Safe Reinforcement Learning with Shields

Natasha Alechina[1,2] and Brian Logan[2,3]

[1]Open University NL, [2]Utrecht University, [3]University of Aberdeen

natasha.alechina@ou.nl, b.s.logan@uu.nl

Outline of this lecture

1. Safe reinforcement learning
2. Shields
3. Synthesising shields
4. Case studies

Safe reinforcement learning

Safe reinforcement learning

- RL is increasingly being used in safety-critical applications
- this means we cannot just do “trial and error” learning and hope for the best
- we need to be able to show that the agent’s behaviour is safe
- however, “safety” can be defined in a number of ways
- Krasowski et al. (2022) identify three main approaches to safe RL:
  - *constraint optimisation* approaches
  - *probabilistic model-based* approaches
  - **provably safe** approaches

Provably safe RL

A number of approaches to provably safe RL have been proposed in the literature:

- *action replacement*, e.g., (Hunt et al. 2021), where unsafe actions are replaced by (default or sampled) safe actions
- *action projection*, e.g., (Cheng et al. 2019), where unsafe actions are replaced by projecting them to the safe action space
- *action masking*, e.g., (Krasowski et al. 2020), where the agent is able to choose actions only from the safe action space

Non-Markovian safety constraints

- most provably safe approaches to RL are *Markovian*, i.e., the set of actions which are considered “safe” is based only on the current state
- we will focus on non-Markovian approaches where the safety constraints are expressed in temporal logic

Shields

Shields for Safe RL

- shields (Alshiekh et al. 2018; ElSayed-Aly et al. 2021) are an action masking approach to **provably** safe RL
- safety constraints are specified in safety LTL, allowing the specification of non-Markovian contraints
- Alshiekh et al. define safe RL as: "the process of learning an optimal policy while satisfying a temporal logic safety specification φ_s during the learning and execution phases"

What is a shield?

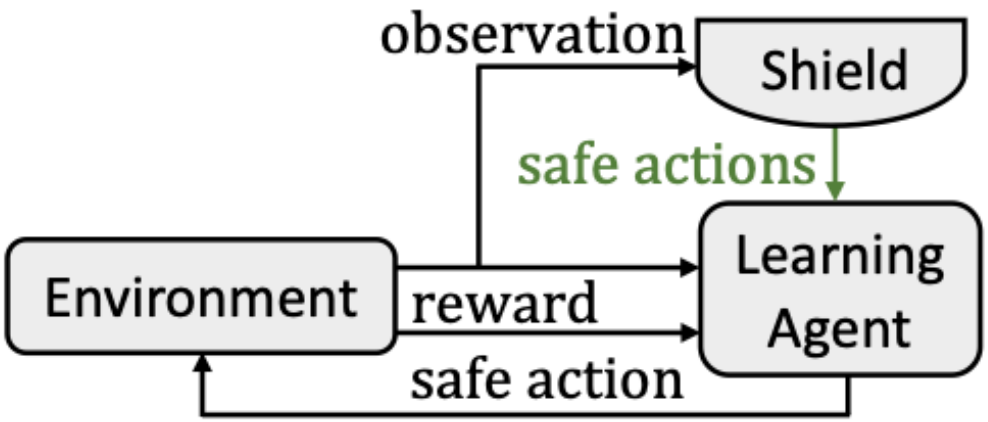
- the key idea behind shields is to ensure the agent does not take actions that can lead to safety violations now or in the future
- the shield “looks ahead” an arbitrary number of steps to check whether an action executed now must lead to a safety violation in the future
- shields are synthesised from:
  - a safety specification expressed as a safety LTL formula and
  - an abstraction of the MDP in which the agent learns

Shields and RL

- the shield is placed in the classic reinforcement learning loop
- shields operate either **before** the agent decides which action to take (*preemptive shield*), or **after** (*post-posed shield*)
- in this lecture, we only consider preemptive shields as they are simpler to explain
- however post-posed shields can be easier to integrate with some Deep RL algorithms

