direct comparison is not fair because the grid granularity differs vastly, overall the shapes are similar.

4.3 Cart-Pole Model

For the third case study, we consider a model of an inverted pendulum installed on a cart that can move horizontally. This model is known as the cart-pole model. An illustration is shown in Figure 28. The dynamics are given by the following differential equations [83]:

$$\begin{aligned} \dot{\theta} = \omega\dot{\omega} &= \frac{g\sin(\theta) + \cos(\theta) \cdot \left(\frac{-F - m_p \ell \omega^2 \sin(\theta)}{m_c + m_p}\right)}{\ell\left(\frac{4}{3} - \frac{m_p \cos^2(\theta)}{m_c + m_p}\right)} \\ \dot{x} = v\,\dot{v} &= \frac{F + m_p \ell(\omega^2 \sin(\theta) - \dot{\omega}\cos(\theta))}{m_c + m_p} \end{aligned} \tag{18}$$

The state dimensions are the pole's angle θ and angular velocity ω as well as the cart's position x and velocity v. Moreover, $g = 9.8 m/s^2$ is gravity, $\ell = 0.5\,\text{m}$ is the pole's length, $m_p = 0.1\,\text{kg}$ is the pole's mass, and $m_c = 1\,\text{kg}$ is the cart's mass. Finally, $F = \pm 10$ is the force that is applied, corresponding to the action from $Act = \{left, right\}$, which can be changed at a rate of 0.02 (control period).

The goal of the shield is to balance the pole upright, which translates to the condition that the angle stays in a small cone $|\theta| \leq 0.2095$.

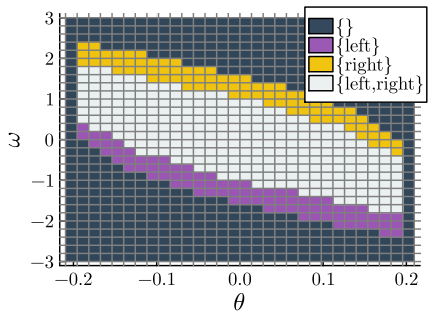
The optimization goal is to keep the cart near its initial position $x(0)$. Moving more than $2.4\,\text{m}$ away yields a penalty of 1 and resets the cart.

Observe that the property for the shield only depends on the pole and not on the cart. Hence, it is sufficient to focus on the pole dimensions θ and ω for shield synthesis, and leave the cart dimensions for the optimization. A shield in the original state space is shown in Figure 29a.

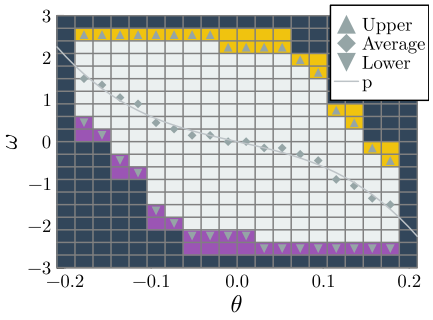
In the following, we describe a state-space transformation for shield synthesis. Unlike for the other models, we are not aware of an invariant property that is useful for our purposes. Instead, we will derive a transformation in two steps.

Recall that a transformation is useful if a grid in the new state space captures the decision boundaries well, i.e., the new decision boundaries are roughly axis-aligned. Thus, our plan is to approximate the shape of the decision boundaries in the first step and then craft a suitable transformation in the second step.

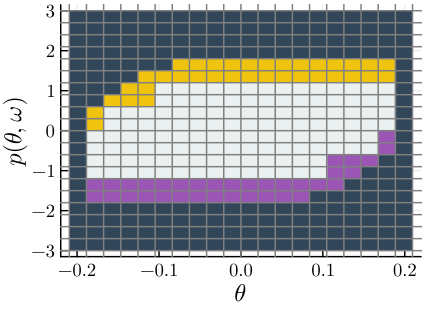
Approximating the Decision Boundaries.



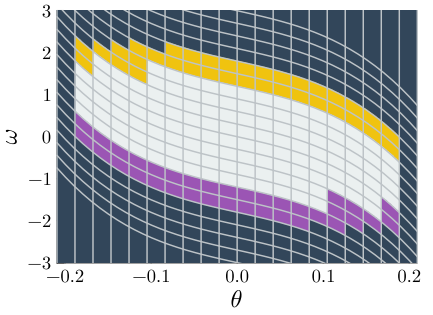
(a) Shield in S (900 cells).



(b) Approximating the decision boundaries in S.



(c) Shield in T (400 cells).



(d) Shield in T projected back to S.

Figure 29: Shield computation for the cart-pole model. The legend in Figure 29a applies to all subfigures.

Figure 29a shows the decision boundaries of a fixpoint computed using 30×30 cells. However, our work of state-space transformations was motivated because computing the shield is generally not feasible in the first place.

Therefore, here we take a different approach, which uses a grid of just 20×20 cells. Computing a shield for such a coarse grid in the original state space yields $\mathcal{C}_\varphi = \emptyset$, i.e., all cells become unsafe. This is a consequence of the abstraction, i.e., a trajectory at the grid level may be spurious at the state level. This abstraction grows with the number of steps of the trajectory. Our idea is thus to only perform the fixpoint iteration at the grid level for a low number (here: three) of steps. (Technically, this means that the strategy is only guaranteed to be safe for three steps.) The result

is the marking of cells in Figure 29b. Indeed, the decision boundaries roughly approximate those in Figure 29a.

Crafting a Transformation.
We want to find a grid-friendly transformation that "flattens out" the decision boundaries. Our idea is to keep the dimension θ and replace ω by a transformation that is "flatter." We observe that the upper (yellow) and lower (purple) decision boundaries are symmetric. Hence, the distance to the average of the upper and lower boundaries is a good approximation.

This idea is visualized in Figure 29b. Here we compute the average (diamonds) of the upper and lower boundaries (triangles). Then we fit a polynomial to approximate this shape. In our implementation, we used the Julia `Polynomials` library, which implements a standard linear least squares method [84]. Here, a third-degree polynomial $p(\theta) = -141.6953 \cdot \theta^3 - 4.5508 \cdot \theta$ is sufficient.

To obtain the full transformation, we need to express the offset from $p(\theta)$. Thus, we choose $f(\theta, \omega) = (\theta, \omega - p(\theta))^\top$. The inverse function is $f^{-1}(\theta, z) = (\theta, z + p(\theta))^\top$, where z is the new dimension in the transformed state space (T).

The resulting shield is shown in Figure 29c. The grid size is 400 cells, as compared to 900 cells in the original state space. Both took less than a second to synthesize, at 244 ms and 512 ms, respectively.

4.4 Strategy Reduction

Model	State space	Number of cells	Number of nodes
Satellite	S	176,400	4,913
	T	27,300	544
Bouncing ball	S	520,000	940
	T	650	49
Cart-pole	S	900	99
	T	400	32

Table 7: Representation sizes of the computed shields.

We provide an overview of the savings due to computing the shield in the transformed state space in Table 7. The column labeled *Number of cells* clearly shows a significant reduction in all cases. We remark that, in order to have a fair comparison, we have selected the grid sizes from visual inspection to ensure that the plots look sufficiently close. However, it is not the case that one of the shields is more permissive than the other.

The strategies above can be represented with a d-dimensional matrix. Matrices are inherently limiting representations of shields, especially when the shield should be stored on an embedded device. Empirically, a decision tree with axis-aligned predicates is a much better representation. To demonstrate the further saving potential, we converted the shields to decision trees and additionally applied the reduction technique from [85]. The last column in Table 7 shows the number of nodes in the decision trees. As can be seen, we always achieve another significant reduction by one to two orders of magnitude.

4.5 Shielded Reinforcement Learning

Learning	**Satellite (↗)**			**Bouncing ball (↘)**			**Cart-pole, (↘)**		
	None	S	T	None	S	T	None	S	T
S	1.123	0.786	**1.499**	39.897	37.607	**36.593**	0.007	0.019	**0.001**
T	0.917	0.889	**1.176**	39.128	40.024	**39.099**	**0.000**	**0.000**	**0.000**

Table 8: Cumulative return over 1000 episodes with both shielding and learning in either of the state spaces. Higher return is better for the satellite model, and vice versa for the other models. Each row's best result is marked in bold face.

The only motivation for applying a state-space transformation was to be able to compute a cheaper shield. From the theory, we cannot draw any conclusions about the impact on the controller performance. We investigate this impact in the following experiments, with the main result that the transformed shield actually increases the performance consistently.

We conduct six experiments for each of the three models. For shielding, we consider three variants (no shield, shielding in the original state space S, and shielding in the transformed state space T). For each variant, we reinforcement-learn two controllers. One controller is trained in the original state space S, while the other controller is trained in the transformed state space T.

In Table 8, we provide the learning results for all combinations of shielding and learning. The data is given as the cumulative return obtained over 1000 executions of the environment and the respective learned agent using UPPAAL.

The results show that, for all models, the highest reward is achieved by the controller operating under the shield in the transformed state space. This holds regardless of which state space the controller was trained in. Additionally, the controller that was trained in the original state space achieves higher performance. Thus, the transformation was not helpful for the learning process itself.

5 Conclusion

We have demonstrated that state-space transformations hold great potential for shield synthesis. We believe that they are strictly necessary when applying shield synthesis to many practical systems due to state-space explosion.

In the first two case studies, we used domain knowledge to select a suitable transformation. In the third case study, we instead engineered a transformation in two steps. We plan to generalize these steps to a principled method and investigate how well it applies in other cases.

State-space transformations can be integrated with many orthogonal prior extensions of grid-based synthesis. One successful extension is, instead of precomputing the full labeled transition system, to compute its transitions on the fly [86]. Another extension is the multilayered abstraction [73], [74]. Going one step further, in cases where a single perfect transformation does not exist, we may still be able to find a family of transformations of different strengths.

Acknowledgments

We thank Tom Henzinger for the suggestion to study level sets.

Paper C:

Compositional Shielding and Reinforcement Learning for Multi-agent Systems

Asger Horn Brorholt
Department of Computer Science
Aalborg University, Aalborg, Denmark

Kim Guldstrand Larsen
Department of Computer Science
Aalborg University, Aalborg, Denmark

Christian Schilling
Department of Computer Science
Aalborg University, Aalborg, Denmark

Abstract

Deep reinforcement learning has emerged as a powerful tool for obtaining high-performance policies. However, the safety of these policies has been a long-standing issue. One promising paradigm to guarantee safety is a *shield*, which "shields" a policy from making unsafe actions. However, computing a shield scales exponentially in the number of state variables. This is a particular concern in multi-agent systems with many agents. In this work, we propose a novel approach for multi-agent shielding. We address scalability by computing individual shields for each agent. The challenge is that typical safety specifications are global properties, but the shields of individual agents only ensure local properties. Our key to overcome this challenge is to apply assume-guarantee reasoning. Specifically, we present a sound proof rule that decomposes a (global, complex) safety specification into (local, simple) obligations for the shields of the individual agents. Moreover, we show that applying the shields during reinforcement learning significantly improves the quality of the policies obtained for a given training budget. We demonstrate the effectiveness and scalability of our multi-agent shielding framework in two case studies, reducing the computation time from hours to seconds and achieving fast learning convergence.

1 Introduction

Reinforcement learning (RL) [87], [88], and in particular deep RL, has demonstrated success in automatically learning high-performance policies for complex systems [89], [90]. However, learned policies lack guarantees, which prevents applications in safety-critical domains.

An attractive algorithmic paradigm to provably safe RL is *shielding* [9]. In this paradigm, one constructs a *shield*, which is a nondeterministic policy that only allows safe actions. The shield acts as a guardrail for the RL agent to enforce safety both during learning (of a concrete policy) and operation. This way, one obtains a *safe-by-design shielded policy with high performance*.

Shield synthesis automatically computes a shield from a safety specification and a model of the system, but scales exponentially in the number of state variables. This is a particular concern in multi-agent (MA) systems, which typically consist of many variables. Shielding of MA systems will be our focus in this work.

Existing approaches to MA shielding address scalability by computing individual shields for each agent. Yet, these shields are either not truly safe or not truly independent; rather, they require online communication among all agents, which is often unrealistic.

In this paper, we present the first MA shielding approach that is truly compositional, does not require online communication, and provides absolute safety guarantees. Concretely, we assume that agents observe a subset of all system variables (i.e., operate in a projection of the global state space). We show how to tractably synthesize individual shields in low-dimensional projections. The challenge we need to overcome is that a straightforward generalization of the classical shield synthesis to the MA setting for truly independent shields often fails. The reason is that the projection removes the potential to coordinate between the agents, but often some form of coordination is required.

To address the need for coordination, we get inspiration from *compositional reasoning*, which is a powerful approach, allowing to scale up the analysis of distributed systems. The underlying principle is to construct a correctness proof of multi-component systems by smaller, "local" proofs for each individual component. In particular, *assume-guarantee reasoning* for concurrent programs was popularized in seminal works [91], [92], [93], [94], [95]. By writing $\langle A \rangle C \langle G \rangle$ for "assuming A, component C will guarantee G," the standard (acyclic) assume-guarantee rule for finite state machines with handshake synchronization looks as follows [96]:

$$\frac{\langle \top \rangle C_1 \langle G_1 \rangle, \langle G_1 \rangle C_2 \langle G_2 \rangle, ..., \langle G_{n-2} \rangle C_{n-1} \langle G_{n-1} \rangle, \langle G_{n-1} \rangle C_n \langle \phi \rangle}{\langle \top \rangle C_1 \parallel C_2 \parallel \cdots \parallel C_n \langle \phi \rangle} \tag{19}$$

By this chain of assume-guarantee pairs, it is clear that, together, the components ensure safety property ϕ.

In this work, we adapt the above rule to multi-agent shielding. Instead of one shield for the whole system, we synthesize an individual shield for each agent, which together we call a *distributed shield*. Thus, we arrive at n shield synthesis problems (corresponding to the rule's premise), but each of which is efficient. In our case studies, this reduces the synthesis time from hours to seconds. The *guarantees* G_i *allow the individual shields to coordinate* on responsibilities at synthesis time. Yet, distributed shields do not require communication when deployed. Altogether, this allows us to *synthesize safe shields* in a compositional and scalable way.

The crucial challenge is that, in the classical setting, the components C_i are fixed. In our synthesis setting, the components C_i are our agents, which are *not* fixed at the time of the shield synthesis. In this work, we assume that the guarantees G_i are given, which allows us to derive corresponding individual agent shields via standard shield synthesis.

1.1 Motivating Example

A multi-agent car platoon with adaptive cruise controls consists of n cars, numbered from back to front [97] (Figure 30). The cars 1 to $n-1$ are each controlled by an agent, while (front) car n is driven by the environment. The state variables are the car velocities v_i and distances d_i between cars i and $i+1$. For $i < n$, car i follows car $i+1$, observing the variables (v_i, v_{i+1}, d_i). With a decision period of 1 second, cars act by choosing an acceleration from $\{-2, 0, 2\}$ [m/s^2]. Velocities are capped between $[-10; 20]$ m/s.

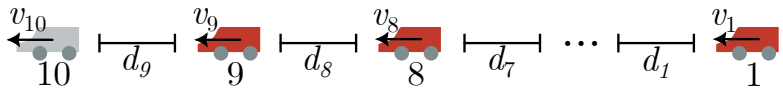


Figure 30: Car platoon example for $n = 10$ cars.

When two cars have distance 0, they enter an uncontrollable "damaged" state where both cars get to a standstill. The global safety property is to maintain a safe but bounded distance between all cars, i.e., the set of safe states is $\phi = \{s \mid \bigwedge_i 0 < d_i < 200\}$.

As a first attempt, we design the agents' individual safety properties to only maintain a safe distance to the car in front, i.e., $\phi_i = \{s \mid 0 < d_i < 200\}$ for all i. However, safe agent shields for cars $i > 1$ do not exist for this property: car i cannot prevent a crash with car $i-1$ (behind), and in the "damaged" (halting) state car i cannot guarantee to avoid crashing with car $i+1$. Note that making the distance d_{i-1} observable for car i does not help.

To overcome this seemingly impossible situation, we will allow car i to *assume* that the (unobservable) car $i-1$ *guarantees* to never crash into car i. This guarantee will be provided by the shield of car $i-1$ and eliminates the critical behavior

preventing a local shield for car i. In that way, we iteratively obtain local shields for all agents. Note that this coincides with human driver reasoning.

Beside synthesis of a distributed shield, we also study learning policies for shielded agents. In general, multi-agent reinforcement learning (MARL) [98] is complex due to high-dimensional state and action spaces, which impede convergence to optimal policies.

Here, we identify a class of systems where learning the agents in a *cascading* way is both effective and efficient. Concretely, if we assign an index to each agent, and each agent only depends on agents with lower index, we can learn policies in a sequential order. This leads to a low-dimensional space for the learning algorithm, which leads to fast convergence. While in general suboptimal, we show that this approach still leads to Pareto-optimal results.

In summary, this paper makes the following main contributions:

- We propose *distributed shielding*, the first MA shielding approach with absolute safety guarantees and scalability, yet without online communication. To this end, our approach integrates shield synthesis and assume-guarantee reasoning.
- We propose (shielded) *cascading learning*, a scalable MARL approach for systems with acyclic dependency structure, which further benefits from assume-guarantee reasoning.
- We evaluate our approaches in two case studies. First, we demonstrate that distributed shielding is scalable and, thanks to the integration of assume-guarantee reasoning, applicable. Second, we demonstrate that shielded cascading learning is efficient and achieves state-of-the-art performance.