Preemptive Shields

A preemptive shield outputs a set of safe actions at each timestep:



How shields work

- like a reward machine, a shield is a Mealy machine but instead of rewards, it outputs sets of safe actions
- the output function κ takes the state of the shield and current observation as inputs and outputs a set of actions for the agent to choose from
- the shield should be minimally restrictive, i.e., an action is only excluded if it must lead to an unsafe state
- example: a car heading towards a cliff; actions that should be excluded by the shield involve continuing in the same direction; stopping or going in a different direction should be allowed

Formal definition of shields

Definition (Shield (Alshiekh et al. 2018))

A shield $\mathcal{S} = (Q, q_0, \Sigma_I, \Sigma_O, \delta, \kappa)$ is a Mealy machine where:

- Q is a finite set of states
- q_0 is the initial state
- Σ_I is the input alphabet (observations)
- Σ_O is the output alphabet (sets of actions)
- $\delta : Q \times \Sigma_I \to Q$ is the state transition function
- $\kappa : Q \times \Sigma_I \to \Sigma_O$ is the output function

Simplifying the definition

Σ_I consists of pairs (agent action, environment response), so is a product $A \times L$ of two alphabets, A of actions and L of labels

Definition (Shield in this lecture)

A shield $\mathcal{S} = (Q, q_0, A \times L, 2^A, \delta, \kappa)$ is a Mealy machine where:

- Q is a finite set of states
- q_0 is the initial state
- $\delta : Q \times A \times L \to Q$ is the state transition function
- $\kappa : Q \times A \times L \to 2^A$ is the output function

Synthesising shields

Reactive systems

- two interacting entities: agent who produces actions from A and environment that produces labels from L
- their (infinite) interactions produce runs of the system: paths of pairs (action, label)
- not all paths are possible
- agent's actions allow only a certain number of environment's responses (e.g. postconditions of actions)
- environment's responses limit actions executable by agents and/or their effects (e.g. preconditions of actions)

Reactive synthesis

- MDP abstraction is an automaton that accepts all actually possible runs of the system
- safety automaton accepts safe runs of the system
- goal: synthesise a strategy for the agent to stay in the safe states
- the strategy should return the set of all safe actions, at each step
- this is achieved by solving a safety game on the product of the MDP abstraction and the safety automaton

Safety LTL

- Safety LTL is a fragment of the full LTL language.
- syntactically, it contains only formulas in negation normal form: that is, negations are only on propositional variables
- only operators X and G are allowed (no U or F)
- formally: fix a finite set of propositional symbols $\mathcal{PV}$. A Safety LTL formula φ is:

Definition (Syntactic definition of Safety LTL)

$$\varphi := \top \mid p \in \mathcal{PV} \mid \neg p \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid X\varphi \mid G\varphi$$

Semantic definition of safety LTL

- safety LTL formulas intuitively require traces to have only “good” prefixes
- as soon as a prefix of a trace is “bad”, any trace extending such prefix will not satisfy the formula

Definition (Semantic definition of Safety LTL)

An LTL formula φ is a **safety** LTL formula if and only if, for every trace λ such that $\lambda \not\models \varphi$, there exists an index i such that, for any other trace λ', it follows that $\lambda[0 \dots i]\lambda' \not\models \varphi$.

Equivalence of the two definitions

The two definitions are equivalent (not a proof, but an intuition):

- observe that if a trace does not satisfy $X\varphi$, the required i is $= 1$; does not matter how the trace continues after that, $X\varphi$ has been falsified
- if a trace does not satisfy $G\varphi$, there is some index i where φ is false; after this i, again it does not matter how the trace continues
- on the other hand, if an infinite trace λ does not satisfy $F\varphi$, there is no finite 'bad prefix' of λ that is a witness to this fact

Safety automaton

Definition

A *safety automaton* is a deterministic finite state automaton $(Q, q_0, \Sigma, \delta, F)$ where $F \subseteq Q$ is a set of *safe states* that accepts a path if and only if only visits safe states while reading the path.

Safety LTL Formulas and Automata

It has been shown that for each safety LTL formula there is a deterministic finite safety automaton recognising the same language

Fact ((Kupferman et al. 2001))

For every safety LTL formula φ, there exists a deterministic safety automaton $\mathcal{D}^{\varphi} = (Q, q_0, \Sigma, \delta, F)$ such that, for every input sequence $\lambda \in \Sigma^{\omega}$, $\mathcal{D}^{\varphi}$ visits only safe states in F when reading λ if and only if $\lambda \models \varphi$.

Given a safety LTL formula φ, it is possible to build a safety automaton of size double exponential in that of φ. The time complexity of the construction is, again, double exponential in the size of φ (Kupferman et al. 2001)

Example: WaterTank

- agent has to learn how to optimise energy consumption for a hot water storage tank with capacity 100 liters
- this involves adjusting water level in the tank by switching on and off its intake valve
- when the intake valve is open, the water level in the tank increases by between 1 and 2 litres per second
- when the intake valve is closed, the water level decreases by between 0 and 1 litres per second
- rapidly opening and closing the valve damages the valve

Example from (Alshiekh et al. 2018)

Example: WaterTank safety properties

1. the agent has to keep the tank from overflowing or becoming empty
2. to prevent the valve from wearing out, every time it is opened or closed, the valve has to remain open (resp. closed) for at least 3 seconds

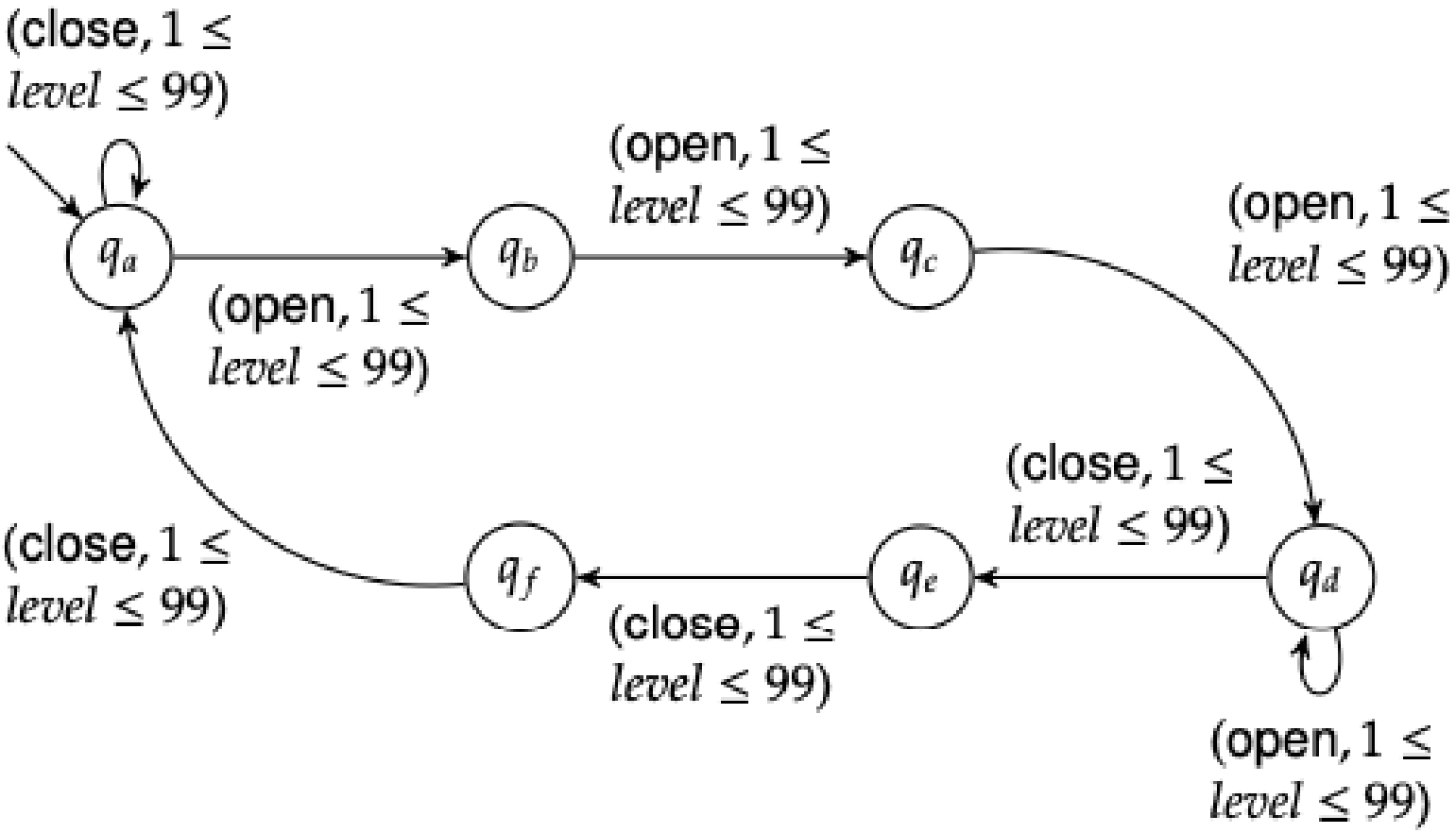
- note that rewards (for optimising energy consumption) and safety properties may conflict; e.g., it may be more energy efficient to close the valve as soon as possible