1.2 Related Work

Shielding.
As mentioned, shielding is a technique that computes a *shield*, which prevents an agent from taking unsafe actions. Thus, any policy under a shield is safe, which makes it attractive for safety both during learning and after deployment. Shields are typically based on game-theoretic results, where they are called *winning strategies* [99]. Early applications of shields in learning were proposed for timed systems [100] and discrete systems [9]. The idea has since been extended to probabilistic systems [101], [102], partial observability [103], and continuous-time dynamics [10], [104]. For more background we refer to surveys [105], [106]. In this work, we focus on discrete but multi-agent systems, which we now review in detail.

Multi-agent shielding.
An early work on multi-agent enforcement considered a very restricted setting with deterministic environments where the specification is already given in terms of valid actions and not in terms of states [107]. Thus, the shield does not reason about the dynamics and simply overrides forbidden actions.

Model-predictive shielding assumes a backup policy together with a set of recoverable states from which this policy can guarantee safety. Such a backup policy may for instance be implemented by a shield, and is combined with another (typically learned) policy. First, a step with the second policy is simulated and, when the target state is recoverable, this step is executed; otherwise, the fallback policy is executed. Crucially, this assumes that the environment is deterministic. Zhang et al. proposed a multi-agent version [108], where the key insight is that only some agents need to use the backup policy. For scalability, the authors propose a greedy algorithm to identify a sufficiently small subset of agents. However, the "shield" is centralized, which makes this approach not scalable.

Another work computes a safe policy online [109], which may be slow. Agents in close proximity create a communication group, and they communicate their planned trajectories for the next k steps. Each agent has an agreed-on priority in which they have to resolve safety violations, but if that is not possible, agents may disturb higher-priority agents. The approach requires strong assumptions like deterministic system dynamics and immediate communication.

One work suggests to directly reinforcement-learn policies by simply encouraging safety [110]. Here, the loss function encodes a safety proof called *barrier certificate*. But, as with any reward engineering, this approach does not guarantee safety in any way.

Another way to scale up shielding for multi-agent systems is a so-called *factored shield*, which safeguards only a subset of the state space, independent of the number of agents [111]. When an agent moves, it joins or leaves a shield at border states. However, this approach relies on very few agents ever interacting with each other, as otherwise, there is no significant scalability gain.

Factored shields were extended to *dynamic shields* [112]. The idea is that, in order to reduce the communication overhead, an agent's shield should "merge" dynamically with the shields of other agents in the proximity. Since the shields are computed with a k-step lookahead only, safety is not guaranteed invariantly.

Multi-agent verification.

Rational verification proposes to study specifications only from initial states in Nash equilibria, i.e., assuming that all agents act completely rationally [113]. While that assumption may be useful for rational/optimal agents, we typically have learned agents in mind, which do not always act optimally.

The tool *Verse* lets users specify multi-agent scenarios in a Python dialect and provides black-box (simulations) and white-box (formal proofs; our setting) analysis for time-bounded specifications [114].

Assume-guarantee reasoning has been applied to multi-agent systems in [115] and in [116], but not yet to (multi-agent) shielding.

Outline.
In the next section, we define basic notation. In Section 3, we introduce distributed shielding based on projections and extend it with assume-guarantee reasoning. In Section 4, we develop cascading learning, tailored to systems with acyclic dependencies. In Section 5, we evaluate our approaches in two case studies. In Section 6, we conclude and discuss future work.

2 Preliminaries

2.1 Transition Systems (MDPs & LTSs)

We start with some basic definitions of transition systems.

Definition 1 *(Labeled transition system)* A *labeled transition system* (LTS) is a triple $\mathcal{T} = (S, Act, T)$ where S is the finite state space, Act is the action space, and $T \subseteq S \times Act \times S$ is the transition relation with no dead ends, i.e., for all $s \in S$ there exists some $a \in Act$ and $s' \in S$ such that $(s, a, s') \in T$.

Definition 2 *(Markov decision process)* A *Markov decision process* (MDP) is a triple $\mathcal{M} = (S, Act, P)$ where S is the finite state space, Act is the action space, and $P : S \times Act \times S \to [0, 1]$ is the probabilistic transition relation satisfying $\sum_{s' \in S} P(s, a, s') \in \{0, 1\}$ for all $s \in S$ and $a \in Act$, and for at least one action, the sum is 1.

We will view an LTS as an abstraction of an MDP where probabilities are replaced by possibilities.

Definition 3 *(Induced LTS)* Given an MDP $\mathcal{M} = (S, Act, P)$, the *induced LTS* is $\mathcal{T}_{\mathcal{M}} = (S, Act, T)$ with $(s, a, s') \in T$ iff $P(s, a, s') > 0$.

Definition 4 *(Run)* Assume an LTS $\mathcal{T} = (S, Act, T)$ and a finite alternating sequence of states and actions $\rho = s_0 a_0 s_1 a_1 ...$; then, ρ is a *run* of $\mathcal{T}$ if $(s_i, a_i, s_{i+1}) \in T$ for all $i \geq 0$. Similarly, for an MDP $\mathcal{M} = (S, Act, P)$, ρ is a *run* of $\mathcal{M}$ if $P(s_i, a_i, s_{i+1}) > 0$ for all $i \geq 0$.

We distinguish between strategies and policies in this work. A strategy prescribes a nondeterministic choice of actions in each LTS state. Similarly, a policy prescribes a probabilistic choice of actions in each MDP state. Before defining them formally, we need a notion of restricting the actions to sensible choices.

Definition 5 *(Enabled actions)* Given an LTS, $\mathcal{E}(s) = \{a \in Act \mid \exists s' : (s, a, s') \in T\}$ denotes the *enabled actions* in state s. Similarly, given an MDP, $\mathcal{E}(s) = \{a \in Act \mid \exists s' : P(s, a, s') > 0\}$.

Definition 6 *(Strategy; policy)* Given an LTS, a (nondeterministic) *strategy* is a function $\sigma : S \to 2^{Act}$ such that $\emptyset \neq \sigma(s) \subseteq \mathcal{E}(s)$ for all $s \in S$. Given an MDP, a

(probabilistic) *policy* is a function $\pi : S \times Act \to [0,1]$ such that $\sum_{a \in \mathcal{E}(s)} \pi(s,a) = 1$ and $\bigwedge_{a' \in Act \setminus \mathcal{E}(s)} \pi(s,a') = 0$ for all $s \in S$.

Note that our strategies and policies are memoryless. This is justified as we will only consider safety properties in this work, for which memory is not required [99]. Strategies and policies restrict the possible runs, and we call these runs the outcomes.

Definition 7 *(Outcome)* A run $\rho = s_0 a_0 s_1 a_1 \ldots$ of an LTS is an *outcome* of a strategy σ if $a_i \in \sigma(s_i)$ for all $i \geq 0$. Similarly, a run $\rho = s_0 a_0 s_1 a_1 \ldots$ of an MDP is an outcome of a policy π if $\pi(s_i, a_i) > 0$ for all $i \geq 0$.

2.2 Safety and Shielding

In this work, we are interested in safety properties, which are characterized by a set of safe (resp. unsafe) states. The goal is to stay in the safe (resp. avoid the unsafe) states. In this section, we introduce corresponding notions, in particular (classical) shields and how they can be applied.

Definition 8 *(Safety property)* A *safety property* is a set of states $\phi \subseteq S$.

Definition 9 *(Safe run)* Given a safety property $\phi \subseteq S$, a run $s_0 a_0 s_1 a_1 \ldots$ is *safe* if $s_i \in \phi$ for all $i \geq 0$.

Given an LTS, a safety property $\phi \subseteq S$ partitions the states into two sets: the *winning states*, from which a strategy exists whose outcomes are all safe, and the complement. The latter can be computed as the attractor set of the complement $S \setminus \phi$ [99]. Since it is hopeless to ensure safe behavior from the complement states, in the following we will only be interested in outcomes starting in winning states, which we abstain from mentioning explicitly.

A shield is a (typically nondeterministic) strategy that ensures safety. In game-theory terms, a shield is called a *winning strategy*.

Definition 10 *(Shield)* Given an LTS (S, Act, T) and a safety property $\phi \subseteq S$, a *shield* $\shield[\phi]$ is a strategy whose outcomes starting in any winning state are all safe wrt. ϕ.

We often omit ϕ and just write $\shield$. Among all shields, it is known that there is a "best" one that allows the most actions.

Definition 11 *(Most permissive shield)* Given an LTS and a safety property ϕ, the *most permissive shield* $\shield^*[\phi]$ is the shield that allows the largest set of actions for each state $s \in S$.

Lemma 12 *([99])* $\shield^*$ is unique and obtained as the union of all shields $\shield$ for ϕ: $\shield^*(s) = \{a \in Act \mid \exists \shield : a \in \shield(s)\}$.

The standard usage of a shield is to restrict the actions of a policy for guaranteeing safety. In this work, we also compose it with another strategy. For that, we introduce the notion of composition of strategies (recall that a shield is also a strategy). We can, however, only compose strategies that are compatible in the sense that they allow at least one common action in each state (otherwise the result is not a strategy according to our definition).

Definition 13 *(Composition)* Two strategies σ_1 and σ_2 over an LTS (S, Act, T) are *compatible* if $\sigma_1(s) \cap \sigma_2(s) \neq \emptyset$ for all $s \in S$.

Given compatible strategies σ and σ', their composition $\sigma \sqcap \sigma'$ is the strategy $(\sigma \sqcap \sigma')(s) = \sigma(s) \cap \sigma'(s)$.

We write $\sqcap_{i<j}\sigma_i$ to denote $\sigma_1 \sqcap ... \sqcap \sigma_{j-1}$, and $\sqcap_i \sigma_i$ to denote $\sigma_1 \sqcap ... \sqcap \sigma_n$ when n is clear from the context.

Given a strategy σ and a compatible shield $\triangledown$, we also use the alternative notation of the *shielded strategy* $\triangledown(\sigma) = \sigma \sqcap \triangledown$.

Given a set of states ϕ, we are interested whether an LTS ensures that we will stay in that set ϕ, independent of the strategy.

Definition 14 Assume an LTS $\mathcal{T}$ and a set of states ϕ. We write $\mathcal{T} \vDash \phi$ if for all strategies σ, all corresponding outcomes $s_0 a_0 s_1 a_1 ...$ satisfy $s_i \in \phi$ for all $i \geq 0$.

We now use a different view on a shield and apply it to an LTS in order to "filter out" those actions that are forbidden by the shield.

Definition 15 *(Shielded LTS)* Given an LTS $\mathcal{T} = (S, Act, T)$, a safety property ϕ, and a shield $\triangledown[\phi]$, the *shielded LTS* $\mathcal{T}_{\triangledown} = (S, Act, T_{\triangledown})$ with $T_{\triangledown} = \{(s, a, s') \in T \mid a \in \triangledown(s)\}$ is restricted to transitions whose actions are allowed by the shield.

The next proposition asserts that a shielded LTS is safe.

Proposition 16 Given an LTS $\mathcal{T}$, a safety property ϕ, and a corresponding shield $\triangledown[\phi]$, all outcomes of any strategy for $\mathcal{T}_{\triangledown}$ are safe.

In other words, $\mathcal{T}_{\triangledown} \vDash \phi$. We analogously define shielded MDPs.

Definition 17 *(Shielded MDP)* Given an MDP $\mathcal{M} = (S, Act, P)$, a safety property ϕ, and a shield $\triangledown$ for $\mathcal{T}_{\mathcal{M}}$, the *shielded MDP* $\mathcal{M}_{\triangledown} = (S, Act, P_{\triangledown})$ is restricted to transitions with actions allowed by $\triangledown$: $P_{\triangledown}(s, a, s') = P(s, a, s')$ if $a \in \triangledown(s)$, and $P_{\triangledown}(s, a, s') = 0$ otherwise.

Proposition 18 Assume an MDP $\mathcal{M}$, a safety property ϕ, and a corresponding shield $\triangledown[\phi]$ for $\mathcal{T}_{\mathcal{M}}$. Then all outcomes of any policy for $\mathcal{M}_{\triangledown}$ are safe.

The last proposition explains how standard shielding is applied to learn safe policies. Given an MDP $\mathcal{M}$, we first compute a shield $\triangledown$ over the induced LTS $\mathcal{T}_{\mathcal{M}}$. Then we apply the shield to the MDP $\mathcal{M}$ to obtain $\mathcal{M}_{\triangledown}$ and filter unsafe actions. The shield guarantees that the agent is safe both during and after learning.

From now on we mainly focus on computing shields from an LTS, as the generalization to MDPs is straightforward.

2.3 Compositional Systems

Now we turn to compositional systems (LTSs and MDPs) with multiple agents. We restrict ourselves to k-dimensional state spaces S, i.e., products of variables $S = \bigtimes_i S_i$. We allow for sharing some of these variables among the agents by projecting to observation subspaces. The following is the standard definition of projecting out certain variables while retaining others. We use the notation that, given an n-vector $v = (v_1, ..., v_n)$, $v[i]$ denotes the i-th element v_i.

Definition 19 *(Projection)* A *projection* is a mapping $prj : S \to O$ that maps k-dimensional vectors $s \in S$ to j-dimensional vectors $o \in O$, where $j \leq k$. Formally, prj is associated with a sequence of j indices $1 \leq i_1 < ... < i_j \leq k$ such that $prj(s) = \big(s[i_1], ..., s\big[i_j\big]\big)$. Additionally, we define $prj(\phi) = \bigcup_{s \in \phi} \{prj(s)\}$.

Definition 20 *(Extension)* Given projection $prj : S \to O$, the set of states projected to o is the *extension* $\uparrow(o) = \{s \in S \mid prj(s) = o\}$.

Later we will also use an alternative projection, which we call *restricted*. The motivation is that the standard projection above sometimes retains too many states. The restricted projection instead only keeps those states such that the extension of the projection ($\uparrow(\cdot)$) is contained in the original set. For instance, for the state space $S = \{0, 1\}^2$, the set of states $\phi = \{(0, 0), (0, 1), (1, 0)\}$, and the one-dimensional projection $prj(s) = s[1]$, we have that $prj(\phi) = \{0, 1\}$. The restricted projection removes 1 as $(1, 1) \notin \phi$.

Definition 21 *(Restricted projection)* A *restricted projection* is a mapping $\overline{prj} : 2^S \to 2^O$ that maps sets of k-dimensional vectors $s \in S$ to sets of j-dimensional vectors $o \in O$, where $j \leq k$. Formally, $\overline{prj}$ is associated with a sequence of j indices $1 \leq i_1 < ... < i_j \leq k$. Let prj be the corresponding (standard) projection and $\phi \subseteq S$. Then $\overline{prj}(\phi) = \{o \in O \mid \{s \in S \mid prj(s) = o\} \subseteq \phi\}$. Again, we define $\overline{prj}(\phi) = \bigcup_{s \in \phi} \{\overline{prj}(s)\}$.

We will apply $\overline{prj}$ only to safety properties ϕ. The following alternative characterization may help with the intuition: $\overline{prj}(\phi) = prj\big(\overline{\phi}\big) = O \setminus prj(S \setminus \phi)$, where $\overline{\phi}$ denotes the complement $S \setminus \phi$ (resp. $O \setminus \phi$) of a set of states $\phi \subseteq S$ (resp. observations $\phi \subseteq O$).

Crucially, prj and $\overline{prj}$ coincide if $\uparrow(prj(\phi)) = \phi$, i.e., if the projection of ϕ preserves correlations. We will later turn our attention to agent safety properties, where this is commonly the case.

Now we can define a multi-agent LTS and MDP.

Definition 22 *(n-agent LTS/MDP)* An *n-agent LTS* (S, Act, T) or an *n-agent MDP* (S, Act, P) have an n-dimensional action space $Act = Act_1 \times ... \times Act_n$ and a family of n projections $prj_i, i = 1, ..., n$. Each *agent* i is associated with the projection $prj_i : S \to O_i$ from S to its *observation space* O_i.

We note that the observation space introduces partial observability. Obtaining optimal strategies/policies for partial observability is difficult and generally requires infinite memory [117]. Since this is impractical, we restrict ourselves to memoryless strategies/policies.

We can apply the projection function prj to obtain a "local" LTS, modeling partial observability.

Definition 23 *(Projected LTS)* For an n-agent LTS $\mathcal{T} = (S, Act, T)$ and an agent i with projection function $prj_i : S \to O_i$, the *projected LTS to agent* i is $\mathcal{T}^i = (O_i, Act_i, T_i)$ where $Act_i = \{a[i] \mid a \in Act\}$ and $T_i = \{(prj_i(s), a[i], prj_i(s')) \mid (s, a, s') \in T\}$.

3 Distributed Shield Synthesis

We now turn to shielding in a multi-agent setting. The straightforward approach is to consider the full-dimensional system and compute a global shield. This has, however, two issues. First, a global shield assumes communication among the agents, which we generally do not want to assume. Second, and more importantly, shield computation scales exponentially in the number of variables.

To address these issues, we instead compute *local* shields, one for each agent. A local shield still keeps its agent safe. But since we only consider the agent's observation space, the shield does not require communication, and the computation is much cheaper.