Example WaterTank safety LTL properties

The two safety properties can be expressed using the following safety LTL formua:

$$G(level > 0) \wedge G(level < 100) \wedge$$
$$G((open \wedge X close) \rightarrow XX close \wedge XXX close) \wedge$$
$$G((close \wedge X open) \rightarrow XX open \wedge XXX open)$$

Example: WaterTank safety automaton



All states are accepting, transitions to the error state s_{fail} not shown.
From (Alshiekh et al. 2018)

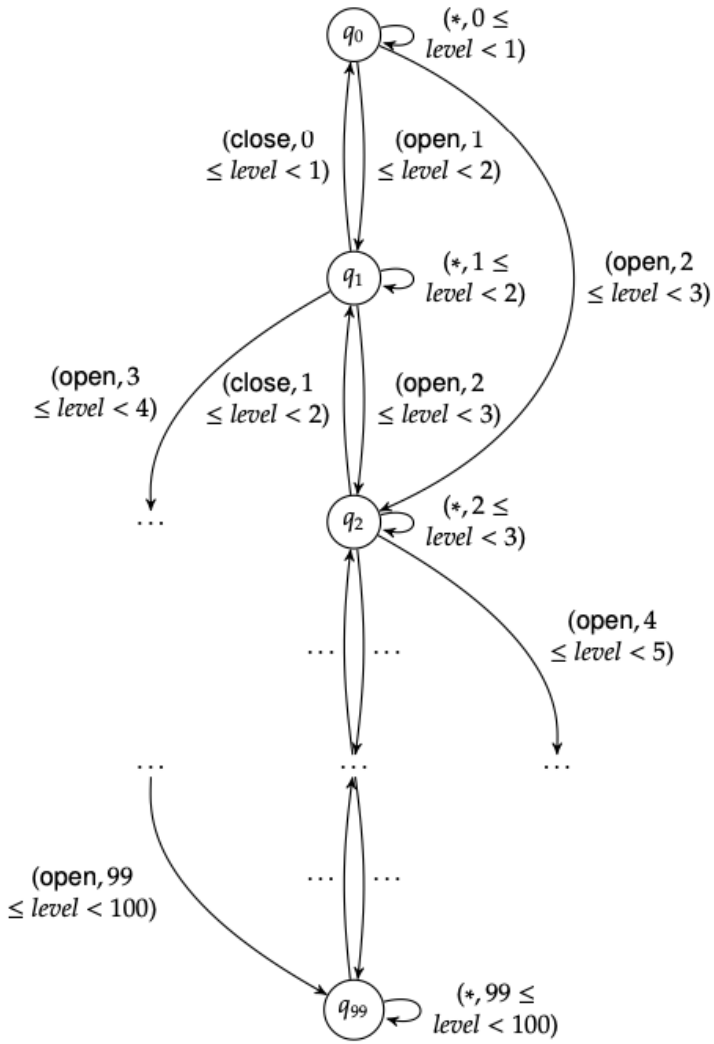
MDP abstractions

Fix an MDP $M = (S, A, p, r, \gamma)$ and an MDP "*observer function*" $f : S \to L$, for some set L.

A deterministic finite state automaton (DFA) $\mathcal{D}^M = (Q, q_0, \Sigma, \delta, F)$ is an "*abstraction*" of M with respect to L if:

1. $\Sigma = A \times L$;
2. For every trace $s_0 s_1 \ldots \in S^\omega$ and the corresponding action sequence $a_0 a_1 \ldots \in A^\omega$ of the MDP, we have that the automaton run $q_0 q_1 \ldots \in Q^\omega$, where $q_{i+1} = \delta(q_i, (a_i, f(s_i)))$ for all $i \in \mathbb{N}$, always remains in F.

Example: WaterTank MDP abstraction (fragment)



From (Alshiekh et al. 2018)

2-player Safety Games

Definition

A 2-player safety game $\mathcal{G} = (G, g_0, A, L, \delta, F)$ is a tuple where:

- G is the finite set of states;
- g_0 is the initial state of the game;
- $\delta : G \times A \times L \rightharpoonup G$ is the deterministic transition function;
- $F \subseteq G$ is the set of safe states.

At each timestep the game is in some state g, the agent takes an action in $a \in A$, the environment picks a label $\ell \in L$, and then the game moves to state $\delta(g, a, \ell)$.