3.1 Projection-Based Shield Synthesis

Rather than enforcing the global safety property, local shields will enforce agent-specific properties, which we characterize next.

Definition 24 *(n-agent safety property)* Given an n-agent LTS or MDP with state space S, a safety property $\phi \subseteq S$ is an *n-agent safety property* if $\phi = \bigcap_{i=1}^{n} \phi_i$ consists of *agent safety properties* ϕ_i for each agent i.

Note that we can let $\phi_i = \phi$ for all i, so this is not a restriction. But typically we are interested in properties that can be accurately assessed in the agents' observation space (i.e., $prj_i(\phi_i) = \overline{prj}_i(\phi_i)$).

Next, we define a local shield of an agent, which, like the agent, operates in the observation space.

Definition 25 *(Local shield)* Given an n-agent LTS $\mathcal{T} = (S, Act, T)$ with observation spaces O_i and an n-agent safety property $\phi = \bigcap_{i=1}^{n} \phi_i \subseteq S$, let $\triangledown_i : O_i \to 2^{Act_i}$ be a shield for $\mathcal{T}^i$ wrt. $\overline{prj}_i(\phi_i)$, for some agent $i \in \{1, ..., n\}$, i.e., $\mathcal{T}^i \underset{\triangledown_i}{\models} \overline{prj}_i(\phi_i)$. We call $\triangledown_i$ a *local shield* of agent i.

We define an operation to turn a j-dimensional (local) shield into a k-dimensional (global) shield. This global shield allows all global actions whose projections are allowed by the local shield.

Definition 26 *(Extended shield)* Assume an n-agent LTS $\mathcal{T} = (S, Act, T)$ with projections prj_i, an n-agent safety property $\phi = \bigcap_{i=1}^{n} \phi_i \subseteq S$, and a corresponding local shield $\triangledown_i$. The *extended shield* $\uparrow(\triangledown_i)$ is defined as $\uparrow(\triangledown_i)(s) = \{a \in Act \mid a[i] \in \triangledown_i(prj_i(s))\}$.

The following definition is just syntactic sugar to ease reading.

Definition 27 Assume an LTS $\mathcal{T}$, a set of states ϕ, and a shield $\triangledown$ for ϕ. We write $\mathcal{T} \underset{\triangledown}{\models} \phi$ as an alternative to $\mathcal{T}_{\triangledown} \models \phi$.

The following lemma says that it is sufficient to have a local shield ensuring the *restricted* projection $\overline{prj}_i(\phi_i)$ of an agent safety property ϕ_i in order to guarantee safety of the extended shield.

Lemma 28 Assume an n-agent LTS $\mathcal{T}$, a safety property ϕ_i, and a local shield $\triangledown_i$ such that $\mathcal{T}^i \underset{\triangledown_i}{\models} \overline{prj}_i(\phi_i)$. Then $\mathcal{T} \underset{\uparrow(\triangledown_i)}{\models} \phi_i$.

Proof The proof is by contraposition. Assume that there is an unsafe outcome ρ in $\mathcal{T}$ (starting in a winning state) under the extended shield $\uparrow(\triangledown_i)$, i.e., ρ contains a state $s \notin \phi$. Then the projected run $prj_i(s_0)\, a[i]\, prj_i(s_1)...$ is an outcome of $\mathcal{T}^i$ under local shield $\triangledown_i$, and $prj_i(s) \notin \overline{prj}_i(\phi)$ by the definition of $\overline{prj}$. This contradicts that $\triangledown_i$ is a local shield. □

The following example shows that the *restricted* projection is necessary. Consider the LTS $\mathcal{T}$ where $S = \{0,1\}^2$, $Act = \{z, p\}^2$, and $T = \{\ ((0,0),(z,z),(0,0)),$ $((0,0),(z,p),(0,1)),\ ((0,0),(p,z),(1,0)),\ ((0,0),(p,p),(1,1))\ \}$. For $i = 1, 2$ let $\phi_i = \{(0,0),(0,1),(1,0)\}$ and prj_i project to the i-th component O_i. Then $prj_i(\phi_i) = \{0,1\} = prj_i(S)$, i.e., all states in the projection are safe, and hence a

local shield may allow $\triangledown_i(0) = \{z, p\}$. But then the unsafe state $(1, 1)$ would be reachable in $\mathcal{T}$.

If $\triangledown = \sqcap_i \uparrow (\triangledown_i)$ exists, we call it a *distributed shield*. This terminology is justified in the next theorem, which says that we can synthesize n local shields in the projections and then combine these local shields to obtain a safe shield for the global system.

Theorem 29 *(Projection-based shield synthesis)* Assume an n-agent LTS $\mathcal{T} = (S, Act, T)$ and an n-agent safety property $\phi = \bigcap_{i=1}^{n} \phi_i \subseteq S$. Moreover, assume local shields $\triangledown_i$ for all $i = 1, ..., n$. If $\triangledown = \sqcap_i \uparrow (\triangledown_i)$ exists, then $\triangledown$ is a shield for $\mathcal{T}$ wrt. ϕ (i.e., $\mathcal{T}_{\triangledown} \vDash \phi$).

Proof By definition, each local shield $\triangledown_i$ ensures that the (*restricted* projected) agent safety property ϕ_i holds in $\mathcal{T}^i$. Since $\mathcal{T}^i$ is a projection of $\mathcal{T}$, any distributed shield with i-th component $\triangledown_i$ also preserves ϕ_i in $\mathcal{T}$ (by Lemma 28). Hence, $\triangledown = \sqcap_i \uparrow (\triangledown_i)$ ensures all agent safety properties ϕ_i and thus $\phi = \bigcap_{i=1}^{n} \phi_i$. □

Unfortunately, the theorem is often not useful in practice because the local shields may not exist. The projection generally removes the possibility to coordinate with other agents. By *coordination* we do not mean (online) communication but simply (offline) agreement on "who does what." Often, this coordination is necessary to achieve agent safety. We address this lack of coordination in the next section.

3.2 Assume-Guarantee Shield Synthesis

Shielding an LTS removes some transitions. Thus, by repeatedly applying multiple shields to the same LTS, we obtain a sequence of more and more restricted LTSs.

Definition 30 *(Restricted LTS)* Assume two LTSs $\mathcal{T} = (S, Act, T), \mathcal{T}' = (S, Act, T')$. We write $\mathcal{T} \preceq \mathcal{T}'$ if $T \subseteq T'$.

Lemma 31 Let $\mathcal{T} \preceq \mathcal{T}'$ be two LTSs. Then $\mathcal{T}' \vDash \phi \Longrightarrow \mathcal{T} \vDash \phi$.

Proof As T' contains all transitions of T, it has at least the same outcomes. If no outcome of $\mathcal{T}'$ leaves ϕ, the same holds for $\mathcal{T}$. □

We now turn to the main contribution of this section. For a safety property ϕ', we assume an n-agent safety property $\phi = \bigcap_{i=1}^{n} \phi_i$ is given such that $\phi \subseteq \phi'$ (i.e., ϕ is more restrictive). We use these agent safety properties ϕ_i to filter out behavior during shield synthesis. They may contain additional guarantees, which are used to coordinate responsibilities between agents.

Crucially, in our work, the guarantees are given in a certain order. We assume *wlog* that the agent indices are ordered from 1 to n such that agent i can only rely on the safety properties of all agents $j < i$. Thus, agent i guarantees ϕ_i by assuming

$\bigcap_{j<i} \phi_j$. This is important to avoid problems with (generally unsound) circular reasoning. In particular, agent 1 cannot rely on anything, and ϕ_n is not relied on.

The theorem then states that if each agent guarantees its safety property ϕ_i, and only relies on guarantees ϕ_j such that $j < i$. The result is a (safe) distributed shield. The described condition is formally expressed as $\left(\mathcal{T}_{\triangledown^*\left[\bigcap_{j<i}\phi_j\right]}\right)^i \underset{\triangledown_i}{\models} \overline{prj}_i(\phi_i)$, where we use the most permissive shield $\triangledown^*$ for unicity.

Theorem 32 (*Assume-guarantee shield synthesis*) Assume an n-agent LTS $\mathcal{T} = (S, Act, T)$ with projections prj_i and an n-agent safety property $\phi = \bigcap_i \phi_i$. Moreover, assume (local) shields $\triangledown_i$ for all i such that $\left(\mathcal{T}_{\triangledown^*\left[\bigcap_{j<i}\phi_j\right]}\right)^i \underset{\triangledown_i}{\models} \overline{prj}_i(\phi_i)$. Then, if $\triangledown = \sqcap_i \uparrow (\triangledown_i)$ exists, it is a shield for $\mathcal{T}$ wrt. ϕ (i.e., $\mathcal{T}_\triangledown \models \phi$).

Proof Assume $\mathcal{T}$, ϕ, and local shields $\triangledown_i$ as in the assumptions. Observe that for $i = 1$, $\bigcap_{j<i} \phi_i = S$, and that $\mathcal{T}_{\triangledown^*[S]} = \mathcal{T}$. Then:

$$
\begin{aligned}
&\bigwedge_i \left(\mathcal{T}_{\triangledown^*\left[\bigcap_{j<i}\phi_j\right]}\right)^i \underset{\triangledown_i}{\models} \overline{prj}_i(\phi_i) \\
\overset{\text{Lem. 28}}{\Longrightarrow} &\bigwedge_i \mathcal{T}_{\triangledown^*\left[\bigcap_{j<i}\phi_j\right]} \underset{\uparrow(\triangledown_i)}{\models} \phi_i \overset{(*)}{\Longrightarrow} \bigwedge_i \mathcal{T}_{\sqcap_{j<i}\uparrow(\triangledown_j)} \underset{\uparrow(\triangledown_i)}{\models} \phi_i \qquad (20) \\
\overset{\text{Def. 27}}{\Longrightarrow} &\bigwedge_i \mathcal{T} \underset{\sqcap_{j\le i}\uparrow(\triangledown_j)}{\models} \phi_i \Longrightarrow \mathcal{T} \underset{\sqcap_i\uparrow(\triangledown_i)}{\models} \phi \overset{\text{Def. 27}}{\Longrightarrow} \mathcal{T}_\triangledown \models \phi
\end{aligned}
$$

Step (*) holds because the composition $\sqcap_{j\le i} \uparrow (\triangledown_j)$ of the local shields up to index i satisfy ϕ_i under the previous guarantees ϕ_j, $j < i$. Thus, $\mathcal{T}_{\sqcap_{j<i}\uparrow(\triangledown_j)} \preceq \mathcal{T}_{\triangledown^*\left[\bigcap_{j<i}\phi_j\right]}$, and the conclusion follows by applying Lemma 31. □

Finding the local safety properties ϕ_i is an art, and we leave algorithmic synthesis of these properties to future work. But we will show in our case studies that natural choices often exist, sometimes directly obtained from the (global) safety property.

4 Cascading Learning

In the previous section, we have seen how to efficiently compute a distributed shield based on assume-guarantee reasoning. In this section, we turn to the question how and under which condition we can efficiently learn multi-agent policies in a similar manner.

We start by defining the multi-agent learning objective.

Definition 33 *(n-agent cost function)* Given an n-agent MDP $\mathcal{M} = (S, Act, P)$ with projections $prj_i : S \to O_i$, an *n-agent cost function* $c = (c_1, ..., c_n)$ consists of (local) cost functions $c_i : O_i \times Act_i \to \mathbb{R}$. The total immediate cost $c : S \times Act \to \mathbb{R}$ is $c(s, a) = \sum_{i=1}^{n} c_i(prj_i(s), a[i])$ for $s \in S$ and $a \in Act$.

An agent policy is obtained by projection, analogous to a local shield. Next, we define the notion of instantiating an n-agent MDP with a policy, yielding an $(n-1)$-agent MDP.

Definition 34 *(Instantiating an agent)* Given an n-agent MDP $\mathcal{M} = (S, Act, P)$ and agent policy $\pi : O_i \times Act_i \to [0, 1]$, the *instantiated MDP* is $\mathcal{M}_\pi = (S, Act', P')$, where $Act' = Act_1 \times ... \times Act_{i-1} \times Act_{i+1} \times ... \times Act_n$ and, for all $s, s' \in S$ and $a' \in Act'$, $P'(s, a', s') = \sum_{a_i} \pi(prj_i(s), a_i) \cdot P(s, (a'[1], ..., a'[i-1], a_i, a'[i], ..., a'[n-1]), s')$.

We will need the concept of a projected, local run of an agent.

Definition 35 *(Local run)* Given a run $\rho = s_0 a_0 s_1 a_1 ...$ over an n-agent MDP (S, Act, P), the projection to agent i is the *local run* $prj_i(\rho) = prj_i(s_0)\, a_0[i]\, prj_i(s_1)\, a_1[i] ...$

Given a policy $\pi : S \times Act \to [0, 1]$, the probability of a finite local run $prj_i(\rho)$ being an outcome of π is the sum of the probabilities of outcomes of π whose projection to i is $prj_i(\rho)$.

The probability of a run ρ of length ℓ being an outcome of policy π is $Pr(\rho \mid \pi) = \prod_{i=0} \pi(s_i, a_i) \cdot P(s_i, a_i, s_{i+1})$. We say that agent i depends on agent j if agent j's action choice influences the probability for agent i to observe a (local) run.

Definition 36 *(Dependency)* Given an n-agent MDP (S, Act, P), agent i *depends* on agent j if there exists a local run $prj_i(\rho)$ of length ℓ and n-agent policies π, π' that differ only in the j-th agent policy, i.e., $\pi = (\pi_1, ..., \pi_n)$ and $\pi' = (\pi_1, ..., \pi_{j-1}, \pi_j{}', \pi_{j+1}, ..., \pi_n)$, such that the probability of observing $prj_i(\rho)$ under π and π' differ:

$$\sum_{\rho' : prj_i(\rho') = prj_i(\rho)} Pr(\rho' \mid \pi) \neq \sum_{\rho' : prj_i(\rho') = prj_i(\rho)} Pr(\rho' \mid \pi') \tag{21}$$

where we sum over all runs ρ' of length ℓ with the same projection.

In practice, we can typically perform an equivalent syntactic check. Next, we show how to arrange dependencies in a graph.

Definition 37 *(Dependency graph)* The *dependency graph* of an n-agent MDP is a directed graph (V, E) where $V = \{1, ..., n\}$ and $E = \{(i, j) \mid i \text{ depends on } j\}$.

As the main contribution of this section, Algorithm 2 shows an efficient multi-agent learning framework, which we call *cascading learning*. In order to apply the algorithm, we require an acyclic dependency graph (otherwise, an error is thrown in Line 5). Then, we train the agents in the order suggested by the dependencies, which, as we will see, leads to an attractive property.

Algorithm 2: Cascading shielded learning of n-agent policies

Input: Shielded n-agent MDP $\mathcal{M}_{\triangledown}$, n-agent cost function $c = (c_1, ..., c_n)$
Output: n-agent policy $(\pi_1, ..., \pi_n)$

1 Build dependency graph G of $\mathcal{M}_{\triangledown}$;
2 Let $\mathcal{M}' := \mathcal{M}_{\triangledown}$;
3 **while** (true)
4 **if** there is no node in G with no outgoing edges
5 error("Cyclic dependencies are incompatible.");
6 Let i be a node in G with no outgoing edges;
7 Train agent policy π_i on the MDP $\mathrm{sandbox}(\mathcal{M}', i)$ wrt. cost function c_i;
8 Update G by removing node i and all incoming edges;
9 **if** G is empty
10 **return** $(\pi_1, ..., \pi_n)$
11 Update $\mathcal{M}' := \mathcal{M}'_{\{\pi_i\}}$ ▷ I.e., instantiated shielded MDP

To draw the connection to the distributed shield, the crucial insight is that we can again use it for assume-guarantee reasoning to prevent behaviors that may otherwise create a dependency.

The procedure $sandbox(\mathcal{M}, i)$ in Line 7 takes an n-agent MDP $\mathcal{M}$ and an agent index $i \in \{1, ..., n\}$. The purpose is to instantiate every agent except agent i. Since agent i does not depend on these agents, we arbitrary choose a uniform policy for the instantiation.

Next, we show an important property of Algorithm 2: it trains policies in-distribution.

Definition 38 *(In-distribution)* Given two 1-agent MDPs $\mathcal{M} = (S, Act, P)$ and $\mathcal{M}' = (S, Act, P')$, an agent policy π is *in-distribution* if the probability of any local run in $\mathcal{M}$ is the same as in $\mathcal{M}'$.

Now we show that the distribution of observations an agent policy π_i makes during training in Algorithm 2 is identical with the distribution of observations made in $\mathcal{M}^*$, the instantiation with *all other* agent policies computed by Algorithm 2.

Theorem 39 Let $\mathcal{M}$ be an n-agent MDP with acyclic dependency graph. For every agent i, the following holds. Let $\mathcal{M}^*$ be the 1-agent MDP obtained by iteratively instantiating the original MDP $\mathcal{M}$ with policies π_j for all $j \neq i$. The agent policy π_i trained with Algorithm 2 is in-distribution wrt. $sandbox(\mathcal{M}', i)$ (from Line 7) and $\mathcal{M}^*$.