A play $\pi = \pi_0 \pi_1 \ldots \in Q^\omega$ is an infinite sequence of states such that, for each $i \in \mathbb{N}$, there are $a \in A, \ell \in L$ such that $\pi_{i+1} = \delta(\pi_i, a, l)$.
The goal of the agent is to **only** visit safe states in F: a play π is won by the agent if and only if $\pi_i \in F$ for all $i \in \mathbb{N}$.

Strategies and Winning Regions

A strategy for the agent is a function $\rho : G \to A$, mapping states to actions.

A play π from a state $q \in Q$ is "*compatible*" with a strategy ρ if:

1. $\pi_0 = q$;
2. For all $i \in \mathbb{N}$, there is some $\ell \in L$ such that $\pi_{i+1} = \delta(\pi_i, \rho(\pi_i), \ell)$.

A strategy ρ is "*winning at q*" if and only if the agent wins every play from q compatible with ρ: regardless of the environment's choices, the game will always be in a safe state if the agent follows ρ from q.

Definition

The winning region $W \subseteq Q$ is the set of states for which the agent has a strategy ρ that is winning at any state $q \in W$.

Solving safety games

- start with $Win_0 = F$
- at each iteration remove states from Win_i if in those states there is no safe action: an action that for all environment responses keeps the agent in the winning region
- the construction is guaranteed to terminate (possibly with $Win_n = \emptyset$)
- at most linear in $\mathcal{G}$
- once we have Win_n (in all states in Win_n the agent has at least one action that guarantees staying in Win_n), make another pass over Win_n to collect all such safe actions for each state