Proof Fix a policy π_i. If π_i is the last trained policy, the statement clearly holds. Otherwise, let $\pi_j \neq \pi_i$ be a policy that has not been trained at the time when π_i is trained. The algorithm asserts that π_i has no dependency on π_j. Thus, training π_i yields the same policy no matter how π_j behaves. □

Note that, despite trained in-distribution, the policies are not globally optimal. This is because each policy acts egoistically and optimizes its local cost, which may yield suboptimal global cost.

What we can show is that the agent policies $(\pi_1, ..., \pi_n)$ are *Pareto optimal* [118], i.e., they cannot all be strictly improved without raising the cost of at least one agent. That is, there is no policy π_i that can be replaced by another policy $\pi_i{}'$ without strictly increasing the expected local cost of at least one agent. Indeed:

Theorem 40 If the learning method in Line 7 of Algorithm 2 converged to the (local) optima, and these optima are unique, then the resulting policies are Pareto optimal.

Proof The proof is by induction. Assume *wlog* that the policies are trained in the order 1 to n. By assumption, π_1 is locally optimal and unique. Hence, replacing π_1 by another policy would strictly increase its total cost. Now assume we have shown the claim for the first $i-1$ agents. Algorithm 2 trained policy π_i wrt. the instantiation with the policies $\pi_1, ..., \pi_{i-1}$, and by assumption, π_i is also locally optimal and unique. Thus, again, we cannot replace π_i. □

5 Evaluation

We consider two environments with discretized state spaces.[4] All experiments were repeated 10 times; solid lines in plots represent the mean cost of these 10 repetitions, while ribbons mark the minimum and maximum costs. Costs are evaluated as the mean of $1,000$ episodes. We use the learning method implemented in UPPAAL STRATEGO [50] because the implementation has a native interface for shields. This method learns a policy by partition refinement of the state space. With this learning method, only few episodes are needed for convergence. We also compare to the (deep) MARL approach MAPPO [119] later.

[4] Available online at https://github.com/AsgerHB/N-player-shield.

5.1 Car Platoon with Adaptive Cruise Controls

Recall the car platoon model from Section 1.1. The front car follows a random distribution depending on v_n (described in [120], appendix).

The individual cost of an agent is the sum of the observed distances to the car immediately in front of it, during a 100-second episode (i.e., keeping a smaller distance to the car in front is better).

The decision period causes delayed reaction time, and so the minimum safe distance to the car in front depends on the velocity of both cars. An agent must learn to drive up to this distance, and then maintain it by predicting the acceleration of the car in front.

For this model, all agents share analogous observations O_i and safety properties ϕ_i. Hence, instead of computing $n-1$ local shields individually, it is sufficient to compute only one local shield and reuse it across all agents (by simply adapting the variables).

Relative scalability of centralized and distributed shielding

We compare the synthesis of distributed and (non-distributed) classical shields. We call the latter *centralized* shields, as they reason about the global state. Hence, they may permit more behavior and potentially lead to better policies, as the agents can coordinate to take jointly safe actions. Beside this (often unrealistic) coordination assumption, a centralized shield suffers from scalability issues. While the size of a single agent's observation space is modest, the global state space is often too large for computing a shield.

We interrupted the synthesis of a centralized shield with $n = 3$ cars (i.e., 2 agents) and a full state space after 12 hours, at which point the computation showed less than 3% progress. In order to obtain a centralized shield, we reduced the maximum safe distance from 200 to just 50, shrinking the state space significantly. Synthesizing a centralized shield took 78 minutes for this property, compared to just 3 seconds for a corresponding distributed shield.

Because of the exponential complexity to synthesize a centralized shield, we will only consider distributed shields in the following. Synthesizing a shield for a single agent covering the full safety property ($0 < d_i < 200$) took 6.5 seconds, which we will apply to a platoon of 10 cars, well out of reach of a centralized shield.

Comparing centralized, cascading and MAPPO learning

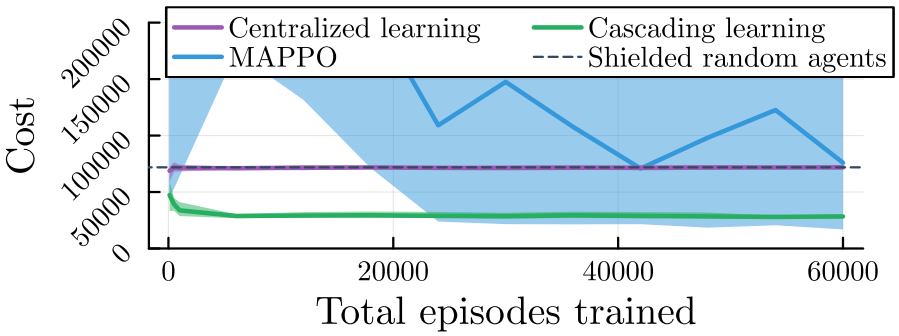


Figure 31: Graph of learning outcomes. Comparison of different learning methods on the 10-car platoon. The centralized and the MAPPO policy were trained for the total episodes indicated, while these episodes were split evenly between each agent in the cascading case.

Given a distributed shield, we consider the learning outcomes for a platoon of 10 cars (9 agents), using the learning method of UPPAAL STRATEGO. We train both a shielded *centralized* policy, which picks a joint action for all cars, and individual shielded policies using cascading learning (Algorithm 2). As expected from shielded policies, no safety violations were observed while evaluating them.

In the results shown in Figure 31, the centralized policy does not improve with more training. While it could theoretically outperform distributed policies through communication, the high dimensionality of the state and action space likely prevents that. It only marginally improves over the random baseline, which has an average cost of 71 871. On the other hand, cascading learning quickly converges to a much better cost as low as 26 435.

To examine how cascading learning under a distributed shield compares to traditional MARL techniques, we implemented the platoon environment in the benchmark suite BenchMARL [121] and trained an unshielded policy with MAPPO [119], a state-of-the-art MARL algorithm based on PPO [122], using default hyperparameters. To encourage safe behavior, we added a penalty of 1 600 to the cost function for every step upon reaching an unsafe state. (This value was obtained by starting from 100 and doubling it until safety started degrading again.) Recall that shielded agents are safe.

We include the training outcomes for MAPPO in Figure 31. Due primarily to the penalty of safety violations, the agents often have a cost greater than 100 000, even at the end of training. However, the best MAPPO policy achieved a cost of just 16 854, better than the cascading learning method. We inspected that policy and found that the cars drive very closely, accepting the risk of a crash. Overall, there is a large variance of the MAPPO policies in different runs, whereas cascading

learning converges to very similar policies, and does so much faster. This is likely because of the smaller space in which the policies are learned, due to the distributed shield. Thus, cascading learning is more effective.

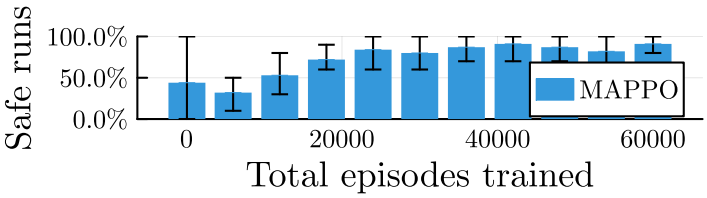


Figure 32: Percentage of safe runs with the MAPPO policy in the 10-car platoon. Blue bars show the mean of 10 repetitions, while black intervals give min and max values.

Since the MAPPO policy is not safe by construction, Figure 32 shows the percentage of safe episodes, out of 1 000 episodes. The agents tend to be safer with more training, but there is no inherent guarantee of safety, and a significant amount of violations remain.

5.2 Chemical Production Plant

In the second case study, we demonstrate that distributed shielding applies to complex dependencies where agents influence multiple other agents asymmetrically. We consider a network of inter-connected chemical production units, each with an internal storage.

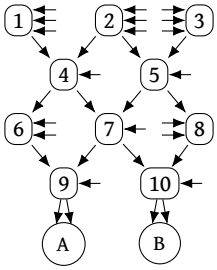


Figure 33: Layout of plant network.

Figure 33 shows the graph structure of the network. Numbered nodes (1 to 10) denote controlled production units, while letter-labeled nodes (A, B) denote uncontrolled consumers with periodically varying demand. Arrows from source to target nodes denote potential flow at no incurred cost. Arrows without a source node denote potential flow from external providers, at a cost that individually and periodically varies. Consumption patterns and examples of the cost patterns are shown in the appendix of [120]. The flow rate in all arrows follows a uniform random distribution in the range $[2.15; 3.15]$ ℓ/s.

Each agent i is associated with a production unit (1 to 10), with internal storage volume v_i. Beside a global periodic timer, each agent can only observe its own volume. At each decision period of 0.5 seconds, an agent can open or close each of the three input flows (i.e., there are $|Act_i| = 9$ actions per agent and hence $|Act| = 9^{10}$ global actions), but cannot prevent flow from outgoing connections.

The individual cost of an agent is incurred by buying from external providers. Agents must learn to take free material from other units, except for agents 1 to 3, which instead must learn to buy from their external providers periodically when the cost is low.

Units must not exceed their storage capacity, and units 9 to 10 must also not run empty to ensure the consumers' demand is met. That is, the safety property is $\phi = \{s \mid \bigwedge_i v_i < 50 \wedge 0 < v_9 \wedge 0 < v_{10}\}$.

Shielding.

The property $0 < v_9$ cannot be enforced by a local shield for agent 9 without additional assumptions that the other agents do not run out. This is because the (single) external provider is not enough to meet the potential (dual) demand of consumer A. This yields the local safety properties $\phi_i = \{s \mid 0 < v_i < 50\}$. Here, agents 1 to 3 do not make assumptions, while agents 4 and 5 depend on agents 1 to 3 not running out, etc. For this model, we do not use the same shield for all agents, since they differ in the number of outgoing flows (either 1 or 2). Still, it is sufficient to compute two types of shields, one for each variant, and adapt them to analogous agents. Computing a centralized shield would again be infeasible, while computing the distributed shield took less than 1 second.

Comparing centralized and cascading learning

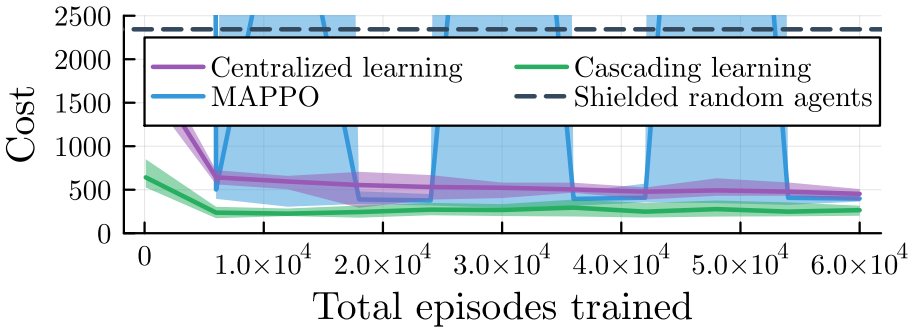


Figure 34: Comparison of different learning methods on the chemical production plant. The centralized policy was trained for the total episodes indicated, while these episodes were split evenly between each agent in the cascading case.

Thanks to the guarantees given by the distributed shields, agents 9 to 10 are only affected by the behavior of the consumers, agents 6 to 8 only depend on agents 9 to 10, etc. Thus, the agent training order is 10, 9, 8 ...

We compare the results of shielded cascading learning, shielded centralized learning, MAPPO, and shielded random agents in Figure 34. Centralized learning achieved a cost of 292. The lowest cost overall, 172, was achieved by cascading learning. We compare this to the (unshielded) MAPPO agents, whose lowest cost was 291. More background information is given in [120].

6 Conclusion

In this paper, we presented distributed shielding as a scalable MA approach, which we made practically applicable by integrating assume-guarantee reasoning. We also presented cascading shielded learning, which, when applicable, is a scalable MARL approach. We demonstrated that distributed shield synthesis is highly scalable and that coming up with useful guarantees is reasonably simple.

While we focused on demonstrating the feasibility in this work by providing the guarantees manually, a natural future direction is to learn them. As discussed, this is much simpler in the classical setting [96] because the agents/components are fixed. We believe that in our setting where both the guarantees and the agents are not given, a trial-and-error approach (e.g., a genetic algorithm) is a fruitful direction to explore. Another relevant future direction is to generalize our approach to continuous systems [10].

This research was partly supported by the Independent Research Fund Denmark under reference number 10.46540/3120-00041B, DIREC - Digital Research Centre Denmark under reference number 9142-0001B, and the Villum Investigator Grant S4OS under reference number 37819.

Paper D:

UPPAAL COSHY: Automatic Synthesis of Compact Shields for Hybrid Systems

Asger Horn Brorholt
Department of Computer Science
Aalborg University, Aalborg, Denmark

Andreas Holck Høeg-Petersen
Department of Computer Science
Aalborg University, Aalborg, Denmark

Peter Gjøl Jensen
Department of Computer Science
Aalborg University, Aalborg, Denmark

Kim Guldstrand Larsen
Department of Computer Science
Aalborg University, Aalborg, Denmark

Marius Mikučionis
Department of Computer Science
Aalborg University, Aalborg, Denmark

Christian Schilling
Department of Computer Science
Aalborg University, Aalborg, Denmark

Andrzej Wąsowski
Department of Computer Science
IT University of Copenhagen, Copenhagen, Denmark

Abstract

We present UPPPAAL COSHY, a tool for automatic synthesis of a safety strategy—or *shield*—for Markov decision processes over continuous state spaces and complex hybrid dynamics. The general methodology is to partition the state space and then solve a two-player safety game [10], which entails a number of algorithmically hard problems such as reachability for hybrid systems. The general philosophy of UPPPAAL COSHY is to approximate hard-to-obtain solutions using simulations. Our implementation is fully automatic and supports the expressive formalism of UPPAAL models, which encompass stochastic hybrid automata.

The precision of our partition-based approach benefits from using finer grids, which however are not efficient to store. We include an algorithm called CAAP to efficiently compute a compact representation of a shield in the form of a decision tree, which yields significant reductions.

1 Introduction

In prior work, we proposed an algorithm to synthesize *shields* (i.e., nondeterministic safety strategies) for Markov decision processes with hybrid dynamics [10]. The algorithm partitions the state space into finitely many cells and then solves a two-player safety game, where it uses approximation through simulation to efficiently tackle algorithmically hard problems. In this tool paper, we present our implementation UPPAAL COSHY, which is fully integrated in UPPAAL, offering an automatic tool[5] that supports the expressive UPPAAL modeling formalism, including reinforcement learning under a shield.

Our algorithm represents a shield by storing the allowed actions for each cell individually, which results in a large data structure. Since many neighboring cells allow the same actions in practice, as a second contribution, we propose a new algorithm called CAAP to compute a compact representation in the form of a decision tree. We demonstrate that this algorithm leads to significant reductions as part of the workflow in UPPAAL COSHY.

An extended version of this paper is available online [123].

1.1 Related Tools for Shield Synthesis and Compact Representation

Shielding.
Shields are obtained by solving games, for which there exist a wide selection of tools for discrete state spaces [124], [125], [126]. Notably, TEMPEST [127] synthesizes shields for discrete systems and facilitates learning through integration with PRISM [128]. UPPAAL TIGA synthesizes shields for timed games [129].

In contrast, our tool applies to a richer class of models, including stochastic hybrid systems with non-periodic control and calls to external C libraries.

One benefit of our tool is the full integration with UPPAAL STRATEGO [130] to directly use the synthesized shield in reinforcement learning.

Decision trees.
Encoding strategies as decision trees is a popular approach to achieving compactness and interpretability [131], [132], [133], [134], [135]. However, these works focus on creating approximate representations from tabular data. For a fixed set of predicates, the smallest possible tree can be obtained by enumeration techniques [136], [137]. In contrast, our method transforms a given decision tree into an *equivalent* decision tree. Our method is specifically designed to efficiently cope with strategies of many axis-aligned decision boundaries.

[5] Available at https://uppaal.org/features/#coshy

2 Shield Synthesis for Hybrid Systems

In this section, we recall a general shield synthesis algorithm for hybrid systems outlined in prior work [10]. We start by recalling the formalism for control systems.

2.1 Euclidean Markov Decision Processes

Definition 1 A k-dimensional *Euclidean Markov decision process* (EMDP) is a tuple $\mathcal{M} = (S, A, T)$ where

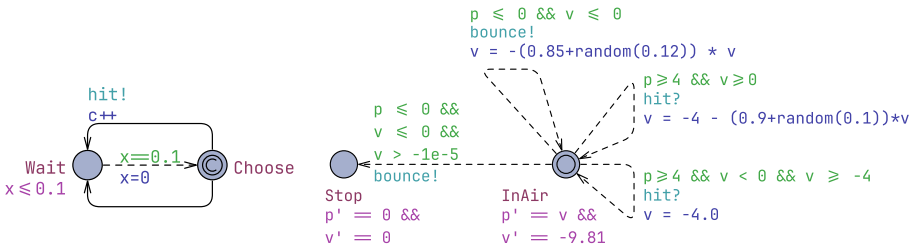
- $S \subseteq \mathbb{R}^k$ is a closed and bounded subset of the k-dimensional Euclidean space,
- A is a finite set of actions, and
- $T : S \times A \to (S \to \mathbb{R}_{\geq 0})$ maps each state-action pair (s, a) to a probability density function over S, i.e., we have $\int_{s' \in S} T(s, a)(s')ds' = 1$.