Simple example

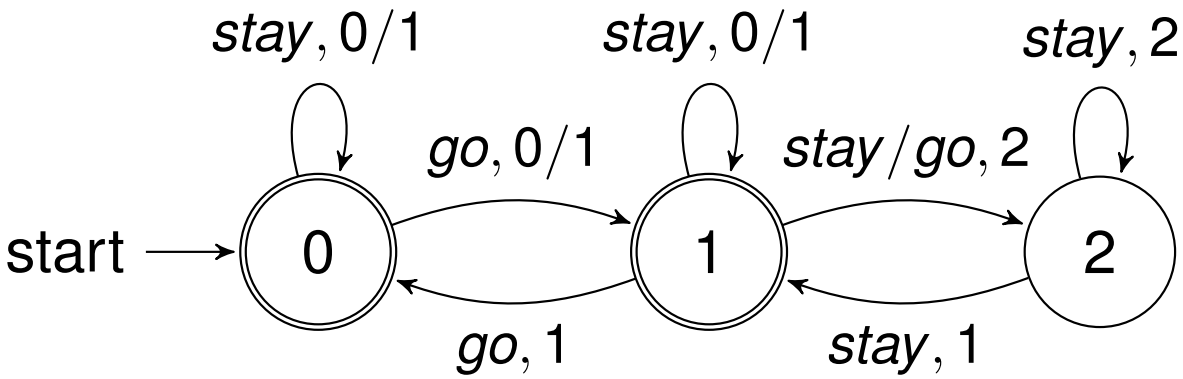


Figure 1: Safe states are $\{0, 1\}$, agent's actions *stay*, *go*, environment's labels $0, 1, 2$.

- $Win_0 = \{0, 1\}$
- $Win_1 = \{0\}$ (in 1, whatever action the agent choses, *stay* or *go*, environment may play 2; in 0, it can chose *stay* to remain in 0
- $\{0\}$ is the winning region and the only safe action in 0 is *stay*.

Synthesising shields

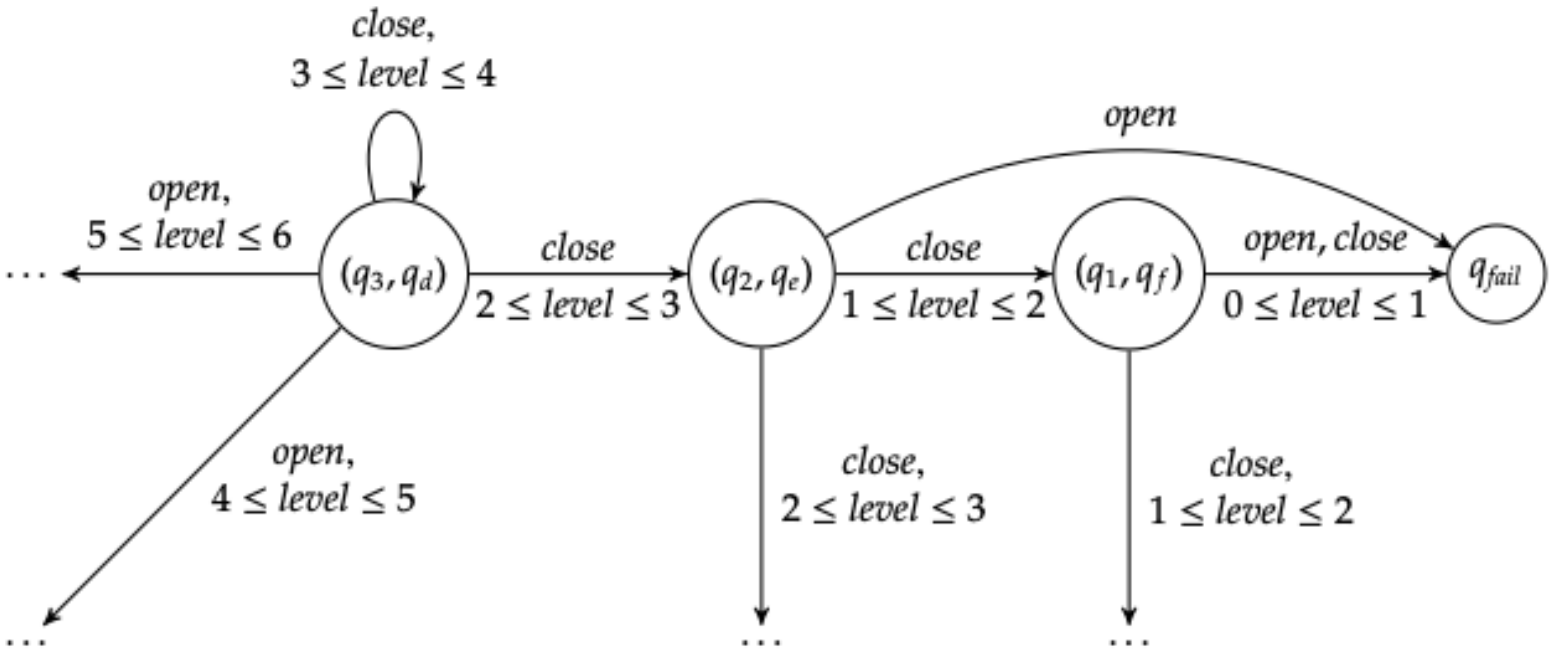
Given an MDP abstraction $\mathcal{D}^M = (Q^M, q_0^M, A \times L, \delta^M, F^M)$, for an MDP $M = (S, A, p, r, \gamma)$ and a set of observations L, and a safety LTL automaton $\mathcal{D}^\varphi = (Q^\varphi, q_0^\varphi, A \times L, \delta^\varphi, F^\varphi)$.