For simplicity, the state space S is continuous. However, the extension to discrete variables, e.g., locations of hybrid components, is straightforward. Since optimizing strategies is not our focus, we do not formally introduce the notion of cost and rely on the reader's intuition. (See [10] for details.)

A *run* π of an EMDP is an alternating sequence $\pi = s_0 a_0 s_1 a_1 \ldots$ of states and actions such that $T(s_i, a_i)(s_{i+1}) > 0$ for all $i \geq 0$. A (memoryless) stochastic *strategy* for an EMDP is a function $\sigma : S \to (A \to [0, 1])$, mapping a state to a probability distribution over the actions. A run $\pi = s_0 a_0 s_1 a_1 \ldots$ is an *outcome* of σ if $\sigma(s_i)(a_i) > 0$ for all $i \geq 0$. Similarly, a (memoryless) nondeterministic strategy is a function $\sigma : S \to 2^A$, mapping a state to a set of actions. A run $\pi = s_0 a_0 s_1 a_1 \ldots$ is an outcome of σ if $a_i \in \sigma(s_i)$ for all $i \geq 0$.

A *safety property* (or invariant) φ is a set of states $\varphi \subseteq S$. A run $\pi = s_0 a_0 s_1 a_1 \ldots$ is *safe* with respect to φ if $s_i \in \varphi$ for all $i \geq 0$. A nondeterministic strategy σ is a *shield* with respect to φ if all outcomes of σ are safe.

2.2 Running Example (Bouncing Ball)



(a) Player component. (b) Ball component.

Figure 35: The *bouncing ball* modeled in UPPAAL.

We introduce our running example: a *bouncing ball* that can be hit by a player to keep it bouncing [10], [50]. We shortly explain our two-component UPPAAL model. The player component is shown in Figure 35a. In the (initial) location `Choose`, there are two available control actions (solid lines). The player chooses every 0.1 seconds (enforced by the clock `x`). The action (upper edge) attempts to hit the ball, and increments the cost counter `c` to be used for reinforcement learning in Section 5.1. The other action (lower edge) does not attempt to hit the ball.

The ball component, shown in Figure 35b, is described by two state variables, position `p` and velocity `v`, which evolve according to the ordinary differential equations shown below the initial location `InAir`. The two dashed edges on the right model a successful `hit` action, which is only triggered if the ball is high enough (four meters or higher above the ground); they differ in whether the ball is currently jumping up or falling down. The two dashed edges on the left model a `bounce` on the ground. The ball bounces back up with a random dampening (upper edge) or goes to the state `Stop` if the velocity is very low (lower edge). In the following, we shall see how to obtain a shield that enforces the safety property that `Stop` is never reached, i.e., $\varphi = \{s \mid \text{Ball is not in } \texttt{Stop} \text{ in } s\}$.

2.3 Partition-Based Shield Synthesis

Since an EMDP consists of infinitely many states, we employ a finite-state abstraction. For that, we partition the state space $S \subseteq \mathbb{R}^k$ with a regular *rectangular* grid. (In [10], we only allowed a grid of uniform size in all dimensions.) Formally, given a (user-defined) granularity vector $\gamma \in \mathbb{R}^k$ and offset vector $\omega \in \mathbb{R}^k$, we partition the state space into disjoint *cells* of equal size. Each cell C is the Cartesian product of half-open intervals $[\omega_i + p_i\gamma_i; \omega_i + (p_i+1)\gamma_i[$ in each dimension i, for cell index $p \in \mathbb{N}^k$. We define the *grid* as the set $\mathcal{P}_\gamma^\omega = \{C \mid C \cap S \neq \emptyset\}$ of all cells that overlap with the bounded state space. Note the number of cells will depend on γ. For each $s \in S$, $[s]_{\mathcal{P}_\gamma^\omega}$ denotes the unique cell containing s.

An EMDP $\mathcal{M}$, a granularity vector γ and offset vector ω induce a finite labeled transition system $\mathcal{T}_{\mathcal{M},\gamma,\omega} = \left(\mathcal{P}_\gamma^\omega, A, \to\right)$, where

$$C \xrightarrow{a} C' \Leftrightarrow \exists s \in C.\, \exists s' \in C'.\, T(s,a)(s') > 0. \tag{22}$$

Given a safety property $\varphi \subseteq S$ and a grid $\mathcal{P}_\gamma^\omega$, let $\mathcal{C}_\varphi^0 = \{C \in \mathcal{P}_\gamma^\omega \mid C \subseteq \varphi\}$ denote those cells that are safe in zero steps. We define the set of *safe cells* as the maximal set $\mathcal{C}_\varphi$ such that

$$\mathcal{C}_\varphi = \mathcal{C}_\varphi^0 \cap \{C \in \mathcal{P}_\gamma^\omega \mid \exists a \in A.\, \forall C' \in \mathcal{P}_\gamma^\omega.\, C \xrightarrow{a} C' \implies C' \in \mathcal{C}_\varphi\}. \tag{23}$$

Given the finiteness of $\mathcal{P}_\gamma^\omega$ and monotonicity of Equation 23, $\mathcal{C}_\varphi$ may be obtained in a finite number of iterations using Tarski's fixed-point theorem [53].

A (nondeterministic) strategy for $\mathcal{T}_{\mathcal{M},\gamma,\omega}$ is a function $\nu : \mathcal{P}_\gamma^\omega \to 2^A$. The most permissive shield ν_φ (i.e., safe strategy) obtained from $\mathcal{C}_\varphi$ [54] is given by

$$\nu_\varphi(C) = \{a \in A \mid \forall C' \in \mathcal{P}_\gamma^\omega.\, C \xrightarrow{a} C' \implies C' \in \mathcal{C}_\varphi\}. \tag{24}$$

A shield ν for $\mathcal{T}_{\mathcal{M},\gamma,\omega}$ induces a shield σ for $\mathcal{M}$ in the standard way [10]:

Given an EMDP $\mathcal{M}$, a safety property $\varphi \subseteq S$, and a grid $\mathcal{P}_\gamma^\omega$, if ν is a shield for $\mathcal{T}_{\mathcal{M},\gamma,\omega}$, then $\sigma(s) = \nu\left([s]_{\mathcal{P}_\gamma^\omega}\right)$ is a shield for $\mathcal{M}$.

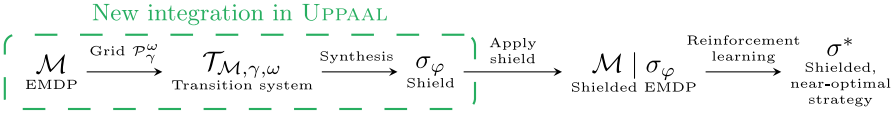
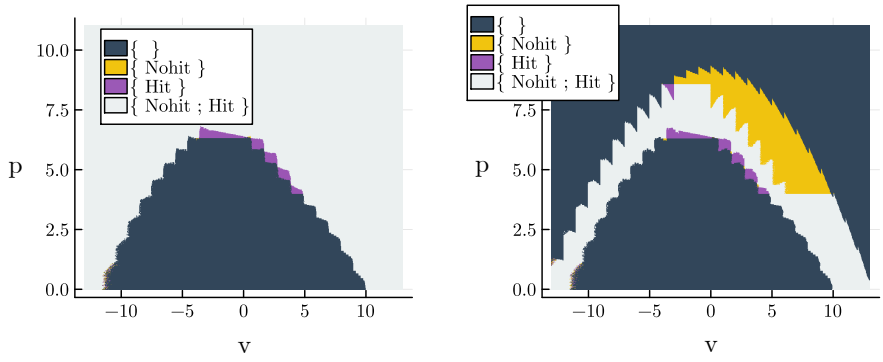


Figure 36: Workflow for obtaining a near-optimal shielded strategy in UPPAAL.

Figure 36 shows the overall workflow of the shield synthesis and how the shield can later be used to (reinforcement-) learn a near-optimal strategy *under this shield*. The green box marks the steps that we newly integrated in UPPAAL.

For the *bouncing ball*, we will obtain the shield shown in Figure 37a. To effectively implement the aforementioned approach, there are additional challenges which we address in the following section.



(a) It is safe to leave the bounds.

(b) It is unsafe to leave the bounds.

Figure 37: Two shields for the *bouncing ball*. Colors represent the allowed actions in the corresponding state of velocity v and position p while in location `InAir`.

3 Effective Implementation of Shield Synthesis

In this section, we discuss our implementation of the approach to synthesize a shield as outlined in Section 2 in UPPAAL COSHY. In particular, a practical implementation faces the following two main challenges.

First, we receive the safety property φ in the form of a user query (see Section 5.1). Thus, the definition of the cells $\mathcal{C}_\varphi^0$ that are immediately safe generally requires

symbolic reasoning, which is not readily available. Instead, we check a finite number of states within each cell, which we describe in Section 3.1.

Second, determining Equation 22 requires to solve reachability questions for infinitely many states. While this can be done for simple classes of systems, we deal with very general systems (e.g., nonlinear hybrid dynamics), for which reachability is undecidable [58]. This motivated us to instead compute an approximate solution, which we outline in Section 3.2.

Thanks to the above design decisions, our implementation is fully automatic and supports the expressive formalism of general UPPAAL models (e.g., stochastic hybrid automata with calls to general C code).

We also identified further practical challenges, which we address in the later parts of this section. Definition 1 requires a bounded state space, but it is for instance difficult to determine upper bounds for the position and velocity of the *bouncing ball* in Section 3.3, we explain how we treat such cases in practice. In Section 3.4, we discuss an optimization to omit redundant dimensions.

3.1 Determining Initial Safe Cells

We apply *systematic sampling* from a cell, i.e., samples are not drawn at random. Rather, we uniformly cover the cell with n^k samples, where $n \in \mathbb{N}, n \neq 0$ is a user-defined parameter. Recall from Section 2.3 that a cell C of a grid $\mathcal{P}_\gamma^\omega$ is rectangular and defined by an index vector p, an offset ω and a granularity vector γ, all of dimension k. Let $\delta_i = \frac{\gamma_i}{n-1}$ be the distance between two samples in dimension i when $n > 1$, and $\delta_i = 0$ otherwise. For any cell, we define the corresponding set of samples as $\{(\omega_1 + p_1\gamma_1 + q_1\delta_1, ..., \omega_k + p_k\gamma_k + q_k\delta_k) \mid q_i \in \{0, 1, ..., n-1\}\}$. To account for the open upper bounds, we subtract a small number $\epsilon > 0$ from the highest samples. An example of a two-dimensional set of samples for $n = 4$ is shown as the dark blue points inside the light blue cells in Figure 38.

We note that the above only applies to continuous variables. Our implementation treats discrete variables (e.g., component locations) in the natural way.

Finally, to approximate the set $\mathcal{C}_\varphi^0$, we draw samples from each cell and check for each sample whether it violates the specification. A cell is added to $\mathcal{C}_\varphi^0$ only if all samples in that cell satisfy the specification.

For the *bouncing ball*, the ball should never be in the `stop` location. Since the location is a discrete variable, and each cell only belongs to one location, checking a single sample from a cell C already determines whether $C \in \mathcal{C}_\varphi^0$. Thus, our approach is exact and efficient in the common case where the safety property is given via an error location.

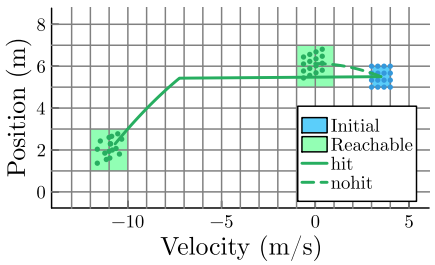
3.2 Determining Reachability

We approximate cell reachability $C \xrightarrow{a} C'$, as defined in Equation 22, similarly to [10] but adapted to work in Uppaal. In a UPPAAL model, actions $a \in A$ correspond to controllable edges (indicating that the controller can act).

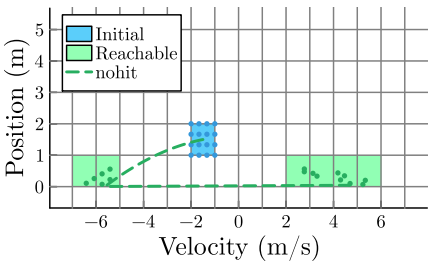
For each cell C and action $a \in A$, we iterate over all sampled states s (as described before) and select the edge corresponding to a, which gives us a new state s'; starting from s', we simulate the environment (using the built-in simulator in UPPAAL) until a state s'' is reached in which the controller has the next choice (i.e., multiple action edges are enabled) again.[6] Thus, s'' is a witness to add the corresponding cell $[s'']_{\mathcal{P}^\omega_\gamma}$ to the transition relation $C \xrightarrow{a} [s'']_{\mathcal{P}^\omega_\gamma}$. Assuming the simulator is numerically sound, the resulting transition system underapproximates $\mathcal{T}_{\mathcal{M},\gamma,\omega}$. As observed in [10], the more simulations are run, the more likely do we obtain the true solution. To check whether this underapproximation is sufficiently accurate, the existing queries for statistical model checking in UPPAAL can be used, as we shall see in Section 5.

In general, two simulations starting in the state s may not yield the same state s'' due to stochasticity. In [10], stochasticity was treated as additional dimensions over which to sample (systematically). This was possible by manually crafting the reachability sampling for each model. Detecting stochastic behavior in UPPAAL models automatically turned out to be difficult due to the rich formalism. Instead, we decided to simply let the simulator sample from the stochastic distribution. As a side effect, this new design allows us to support stochasticity with general distributions, particularly with unbounded support.

Since this design may generally miss some corner-case behavior, we expose a user-defined parameter m to control the number of times sampling is repeated.



(a) The ball is rising and high enough to be hit. When the ball is hit, the outcome is partially random.



(b) The ball is too low to be hit, but it bounces off the ground. The velocity loss upon a bounce is partially random.

Figure 38: Example of a grid for the *bouncing ball*. By sampling from the initial cell (blue) and simulating the dynamics, we discover reachable cells (green).

[6] Where [10] required a fixed control period, UPPAAL COSHY supports non-periodic control. This is demonstrated in [123].

We illustrate the reachability approximation for the *bouncing ball* in Figure 38 for $n = 4$ (number of samples per dimension) and $m = 1$ (number of re-sampling). When the ball moves through the air, it behaves deterministically. In Figure 38a, when the ball is not hit, we obtain successor states that keep a regular "formation" (top right green dots). When the ball is hit, the successor states are affected by randomness (bottom left green dots). Figure 38b shows a similar randomized effect when the ball touches the ground.

3.3 Generalization to Unbounded State Spaces

Definition 1 requires the state space to be bounded, but bounds can be hard to determine for some systems. This includes the *bouncing ball*, for which upper bounds for position and velocity are not immediately clear. Indeed, if we consider the bounded state space where $p \in [0; 11]$ and $v \in [-13; 13]$, the system dynamics do not guarantee that the ball stays within these bounds. If we plot velocity against position, as in Figure 37, then a falling ball near the left end of the plot may leave the bounds on the left (because it becomes too fast).

Conceptually, our implementation deals with out-of-bounds situations by modifying the transition system. All samples leading to a state outside the specified bounds go to a dummy cell C_{out}, for which all transitions lead back to itself. A user-defined option with the following choices determines the behavior:

1 Raise an error when reaching C_{out} during simulation (default behavior).
2 Include C_{out} in C^0_φ, i.e., leaving the bounds is always safe.
3 Exclude C_{out} from C^0_φ, i.e., leaving the bounds is always unsafe.
4 Automatically choose between Line 2 and Line 3 using sampling.

With Line 4, samples are taken outside the specified bounds, similar to Section 3.1. For the *bouncing ball*, our tool samples states such as $(v = 26, p = 22,$ `Ball.Stop`$)$, even though these states may not be reachable in practice. If any sample state is found to be unsafe, C_{out} is considered unsafe, and safe otherwise. The result of synthesizing a shield with this option is shown in Figure 37b. In particular, that shield forbids to hit the ball when it is too fast, which ensures that it does not leave the bounds. Alternatively, we obtain a more permissive shield by choosing Line 2, as shown in Figure 37a.

3.4 Omitting Variables from Consideration

As emphasized in [9], a shield can be obtained from an abstract model that only simulates behaviors relevant to the safety specification. For example, cost variables may only be relevant during learning. While every variable in a model can be included in the partitioning, this is computationally demanding.

Therefore, we allow that variables are omitted from the grid specification. However, this raises a new challenge when sampling a state from a cell, since a concrete state requires a value for each variable. To address that, we set each omitted

variable to the unique value of the initial state, which must always be specified in a UPPAAL model. Hence, the user must define the initial state such that the values of omitted variables are sensible defaults. (Note that the initial state is ignored by the shield synthesis in all other respects.)

The choice not to include a variable in the grid must be made carefully, as this can change the behavior of the transition system and potentially lead to an unsound shield. As a rule of thumb, it is appropriate to omit variables if they always have the same value when actions are taken, or if they are only relevant for keeping track of a performance value such as cost.

For the *bouncing ball*, the player (Figure 35a) is always in the location `Choose` when taking an action. By setting `Choose` as the initial location, this component's location is not relevant to keep track of in the partitioning. Moreover, the variable is used to keep track of cost and does not matter to safety. Lastly, the clock variable is used to measure time until the next player action. It is always 0 when it is time for the player to act, and so it can also be omitted.

4 Obtaining a Compact Shield Representation

In this section, we present a new technique for obtaining a compact representation of shields that stem from an axis-aligned state-space partitioning (as described in Section 2.3). Here, we choose to represent the shield as a decision tree. We note that we aim for a functionality-preserving representation, i.e., we transform a grid-based shield to an equivalent decision-tree-based shield.

Recall that each cell prescribes a set of allowed actions. Let two cells be *similar* if the shield assigns the same set of actions to them. Our goal is to form (hyper)rectangular clusters of similar cells, which we call *regions* in other words, we aim to find a coarser partitioning. In a nutshell, our approach works as follows. Initially, we start from the finest partitioning where each cell is a separate region. Then, we iteratively merge neighboring regions of similar cells, thereby obtaining a coarser partitioning, such that the resulting region is rectangular again. We call our algorithm CAAP (**C**oarsify **A**xis-**A**ligned **P**artitionings).

4.1 Representation of Partitionings and Regions

We start by noting that an axis-aligned partitioning of a state space $S \subseteq \mathbb{R}^k$ can be represented by a binary decision tree $\mathcal{T}$ where each leaf node is a set of actions and each inner node splits the state space with a predicate of the form $\rho(s) = s_i < c$, where s is a state vector, s_i is a state dimension, and $c \in \mathbb{R}$. Given a state s, the tree evaluation, written $\mathcal{T}(s)$, is defined as usual: Start at the root node. At an inner node, evaluate the predicate $\rho(s)$. If $\rho(s) = \top$, descend to the left child; otherwise, descend to the right child. At a leaf node, return the corresponding set of actions. We denote the partitioning induced by a decision tree $\mathcal{T}$ as $\mathcal{P}_\mathcal{T}$. Our goal in this section is: given a decision tree $\mathcal{T}$ inducing a partitioning $\mathcal{P}_\mathcal{T}$, find an equivalent but smaller decision tree.

Given a tree $\mathcal{T}$, we store all bounds c of the predicates $s_i < c$ in a matrix M of k rows where the i-th row contains the bounds associated with state dimension s_i in ascending order. For example, consider the bounds in Figure 39a and M in Equation 25.

$$\begin{array}{c} \\ s_1 \\ s_2 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \left[\begin{array}{cccc} 0 & 2 & 3 & 4 \\ 0 & 2 & 3 & 4 \end{array}\right] \end{array} \tag{25}$$

We extract a bounds vector from M via an index vector $p \in \mathbb{N}^k$ such that the i-th entry of p contains the column index for the i-th row. In other words, the resulting vector consists of the values M_{i,p_i}. For instance, $p = (1, 3)$ yields the vector $s^p = (0, 3)$ (row s_1 column 1 and row s_2 column 3). We can view this vector as a state in the state space given as $s^p = \left(M_{1,p_1}, ..., M_{k,p_k}\right)$.

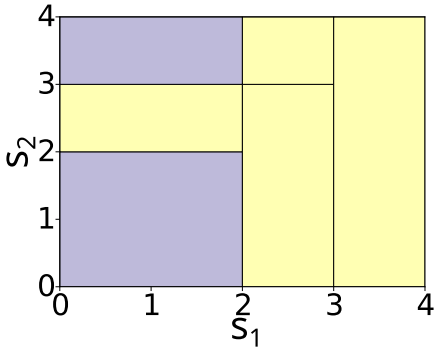
We define a region R in terms of two index vectors $\left(p^{\min}, p^{\max}\right)$ representing the minimal and maximal corner in each dimension. Then, increasing $p_i^{\max}$ corresponds to expanding R in dimension i.

4.2 Expansion of Rectangular Regions

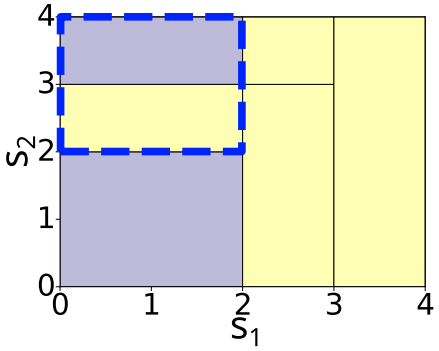
For an expansion to be legal, it must satisfy the following three *expansion rules*:

Definition 2 Let R' be a candidate region for a new partitioning $\mathcal{P}'$ derived from $\mathcal{P}_\mathcal{T}$. Then R' is legal if it satisfies these three rules:

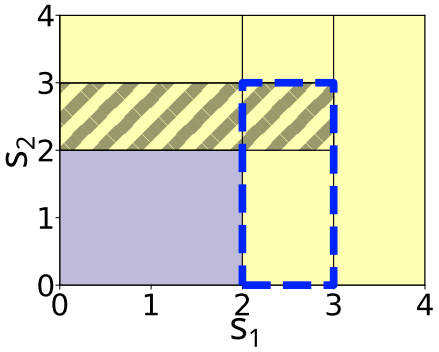
1. All cells in region R' have the same action set,
2. Region R' does not intersect with other regions in $\mathcal{P}'$,
3. Region R' does not cut any other region R from the original partitioning $\mathcal{P}_\mathcal{T}$ in two, i.e., the difference $R \setminus R'$ is either empty or rectangular.



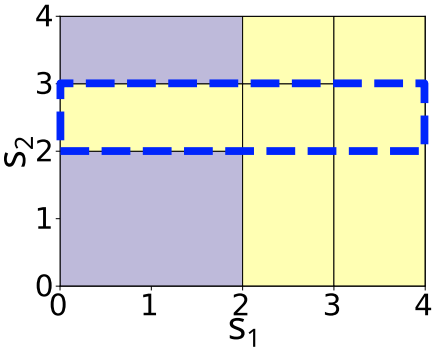
(a) An input partitioning.



(b) A violation of Rule 1, since the expanded region contains different actions.



(c) A violation of Rule 2, since the expanded region overlaps with a striped area.



(d) A violation of Rule 3, since the expansion cuts the rightmost region into two new regions.

Figure 39: Expansion example. Yellow and purple denote distinct actions. Striped regions have been fixed in previous iterations. The dashed border is the new candidate region R'.

The first two cases are directly related to the definition of the problem, i.e., the produced partitioning should respect $\mathcal{T}$ and only have non-overlapping regions (see Figure 39b and Figure 39c). The third case is required in order to ensure that in each iteration, the algorithm does not increase the overall number of regions when adding a region from the original partitioning to the new partitioning. To appreciate this, consider the visualization in Figure 39d. The candidate expansion cuts the rightmost region (given by $(3, 0)$ and $(4, 4)$) in two such that the remainder would have to be represented by *two* regions — one given by $((3, 0), (4, 2))$ and one given by $((3, 3), (4, 4))$. Clearly, all three expansion rules of Definition 2 can be checked in time linear in the number of nodes of $\mathcal{P}_{\mathcal{T}}$.

To determine the expansion of regions, we propose the following greedy approach: let $(p^{\min}, p^{\max})$ define a region. We then want to find a vector $\Delta_p \in \mathbb{Z}^k$ such that $\left(p^{\min}, p^{\min} + \Delta_p\right)$ defines a region that obeys the three expansion rules and is (locally) maximal, in the sense that increasing it in any dimension would violate at least one of the expansion rules. Note that a vector $\Delta_p = p^{\max} - p^{\min}$ satisfies the expansion rules trivially but is possibly not maximal. Thus, a solution is guaranteed to exist. However, note that there is not necessarily a unique maximal solution, and that the set of solutions is not convex, i.e., there may exist solutions Δ_p^1 and Δ_p^2 such that $\Delta_p^1 \leq \Delta_p^2$ but no other ${\Delta_p}'$ with $\Delta_p^1 \leq {\Delta_p}' \leq \Delta_p^2$ satisfies the expansion rules. Formally:

Definition 3 Given $p^{\min} \in \mathbb{Z}^k$, a decision tree $\mathcal{T}$ over a k-dimensional state space, and a set $\mathcal{P}$ of fixed regions, $\Delta_p \in \mathbb{Z}^k$ is a vector such that for $p^{\max} = p^{\min} + \Delta_p$ the region $R = (p^{\min}, p^{\max})$ does not violate any of the expansion rules in Definition 2 and for any vector ${\Delta'}_p = \left(\Delta_{p_1}, ..., \Delta_{p_i} + 1, ..., \Delta_{p_k}\right)$ at least one of the rules is violated.

Our greedy approach to finding Δ_p starts with $\Delta_p = p^{\max} - p^{\min}$ for some region $R = (p^{\min}, p^{\max})$. It then iteratively selects a dimension d by a uniformly random choice and attempts to increment the d-th entry of Δ_p. For that, we define the candidate region $R' = \left(p^{\min}, p^{\min} + \Delta_p\right)$ and check the Line 1 Line 2. If any of them is violated, we mark the corresponding dimension d as exhausted, roll back the increment, and continue with a new dimension not marked as exhausted yet, until none is left.

As mentioned above, the set of solutions is not convex. Correspondingly, if Line 3 is violated, the algorithm initiates an attempt at *repairing* the candidate expansion by continuing the expansion to the largest bound in the expansion dimension of any of the broken regions. This way, we check whether the violation can be overcome by simply expanding more aggressively. When all dimensions have been exhausted, Δ_p adheres to Definition 3.

We note that the algorithm is not guaranteed to find a local optimum. One reason is that the repair only expands in one dimension. This choice is deliberate to keep the algorithm efficient and avoid a combinatorial explosion. A more detailed description including pseudocode can be found in [123].

5 Case Studies and Evaluation

In this section, we evaluate our implementation of UPPAAL COSHY and CAAP. In Section 5.1, we demonstrate a typical application. In Section 5.2, we benchmark the implementations on several models.

5.1 A Complete Run of the Bouncing Ball

#	Query	Result
1	`strategy efficient = minE(c) [⩽120] {} → {v, p} : <> time⩾120`	✓
2	`simulate [⩽120]{ p, v } under efficient`	✓
3	`E[⩽120;100] (max: c) under efficient`	≈ 0
4	`Pr[⩽120;10000] (<> Ball.Stop) under efficient`	$[0.9995; 1]$
5	`strategy shield = acontrol: A[] !Ball.Stop { v[-13, 13]:1300, p[0, 11]:550, Ball.location }`	✓
6	`saveStrategy("shield.json", shield)`	✓
7	`strategy compact_shield = loadStrategy("compact.json")`	✓
8	`simulate [⩽120]{ p, v } under compact_shield`	✓
9	`strategy shielded_efficient = minE(c) [⩽120] {} → {v, p} : <> time⩾120 under compact_shield`	✓
10	`simulate [⩽120]{ p, v } under shielded_efficient`	✓
11	`E[⩽120;100] (max: c) under shielded_efficient`	34.6 ± 0.6
12	`Pr[⩽120;10000] (<> Ball.Stop) under shielded_efficient`	$[0; 0.00053]$

Table 9: Queries run on the *bouncing ball* model. New query type highlighted. All statistical results are given with a 99% confidence interval.

Table 9 shows a typical usage of UPPAAL with a sequence of queries on the *bouncing ball* example to produce a safe and efficient strategy (cf. Figure 36). Documentation of the new query syntax is available online and in [123].[7]

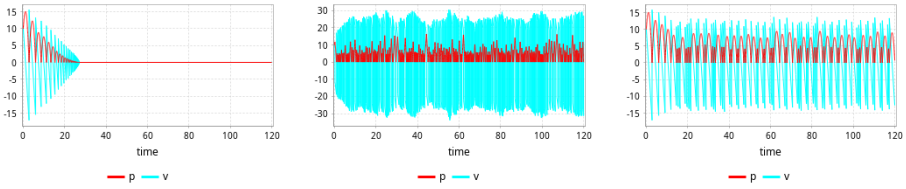
In Query 1, we train a strategy called , which is only concerned with cost and does not consider safety. Such a strategy is trivial: simply never pick the `hit` action. This is seen in Query 2, which simulates a single run of 120 seconds. It outputs position and velocity , which are visualized in Figure 40a. Query 3 statistically evaluates the strategy in 100 runs to estimate the expected value of . The result "≈ 0" indicates that only this value was observed. Query 4 estimates the probability of a run being unsafe to be in the interval $[0.9995; 1]$ with 99% confidence; in this case, as expected, all 10 000 runs were unsafe.

[7] https://docs.uppaal.org/language-reference/query-syntax/controller_synthesis/#approximate-control-queries

Query 5 synthesizes a shield . The shield matches the one shown in Figure 37a. In queries 6 and 7, the shield is converted to a compact representation by saving it to a file, calling the CAAP implementation, and loading the result back into Uppaal. The shield is simulated in Query 8, for which any of the allowed actions is selected randomly (this happens implicitly); while safe, this shielded but randomized strategy is not efficient and hits the ball more often than needed, as visualized in Figure 40b.

In Query 9, we learn a strategy `shielded efficient` under the shield using Uppaal Stratego [130]. This strategy keeps the ball in the air without excessive hitting, as shown by the output of Query 10 in Figure 40c. The result of Query 11 shows the expected cost, and Query 12 shows that the safety property holds with high confidence: None of the 10 000 runs were unsafe.

5.2 Further Examples



(a) `efficient` (b) `compact_shield` (c) `shielded_efficient`

Figure 40: *Bouncing ball* simulations (position, velocity) under different strategies.

State-space transformations can be used to synthesize a shield more efficiently [104]. Since UPPAAL supports function calls, transformations can be applied by modifying the model. Details can be found in [123].

Next, we show quantitative results of the shield synthesis and subsequent shield reduction, for which we also use three additional models. Firstly, the *boost converter* [10] models a real circuit for stepping up the voltage of a direct current (DC) input. The controller must keep the voltage close to a reference value, without exceeding safe bounds for the voltage and current. The state space is continuous, with significant random variation in the outcome of actions.

In the *random walk* model [10], [138], the player must travel a certain distance before time runs out by choosing between a fast but expensive and a slow but cheap action. The state space is continuous and the outcomes of actions follow uniform distributions.

In the *water tank* model inspired from [9], a tank must be kept from overflowing or running dry. Water flows from the tank at a rate that varies periodically. At each time step, the player can control the inflow by switching a pump on or off. The state space is discrete.

We show results for computing and reducing shields in Table 10. The *water tank* is fully deterministic, and the *bouncing ball* only has low-variance stochastic behavior. The *boost converter* and *random walk* have a high variance in action outcomes, which is why we use $m = 20$ simulation runs per sampled state. We evaluated the shields statistically and found no unsafe runs in $10\,000$ trials. The reduction yields significantly smaller representations at acceptable run time.

Model	n	m	**Synthesis time**	**Size**	**Reduction time**	**Reduced size**
Bouncing ball	3	1	218s	$1\,430\,000$	53s	2972
Boost converter	3	20	$1\,430s$	$136\,800$	21s	571
Random walk	4	20	82s	$40\,000$	1.5s	60
Water tank	3	1	0.1s	168	0.1s	24

Table 10: Computation time and sizes for synthesizing and reducing shields for three models. The original size is the number of cells, whereas the reduced size is the number of regions. All shields were statistically evaluated to be at least 99.47% safe with a confidence interval of 99% (no unsafe runs observed).

6 Conclusion

We have described our implementation of the shield synthesis algorithm from [10] in the tool UPPAAL COSHY. Our tool can work with rich inputs modeled in Uppaal. We have also presented the CAAP algorithm to reduce the shield representation significantly, which is crucial for deployment on an embedded device.

We see several directions for future integration into UPPAAL. As discussed, our implementation does not apply *systematic* sampling for random dynamics; however, we think that many sources of randomness in UPPAAL models can be handled systematically. Currently, the reduction algorithm CAAP is implemented as a standalone tool, but it would be useful to also integrate it directly with UPPAAL. During development, we found it helpful to visualize shields, as in Figure 37, which could be offered in the user interface. In the same line, an explanation why a state is marked unsafe in a shield would help in debugging a model.

This research was partly supported by the Independent Research Fund Denmark under reference number 10.46540/3120-00041B and the Villum Investigator Grant S4OS under reference number 37819.

Bibliography

[1] E. A. Lee, "Cyber-physical systems-are computing foundations adequate," in *Position paper for NSF workshop on cyber-physical systems: research motivation, techniques and roadmap*, 2006, pp. 1–9.

[2] E. A. Lee, "Cyber physical systems: Design challenges," in *2008 11th IEEE international symposium on object and component-oriented real-time distributed computing (ISORC)*, 2008, pp. 363–369.

[3] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE signal processing magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[5] M. Alloghani, D. Al-Jumeily, J. Mustafina, A. Hussain, and A. J. Aljaaf, "A systematic review on supervised and unsupervised machine learning algorithms for data science," *Supervised and unsupervised learning for data science*, pp. 3–21, 2020.

[6] A. Feinberg, "Markov Decision Processes: Discrete Stochastic Dynamic Programming (Martin L. Puterman)," *SIAM Rev.*, vol. 38, no. 4, p. 689, 1996, doi: 10.1137/1038137.

[7] A. David *et al.*, "On Time with Minimal Expected Cost!," in *ATVA*, F. Cassez and J.-F. Raskin, Eds., in LNCS, vol. 8837. Springer, 2014, pp. 129–145. doi: 10.1007/978-3-319-11936-6_10.

[8] R. Bloem, B. Könighofer, R. Könighofer, and C. Wang, "Shield Synthesis: Runtime Enforcement for Reactive Systems," in *TACAS*, C. Baier and C. Tinelli, Eds., in LNCS, vol. 9035. Springer, 2015, pp. 533–548. doi: 10.1007/978-3-662-46681-0_51.

[9] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe Reinforcement Learning via Shielding," in *AAAI*, S. A. McIlraith and K. Q. Weinberger, Eds., AAAI Press, 2018, pp. 2669–2678. doi: 10.1609/aaai.v32i1.11797.

[10] A. H. Brorholt, P. G. Jensen, K. G. Larsen, F. Lorber, and C. Schilling, "Shielded reinforcement learning for hybrid systems," in *AISoLA*, B. Steffen, Ed., in LNCS, vol. 14380. Springer, 2023, pp. 33–54. doi: 10.1007/978-3-031-46002-9_3.

[11] M. Tappler, S. Pranger, B. Könighofer, E. Muskardin, R. Bloem, and K. G. Larsen, "Automata Learning Meets Shielding," in *Leveraging Applications of*

Formal Methods, Verification and Validation. Verification Principles - 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22-30, 2022, Proceedings, Part I, T. Margaria and B. Steffen, Eds., in Lecture Notes in Computer Science, vol. 13701. Springer, 2022, pp. 335–359. doi: 10.1007/978-3-031-19849-6_20.

[12] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe Reinforcement Learning via Shielding," *CoRR*, 2017, [Online]. Available: http://arxiv.org/abs/1708.08611

[13] B. Könighofer, R. Bloem, N. Jansen, S. Junges, and S. Pranger, "Shields for Safe Reinforcement Learning," *Commun. ACM*, vol. 68, no. 11, pp. 80–90, 2025, doi: 10.1145/3715958.

[14] "Strategies and Considerations for Safe Reinforcement Learning in Programming Cardiac Implantable Electronic Devices," *Medical research archives*, vol. 13, no. 3, 2025, doi: 10.18103/mra.v13i3.6363.

[15] {. H. Taankvist, "Safe, Optimal and Compact Strategies for Cyber Physical Systems." Aalborg University Open Publishing, 2025. doi: 10.54337/aau779529764.

[16] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *CoRR*, 2017, [Online]. Available: http://arxiv.org/abs/1707.06347

[18] S. Huang and S. Ontañón, "A Closer Look at Invalid Action Masking in Policy Gradient Algorithms," *CoRR*, 2020, [Online]. Available: https://arxiv.org/abs/2006.14171

[19] M. Seurin, P. Preux, and O. Pietquin, ""I'm Sorry Dave, I'm Afraid I Can't Do That" Deep Q-Learning from Forbidden Actions," in *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020*, IEEE, 2020, pp. 1–8. doi: 10.1109/IJCNN48605.2020.9207496.

[20] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal control*. John Wiley & Sons, 2012.

[21] J. C. Doyle, B. A. Francis, and A. R. Tannenbaum, *Feedback control theory*. Courier Corporation, 2013.

[22] L. Busoniu, T. de Bruin, D. Tolic, J. Kober, and I. Palunko, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annu. Rev. Control.*, vol. 46, pp. 8–28, 2018, doi: 10.1016/j.arcontrol.2018.09.005.

[23] J. G. Vlachogiannis and N. D. Hatziargyriou, "Reinforcement learning for reactive power control," *IEEE Transactions on Power Systems*, vol. 19, no. 3, pp. 1317–1325, 2004, doi: 10.1109/TPWRS.2004.831259.

[24] M. Noaeen *et al.*, "Reinforcement learning in urban network traffic signal control: A systematic literature review," *Expert Syst. Appl.*, vol. 199, p. 116830, 2022, doi: 10.1016/j.eswa.2022.116830.

[25] J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," *J. Mach. Learn. Res.*, vol. 16, pp. 1437–1480, 2015, doi: 10.5555/2789272.2886795.

[26] R. Bloem, B. Könighofer, R. Könighofer, and C. Wang, "Shield Synthesis: Runtime Enforcement for Reactive Systems," in *TACAS*, C. Baier and C. Tinelli, Eds., in LNCS, vol. 9035. Springer, 2015, pp. 533–548. doi: 10.1007/978-3-662-46681-0_51.

[27] A. David *et al.*, "Statistical Model Checking for Stochastic Hybrid Systems," in *HSB*, E. Bartocci and L. Bortolussi, Eds., in EPTCS, vol. 92. 2012, pp. 122–136. doi: 10.4204/EPTCS.92.9.

[28] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's Decidable about Hybrid Automata?," *J. Comput. Syst. Sci.*, vol. 57, no. 1, pp. 94–124, 1998, doi: 10.1006/jcss.1998.1581.

[29] J. Kapinski, B. H. Krogh, O. Maler, and O. Stursberg, "On Systematic Simulation of Open Continuous Systems," in *HSCC*, O. Maler and A. Pnueli, Eds., in LNCS, vol. 2623. Springer, 2003, pp. 283–297. doi: 10.1007/3-540-36580-X_22.

[30] A. Donzé, "Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems," in *CAV*, T. Touili, B. Cook, and P. B. Jackson, Eds., in LNCS, vol. 6174. Springer, 2010, pp. 167–170. doi: 10.1007/978-3-642-14295-6_17.

[31] A. David, P. G. Jensen, K. G. Larsen, M. Mikucionis, and J. H. Taankvist, "Uppaal Stratego," in *TACAS*, C. Baier and C. Tinelli, Eds., in LNCS, vol. 9035. Springer, 2015, pp. 206–211. doi: 10.1007/978-3-662-46681-0_16.

[32] N. Jansen, B. Könighofer, S. Junges, A. Serban, and R. Bloem, "Safe Reinforcement Learning Using Probabilistic Shields," in *CONCUR*, I. Konnov and L. Kovács, Eds., in LIPIcs, vol. 171. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 3:1–3:16. doi: 10.4230/LIPIcs.CONCUR.2020.3.

[33] O. Bastani and S. Li, "Safe Reinforcement Learning via Statistical Model Predictive Shielding," in *Robotics*, D. A. Shell, M. Toussaint, and M. A. Hsieh, Eds., 2021. doi: 10.15607/RSS.2021.XVII.026.

[34] K. P. Wabersich and M. N. Zeilinger, "A predictive safety filter for learning-based control of constrained nonlinear dynamical systems," *Autom.*, vol. 129, p. 109597, 2021, doi: 10.1016/j.automatica.2021.109597.

[35] S. Carr, N. Jansen, S. Junges, and U. Topcu, "Safe Reinforcement Learning via Shielding under Partial Observability," in *AAAI*, B. Williams, Y. Chen, and J. Neville, Eds., AAAI Press, 2023, pp. 14748–14756. doi: 10.1609/aaai.v37i12.26723.

[36] B. Maderbacher, S. Schupp, E. Bartocci, R. Bloem, D. Nickovic, and B. Könighofer, "Provable Correct and Adaptive Simplex Architecture for Bounded-Liveness Properties," in *SPIN*, G. Caltais and C. Schilling, Eds., in LNCS, vol. 13872. Springer, 2023, pp. 141–160. doi: 10.1007/978-3-031-32157-3_8.

[37] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-End Safe Reinforcement Learning through Barrier Functions for Safety-Critical Continuous Control Tasks," in *AAAI*, AAAI Press, 2019, pp. 3387–3395. doi: 10.1609/aaai.v33i01.33013387.

[38] Y. Luo and T. Ma, "Learning Barrier Certificates: Towards Safe Reinforcement Learning with Zero Training-time Violations," in *NeurIPS*, M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021, pp. 25621–25632. [Online]. Available: https://proceedings.neurips.cc/paper/2021/hash/d71fa38b648d86602d14ac610f2e6194-Abstract.html

[39] M. Hasanbeig, A. Abate, and D. Kroening, "Cautious Reinforcement Learning with Logical Constraints," in *AAMAS*, A. E. F. Seghrouchni, G. Sukthankar, B. An, and N. Yorke-Smith, Eds., International Foundation for Autonomous Agents, Multiagent Systems, 2020, pp. 483–491. doi: 10.5555/3398761.3398821.

[40] F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause, "Safe Model-based Reinforcement Learning with Stability Guarantees," in *NeurIPS*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 908–918. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/766ebcd59621e305170616ba3d3dac32-Abstract.html

[41] Y. Chow, O. Nachum, E. A. Duéñez-Guzmán, and M. Ghavamzadeh, "A Lyapunov-based Approach to Safe Reinforcement Learning," in *NeurIPS*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 8103–8112. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/4fe5149039b52765bde64beb9f674940-Abstract.html

[42] K. G. Larsen, A. Le Coënt, M. Mikucionis, and J. H. Taankvist, "Guaranteed Control Synthesis for Continuous Systems in Uppaal Tiga," in *CyPhy/WESE*, R. D. Chamberlain, W. Taha, and M. Törngren, Eds., in LNCS, vol. 11615. Springer, 2018, pp. 113–133. doi: 10.1007/978-3-030-23703-5_6.

[43] R. Majumdar, N. Ozay, and A.-K. Schmuck, "On abstraction-based controller design with output feedback," in *HSCC*, A. D. Ames, S. A. Seshia, and J. Deshmukh, Eds., ACM, 2020, pp. 15:1–15:11. doi: 10.1145/3365365.3382219.

[44] M. Klischat and M. Althoff, "A Multi-Step Approach to Accelerate the Computation of Reachable Sets for Road Vehicles," in *ITSC*, IEEE, 2020, pp. 1–7. doi: 10.1109/ITSC45102.2020.9294328.

[45] Đ. Žikelić, M. Lechner, T. A. Henzinger, and K. Chatterjee, "Learning Control Policies for Stochastic Systems with Reach-avoid Guarantees," in *AAAI*, AAAI Press, 2023, pp. 11926–11935. doi: 10.1609/aaai.v37i10.26407.

[46] T. S. Badings *et al.*, "Robust Control for Dynamical Systems with Non-Gaussian Noise via Formal Abstractions," *J. Artif. Intell. Res.*, vol. 76, pp. 341–391, 2023, doi: 10.1613/jair.1.14253.

[47] A. Abate, S. Amin, M. Prandini, J. Lygeros, and S. Sastry, "Computational Approaches to Reachability Analysis of Stochastic Hybrid Systems," in *HSCC*, A. Bemporad, A. Bicchi, and G. C. Buttazzo, Eds., in LNCS, vol. 4416. Springer, 2007, pp. 4–17. doi: 10.1007/978-3-540-71493-4_4.

[48] F. Shmarov and P. Zuliani, "ProbReach: A Tool for Guaranteed Reachability Analysis of Stochastic Hybrid Systems," in *SNR*, S. Bogomolov and A. Tiwari, Eds., in EPiC Series in Computing, vol. 37. EasyChair, 2015, pp. 40–48. doi: 10.29007/mh2c.

[49] L. M. Bujorianu, *Stochastic reachability analysis of hybrid systems*. Springer Science & Business Media, 2012.

[50] M. Jaeger, P. G. Jensen, K. G. Larsen, A. Legay, S. Sedwards, and J. H. Taankvist, "Teaching Stratego to Play Ball: Optimal Synthesis for Continuous Space MDPs," in *ATVA*, Y.-F. Chen, C.-H. Cheng, and J. Esparza, Eds., in LNCS, vol. 11781. Springer, 2019, pp. 81–97. doi: 10.1007/978-3-030-31784-3_5.

[51] M. Jaeger, G. Bacci, G. Bacci, K. G. Larsen, and P. G. Jensen, "Approximating Euclidean by Imprecise Markov Decision Processes," in *ISoLA*, T. Margaria and B. Steffen, Eds., in LNCS, vol. 12476. Springer, 2020, pp. 275–289. doi: 10.1007/978-3-030-61362-4_15.

[52] K. G. Larsen, "Statistical Model Checking, Refinement Checking, Optimization, ... for Stochastic Hybrid Systems," in *FORMATS*, M. Jurdzinski and D. Nickovic, Eds., in LNCS, vol. 7595. Springer, 2012, pp. 7–10. doi: 10.1007/978-3-642-33365-1_2.

[53] A. Tarski, "A lattice-theoretical fixpoint theorem and its applications.," *Pacific J. Math.*, vol. 5, no. 2, pp. 285–309, 1955, [Online]. Available: https://www.projecteuclid.org/journalArticle/Download?urlId=pjm%2F1103044538

[54] J. Bernet, D. Janin, and I. Walukiewicz, "Permissive strategies: from parity games to safety games," *RAIRO Theor. Informatics Appl.*, vol. 36, no. 3, pp. 261–275, 2002, doi: 10.1051/ita:2002013.

[55] A. Tarski, *A decision method for elementary algebra and geometry*. The RAND Corporation, 1948. [Online]. Available: https://www.rand.org/pubs/reports/R109.html

[56] J. H. Davenport and J. Heintz, "Real quantifier elimination is doubly exponential," *Journal of Symbolic Computation*, vol. 5, no. 1, pp. 29–35, 1988, doi: 10.1016/S0747-7171(88)80004-X.

[57] M. Laczkovich, “The removal of pi from some undecidable problems involving elementary functions”, *Proceedings of the American Mathematical Society*, vol. 131, no. 7, pp. 2235–2240, 2003, doi: 10.1090/S0002-9939-02-06753-9.

[58] L. Doyen, G. Frehse, G. J. Pappas, and A. Platzer, “Verification of Hybrid Systems,” *Handbook of Model Checking*. Springer, pp. 1047–1110, 2018. doi: 10.1007/978-3-319-10575-8_30.

[59] K. G. Larsen, M. Mikucionis, and J. H. Taankvist, “Safe and Optimal Adaptive Cruise Control,” in *Correct System Design*, R. Meyer, A. Platzer, and H. Wehrheim, Eds., in LNCS, vol. 9360. Springer, 2015, pp. 260–277. doi: 10.1007/978-3-319-23506-6_17.

[60] P. Ashok, J. Kretínský, K. G. Larsen, A. Le Coënt, J. H. Taankvist, and M. Weininger, “SOS: Safe, Optimal and Small Strategies for Hybrid Markov Decision Processes,” in *QEST*, D. Parker and V. Wolf, Eds., in LNCS, vol. 11785. Springer, 2019, pp. 147–164. doi: 10.1007/978-3-030-30281-8_9.

[61] P. Karamanakos, T. Geyer, and S. Manias, “Direct voltage control of DC-DC boost converters using enumeration-based model predictive control,” *IEEE Transactions on Power Electronics*, vol. 29, no. 2, pp. 968–978, 2013.

[62] H. Zhao, N. Zhan, D. Kapur, and K. G. Larsen, “A ``Hybrid'' Approach for Synthesizing Optimal Controllers of Hybrid Systems: A Case Study of the Oil Pump Industrial Example,” in *FM*, D. Giannakopoulou and D. Méry, Eds., in LNCS, vol. 7436. Springer, 2012, pp. 471–485. doi: 10.1007/978-3-642-32759-9_38.

[63] AsgerHB, “Reproducibility Package - Shielded Reinforcement Learning for Hybrid Systems.” 2023.

[64] S. Bogomolov, M. Forets, G. Frehse, K. Potomkin, and C. Schilling, “JuliaReach: a toolbox for set-based reachability,” in *HSCC*, N. Ozay and P. Prabhakar, Eds., ACM, 2019, pp. 39–44. doi: 10.1145/3302504.3311804.

[65] C. L. Guernic and A. Girard, “Reachability Analysis of Hybrid Systems Using Support Functions,” in *CAV*, A. Bouajjani and O. Maler, Eds., in LNCS, vol. 5643. Springer, 2009, pp. 540–554. doi: 10.1007/978-3-642-02658-4_40.

[66] C. J. C. H. Watkins, “Learning from Delayed Rewards,” Doctoral dissertation, 1989.

[67] M. Forets, D. Freire, and C. Schilling, “Efficient reachability analysis of parametric linear hybrid systems with time-triggered transitions,” in *MEMOCODE*, IEEE, 2020, pp. 1–6. doi: 10.1109/MEMOCODE51338.2020.9314994.

[68] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal control*. John Wiley & Sons, 2012.

[69] J. C. Doyle, B. A. Francis, and A. R. Tannenbaum, *Feedback control theory*. Courier Corporation, 2013.

[70] A. Girard, "Controller synthesis for safety and reachability via approximate bisimulation," *Autom.*, vol. 48, no. 5, pp. 947–953, 2012, doi: 10.1016/J.AUTOMATICA.2012.02.037.

[71] P. Tabuada, *Verification and control of hybrid systems - A symbolic approach*. Springer, 2009. [Online]. Available: http://www.springer.com/mathematics/applications/book/978-1-4419-0223-8

[72] A. Weber, M. Rungger, and G. Reissig, "Optimized State Space Grids for Abstractions," *IEEE Trans. Autom. Control.*, vol. 62, no. 11, pp. 5816–5821, 2017, doi: 10.1109/TAC.2016.2642794.

[73] A. Girard, G. Gößler, and S. Mouelhi, "Safety Controller Synthesis for Incrementally Stable Switched Systems Using Multiscale Symbolic Models," *IEEE Trans. Autom. Control.*, vol. 61, no. 6, pp. 1537–1549, 2016, doi: 10.1109/TAC.2015.2478131.

[74] K. Hsu, R. Majumdar, K. Mallik, and A.-K. Schmuck, "Multi-Layered Abstraction-Based Controller Synthesis for Continuous-Time Systems," in *HSCC*, M. Prandini and J. V. Deshmukh, Eds., ACM, 2018, pp. 120–129. doi: 10.1145/3178126.3178143.

[75] N. Jansen, B. Könighofer, S. Junges, A. Serban, and R. Bloem, "Safe Reinforcement Learning Using Probabilistic Shields," in *CONCUR*, I. Konnov and L. Kovács, Eds., in LIPIcs, vol. 171. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 3:1–3:16. doi: 10.4230/LIPIcs.CONCUR.2020.3.

[76] P. Cousot and R. Cousot, "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints," in *POPL*, R. M. Graham, M. A. Harrison, and R. Sethi, Eds., ACM, 1977, pp. 238–252. doi: 10.1145/512950.512973.

[77] W. H. Schilders, H. A. Van der Vorst, and J. Rommes, *Model order reduction: theory, research aspects and applications*, vol. 13. Springer, 2008.

[78] K. G. Larsen and A. Skou, "Bisimulation through Probabilistic Testing," *Inf. Comput.*, vol. 94, no. 1, pp. 1–28, 1991, doi: 10.1016/0890-5401(91)90030-6.

[79] P. Buchholz, "Exact and ordinary lumpability in finite Markov chains," *Journal of applied probability*, vol. 31, no. 1, pp. 59–75, 1994.

[80] G. Bacci, G. Bacci, K. G. Larsen, M. Tribastone, M. Tschaikowski, and A. Vandin, "Efficient Local Computation of Differential Bisimulations via Coupling and Up-to Methods," in *LICS*, IEEE, 2021, pp. 1–14. doi: 10.1109/LICS52264.2021.9470555.

[81] A. Jiménez-Pastor, K. G. Larsen, M. Tribastone, and M. Tschaikowski, "Forward and Backward Constrained Bisimulations for Quantum Circuits," in *TACAS*, B. Finkbeiner and L. Kovács, Eds., in LNCS, vol. 14571. Springer, 2024, pp. 343–362. doi: 10.1007/978-3-031-57249-4_17.

[82] M. Jaeger, P. G. Jensen, K. G. Larsen, A. Legay, S. Sedwards, and J. H. Taankvist, "Teaching Stratego to Play Ball: Optimal Synthesis for Continuous Space MDPs," in *ATVA*, Y.-F. Chen, C.-H. Cheng, and J. Esparza, Eds., in LNCS, vol. 11781. Springer, 2019, pp. 81–97. doi: 10.1007/978-3-030-31784-3_5.

[83] R. V. Florian, "Correct equations for the dynamics of the cart-pole system," technical report, 2005. [Online]. Available: https://coneural.org/florian/papers/05_cart_pole.pdf

[84] N. R. Draper and H. Smith, *Applied regression analysis*. John Wiley & Sons, 1998.

[85] A. H. Høeg-Petersen, K. G. Larsen, A. Wąsowski, and P. G. Jensen, "Minimizing State Space Partitionings Using Decision Trees," in *NWPT*, 2023. [Online]. Available: https://mdu.drive.sunet.se/index.php/s/DtC47mJbPdyBREY

[86] K. Hsu, R. Majumdar, K. Mallik, and A.-K. Schmuck, "Lazy Abstraction-Based Control for Safety Specifications," in *CDC*, IEEE, 2018, pp. 4902–4907. doi: 10.1109/CDC.2018.8619659.

[87] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. in Wiley Series in Probability and Statistics. Wiley, 1994. doi: 10.1002/9780470316887.

[88] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998. [Online]. Available: http://incompleteideas.net/book/the-book-1st.html

[89] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nat.*, vol. 518, no. 7540, pp. 529–533, 2015, doi: 10.1038/NATURE14236.

[90] M. G. Bellemare *et al.*, "Autonomous navigation of stratospheric balloons using reinforcement learning," *Nat.*, vol. 588, no. 7836, pp. 77–82, 2020, doi: 10.1038/S41586-020-2939-8.

[91] S. S. Owicki and D. Gries, "Verifying Properties of Parallel Programs: An Axiomatic Approach," *Commun. ACM*, vol. 19, no. 5, pp. 279–285, 1976, doi: 10.1145/360051.360224.

[92] L. Lamport, "Proving the Correctness of Multiprocess Programs," *IEEE Trans. Software Eng.*, vol. 3, no. 2, pp. 125–143, 1977, doi: 10.1109/TSE.1977.229904.

[93] A. Pnueli, "In Transition From Global to Modular Temporal Reasoning about Programs," in *Logics and Models of Concurrent Systems*, K. R. Apt, Ed., in NATO ASI Series, vol. 13. Springer, 1984, pp. 123–144. doi: 10.1007/978-3-642-82453-1_5.

[94] E. W. Stark, "A Proof Technique for Rely/Guarantee Properties," in *FSTTCS*, S. N. Maheshwari, Ed., in LNCS, vol. 206. Springer, 1985, pp. 369–391. doi: 10.1007/3-540-16042-6_21.

[95] A. Benveniste *et al.*, "Contracts for System Design," *Found. Trends Electron. Des. Autom.*, vol. 12, no. 2–3, pp. 124–400, 2018, doi: 10.1561/1000000053.

[96] D. Giannakopoulou, K. S. Namjoshi, and C. S. Pasareanu, "Compositional Reasoning," *Handbook of Model Checking*. Springer, pp. 345–383, 2018. doi: 10.1007/978-3-319-10575-8_12.

[97] K. G. Larsen, M. Mikucionis, and J. H. Taankvist, "Safe and Optimal Adaptive Cruise Control," in *Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday*, R. Meyer, A. Platzer, and H. Wehrheim, Eds., in LNCS, vol. 9360. Springer, 2015, pp. 260–277. doi: 10.1007/978-3-319-23506-6_17.

[98] K. Zhang, Z. Yang, and T. Basar, "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms," *CoRR*, 2019, [Online]. Available: http://arxiv.org/abs/1911.10635

[99] R. Bloem, K. Chatterjee, and B. Jobstmann, "Graph Games and Reactive Synthesis," *Handbook of Model Checking*. Springer, pp. 921–962, 2018. doi: 10.1007/978-3-319-10575-8_27.

[100] A. David *et al.*, "On Time with Minimal Expected Cost!," in *ATVA*, F. Cassez and J.-F. Raskin, Eds., in LNCS, vol. 8837. Springer, 2014, pp. 129–145. doi: 10.1007/978-3-319-11936-6_10.

[101] N. Jansen, B. Könighofer, S. Junges, A. Serban, and R. Bloem, "Safe Reinforcement Learning Using Probabilistic Shields," in *CONCUR*, I. Konnov and L. Kovács, Eds., in LIPIcs, vol. 171. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 3:1–3:16. doi: 10.4230/LIPICS.CONCUR.2020.3.

[102] W.-C. Yang, G. Marra, G. Rens, and L. D. Raedt, "Safe Reinforcement Learning via Probabilistic Logic Shields," in *IJCAI*, ijcai.org, 2023, pp. 5739–5749. doi: 10.24963/IJCAI.2023/637.

[103] S. Carr, N. Jansen, S. Junges, and U. Topcu, "Safe Reinforcement Learning via Shielding under Partial Observability," in *AAAI*, B. Williams, Y. Chen, and J. Neville, Eds., AAAI Press, 2023, pp. 14748–14756. doi: 10.1609/AAAI.V37I12.26723.

[104] A. H. Brorholt, A. H. Høeg-Petersen, K. G. Larsen, and C. Schilling, "Efficient Shield Synthesis via State-Space Transformation," in *AISoLA*, B. Steffen, Ed., in LNCS, vol. 15217. Springer, 2024, pp. 206–224. doi: 10.1007/978-3-031-75434-0_14.

[105] B. Könighofer, R. Bloem, R. Ehlers, and C. Pek, "Correct-by-Construction Runtime Enforcement in AI - A Survey," in *Principles of Systems Design - Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, J.-F. Raskin, K. Chatterjee, L. Doyen, and R. Majumdar, Eds., in LNCS, vol. 13660. Springer, 2022, pp. 650–663. doi: 10.1007/978-3-031-22337-2_31.

[106] H. Krasowski, J. Thumm, M. Müller, L. Schäfer, X. Wang, and M. Althoff, "Provably Safe Reinforcement Learning: Conceptual Analysis, Survey, and Benchmarking," *Trans. Mach. Learn. Res.*, vol. 2023, 2023, [Online]. Available: https://openreview.net/forum?id=mcN0ezbnzO

[107] S. Bharadwaj, R. Bloem, R. Dimitrova, B. Könighofer, and U. Topcu, "Synthesis of Minimum-Cost Shields for Multi-agent Systems," in *ACC*, IEEE, 2019, pp. 1048–1055. doi: 10.23919/ACC.2019.8815233.

[108] W. Zhang and O. Bastani, "MAMPS: Safe Multi-Agent Reinforcement Learning via Model Predictive Shielding," *CoRR*, 2019, [Online]. Available: http://arxiv.org/abs/1910.12639

[109] D. Raju, S. Bharadwaj, F. Djeumou, and U. Topcu, "Online Synthesis for Runtime Enforcement of Safety in Multiagent Systems," *IEEE Trans. Control. Netw. Syst.*, vol. 8, no. 2, pp. 621–632, 2021, doi: 10.1109/TCNS.2021.3061900.

[110] Z. Qin, K. Zhang, Y. Chen, J. Chen, and C. Fan, "Learning Safe Multi-agent Control with Decentralized Neural Barrier Certificates," in *ICLR*, OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=P6_q1BRxY8Q

[111] I. Elsayed-Aly, S. Bharadwaj, C. Amato, R. Ehlers, U. Topcu, and L. Feng, "Safe Multi-Agent Reinforcement Learning via Shielding," in *AAMAS*, F. Dignum, A. Lomuscio, U. Endriss, and A. Nowé, Eds., ACM, 2021, pp. 483–491. doi: 10.5555/3463952.3464013.

[112] W. Xiao, Y. Lyu, and J. M. Dolan, "Model-based Dynamic Shielding for Safe and Efficient Multi-agent Reinforcement Learning," in *AAMAS*, N. Agmon, B. An, A. Ricci, and W. Yeoh, Eds., ACM, 2023, pp. 1587–1596. doi: 10.5555/3545946.3598814.

[113] A. Abate *et al.*, "Rational verification: game-theoretic verification of multi-agent systems," *Appl. Intell.*, vol. 51, no. 9, pp. 6569–6584, 2021, doi: 10.1007/S10489-021-02658-Y.

[114] Y. Li, H. Zhu, K. Braught, K. Shen, and S. Mitra, "Verse: A Python Library for Reasoning About Multi-agent Hybrid System Scenarios," in *CAV*, C. Enea and A. Lal, Eds., in LNCS, vol. 13964. Springer, 2023, pp. 351–364. doi: 10.1007/978-3-031-37706-8_18.

[115] A. Partovi and H. Lin, "Assume-guarantee cooperative satisfaction of multi-agent systems," in *ACC*, IEEE, 2014, pp. 2053–2058. doi: 10.1109/ACC.2014.6859441.

[116] L. Mikulski, W. Jamroga, and D. Kurpiewski, "Assume-Guarantee Verification of Strategic Ability," in *PRIMA*, R. Aydogan, N. Criado, J. Lang, V. Sánchez-Anguix, and M. Serramia, Eds., in LNCS, vol. 13753. Springer, 2022, pp. 173–191. doi: 10.1007/978-3-031-21203-1_11.

[117] K. Chatterjee, M. Chmelik, and M. Tracol, "What is decidable about partially observable Markov decision processes with \(\omega \) -regular objectives," *J. Comput. Syst. Sci.*, vol. 82, no. 5, pp. 878–911, 2016, doi: 10.1016/J.JCSS.2016.02.009.

[118] S. V. Albrecht, F. Christianos, and L. Schäfer, *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024. [Online]. Available: https://www.marl-book.com/

[119] C. Yu *et al.*, "The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games," in *NeurIPS*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., 2022. [Online]. Available: http://papers.nips.cc/paper/_files/paper/2022/hash/9c1535a02f0ce079433344e14d910597-Abstract-Datasets/_and/_Benchmarks.html

[120] A. H. Brorholt, K. G. Larsen, and C. Schilling, "Compositional Shielding and Reinforcement Learning for Multi-Agent Systems," 2024. doi: 10.48550/ARXIV.2410.10460.

[121] M. Bettini, A. Prorok, and V. Moens, "BenchMARL: Benchmarking Multi-Agent Reinforcement Learning," *CoRR*, 2023, doi: 10.48550/ARXIV.2312.01472.

[122] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *CoRR*, 2017, [Online]. Available: http://arxiv.org/abs/1707.06347

[123] A. H. Brorholt *et al.*, "Uppaal Coshy: Automatic Synthesis of Compact Shields for Hybrid Systems." [Online]. Available: https://arxiv.org/abs/2508.16345

[124] K. Chatterjee, T. A. Henzinger, B. Jobstmann, and A. Radhakrishna, "Gist: A Solver for Probabilistic Games," in *CAV*, T. Touili, B. Cook, and P. B. Jackson, Eds., in LNCS, vol. 6174. Springer, 2010, pp. 665–669. doi: 10.1007/978-3-642-14295-6_57.

[125] K. Chatterjee, T. A. Henzinger, B. Jobstmann, and R. Singh, "QUASY: Quantitative Synthesis Tool," in *TACAS*, P. A. Abdulla and K. R. M. Leino, Eds., in LNCS, vol. 6605. Springer, 2011, pp. 267–271. doi: 10.1007/978-3-642-19835-9_24.

[126] M. Kwiatkowska, G. Norman, D. Parker, and G. Santos, "PRISM-games 3.0: Stochastic Game Verification with Concurrency, Equilibria and Time," in *CAV*, S. K. Lahiri and C. Wang, Eds., in LNCS, vol. 12225. Springer, 2020, pp. 475–487. doi: 10.1007/978-3-030-53291-8_25.

[127] S. Pranger, B. Könighofer, L. Posch, and R. Bloem, "TEMPEST - Synthesis Tool for Reactive Systems and Shields in Probabilistic Environments," in *ATVA*, Z. Hou and V. Ganesh, Eds., in LNCS, vol. 12971. Springer, 2021, pp. 222–228. doi: 10.1007/978-3-030-88885-5_15.

[128] M. Z. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of Probabilistic Real-Time Systems," in *CAV*, G. Gopalakrishnan and S. Qadeer, Eds., in LNCS, vol. 6806. Springer, 2011, pp. 585–591. doi: 10.1007/978-3-642-22110-1_47.

[129] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime, "UPPAAL-Tiga: Time for Playing Games!," in *CAV*, W. Damm and H. Hermanns, Eds., in LNCS, vol. 4590. Springer, 2007, pp. 121–125. doi: 10.1007/978-3-540-73368-3_14.

[130] A. David, P. G. Jensen, K. G. Larsen, M. Mikucionis, and J. H. Taankvist, "Uppaal Stratego," in *TACAS*, C. Baier and C. Tinelli, Eds., in LNCS, vol. 9035. Springer, 2015, pp. 206–211. doi: 10.1007/978-3-662-46681-0_16.

[131] M. Du, N. Liu, and X. Hu, "Techniques for interpretable machine learning," *Commun. ACM*, vol. 63, no. 1, pp. 68–77, 2020, doi: 10.1145/3359786.

[132] J. R. Quinlan, "Learning Decision Tree Classifiers," *ACM Comput. Surv.*, vol. 28, no. 1, pp. 71–72, 1996, doi: 10.1145/234313.234346.

[133] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth, 1984.

[134] P. Ashok, M. Jackermeier, P. Jagtap, J. Kretínský, M. Weininger, and M. Zamani, "dtControl: decision tree learning algorithms for controller representation," in *HSCC*, A. D. Ames, S. A. Seshia, and J. Deshmukh, Eds., ACM, 2020, pp. 17:1–17:7. doi: 10.1145/3365365.3382220.

[135] P. Ashok, M. Jackermeier, J. Kretínský, C. Weinhuber, M. Weininger, and M. Yadav, "dtControl 2.0: Explainable Strategy Representation via Decision Tree Learning Steered by Experts," in *TACAS*, J. F. Groote and K. G. Larsen, Eds., in LNCS, vol. 12652. Springer, 2021, pp. 326–345. doi: 10.1007/978-3-030-72013-1_17.

[136] E. Demirovic *et al.*, "MurTree: Optimal Decision Trees via Dynamic Programming and Search," *J. Mach. Learn. Res.*, vol. 23, pp. 26:1–26:47, 2022, [Online]. Available: https://jmlr.org/papers/v23/20-520.html

[137] E. Demirović, C. Schilling, and A. Lukina, "In Search of Trees: Decision-Tree Policy Synthesis for Black-Box Systems via Search," in *AAAI*, AAAI Press, 2025, pp. 27250–27257. doi: https://doi.org/10.1609/aaai.v39i26.34934.

[138] M. Jaeger, G. Bacci, G. Bacci, K. G. Larsen, and P. G. Jensen, "Approximating Euclidean by Imprecise Markov Decision Processes," in *ISoLA*, T. Margaria and B. Steffen, Eds., in LNCS, vol. 12476. Springer, 2020, pp. 275–289. doi: 10.1007/978-3-030-61362-4_15.