1. Build a safety game $\mathcal{G} = (G, g_0, A, L, \delta, F)$, obtained as the product between $\mathcal{D}^M$ and $\mathcal{D}^\varphi$:
   - $G = Q^M \times Q^\varphi$
   - $g_0 = (q_0^M, q_0^\varphi)$
   - $\delta : G \times A \times L \to G$, defined as $\delta((q^M, q^\varphi), a, \ell) = (\delta^M(q^M, (a, \ell)), \delta^\varphi(q^\varphi, (a, \ell)))$
   - $F = (Q^M \times F^\varphi) \cup ((Q^M \setminus F^M) \times Q^\varphi)$
   - Observation: note that safe states include $(Q^M \setminus F^M) \times Q^\varphi$ which are non-accepting states of the MDP abstraction, that is, when we receive inputs incompatible with our model of the system. Debatable choice.

Synthesising shields (cont.)

2. Compute the winning region $W \subseteq Q$ and the set of safe actions for each state in W (maximally permissive strategy for staying in W). This strategy is represented by the Mealy machine (shield) below.
3. Define the shield $\mathcal{S} = (Q_{\mathcal{S}}, q_{0,\mathcal{S}}, \Sigma_{I,\mathcal{S}}, \Sigma_{O,\mathcal{S}}, \delta_{\mathcal{S}}, \kappa_{\mathcal{S}})$ as follows:
   - $Q_{\mathcal{S}} = G$
   - $q_{0,\mathcal{S}} = g_0$
   - $\Sigma_{I,\mathcal{S}} = A \times L$
   - $\Sigma_{O,\mathcal{S}} = 2^A$
   - $\delta_{\mathcal{S}}(g, a, \ell) = \delta(g, a, \ell)$
   - $\kappa_{\mathcal{S}}(g, \ell) = \{a \in A \mid \delta(g, a, \ell) \in W\}$

Example: WaterTank safety game



From Alshiekh et al. (2018)

Complexity of Shields

- As we have seen building a safety DFA $\mathcal{D}^{\varphi}$ that recognizes the safety language of a safety LTL formula φ takes time double exponential in the size of φ
- Moreover, the size of the automaton $\mathcal{D}^{\varphi}$ is also double exponential in that of φ
- This implies that the size of the safety game $\mathcal{G}$ is double exponential in the size of φ, times the size of the MDP abstraction automaton $\mathcal{D}^{M}$
- Solving safety games takes linear, in the size of the game, time
- this implies the following:

The time complexity to generate a shield and the size of the shield itself is double exponential in the size of the input safety LTL formula φ

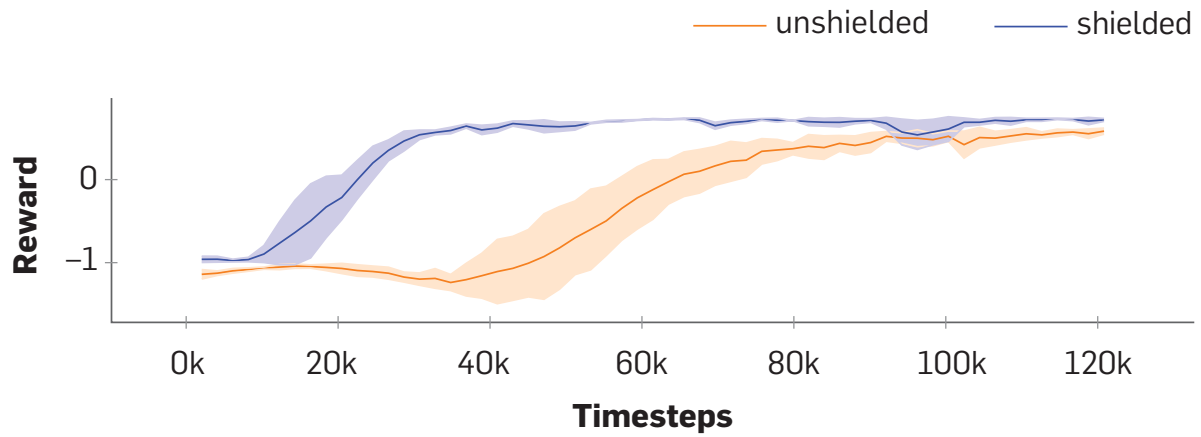
Case studies

Case study: UAV package delivery

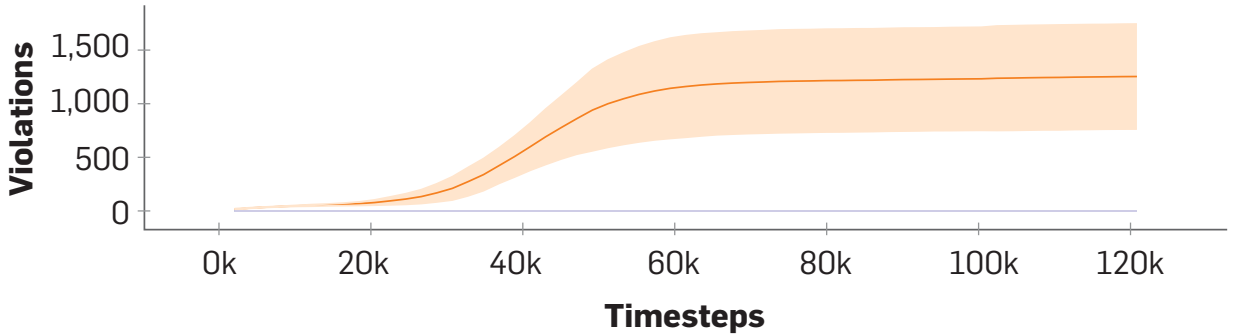
- UAV agent must learn to deliver packages to a specified location
- safety constraint is to avoid collisions with buildings and another UAVs
- the other UAVs are assumed to be *adversarial*, i.e., they actively try to collide with the agent
- agent was trained using maskable PPO with and without a shield to find an optimal and safe policy to deliver the package
- the agent is rewarded for reaching the goal and receives a negative reward for violating safety; it also receives a small negative reward for each step taken

(Könighofer et al. 2025)

Case study: UAV package delivery violations



(a) Reward averaged over five runs



(b) Accumulated safety violations averaged over five runs

From Könighofer et al. (2025)

Beyond absolute safety guarantees

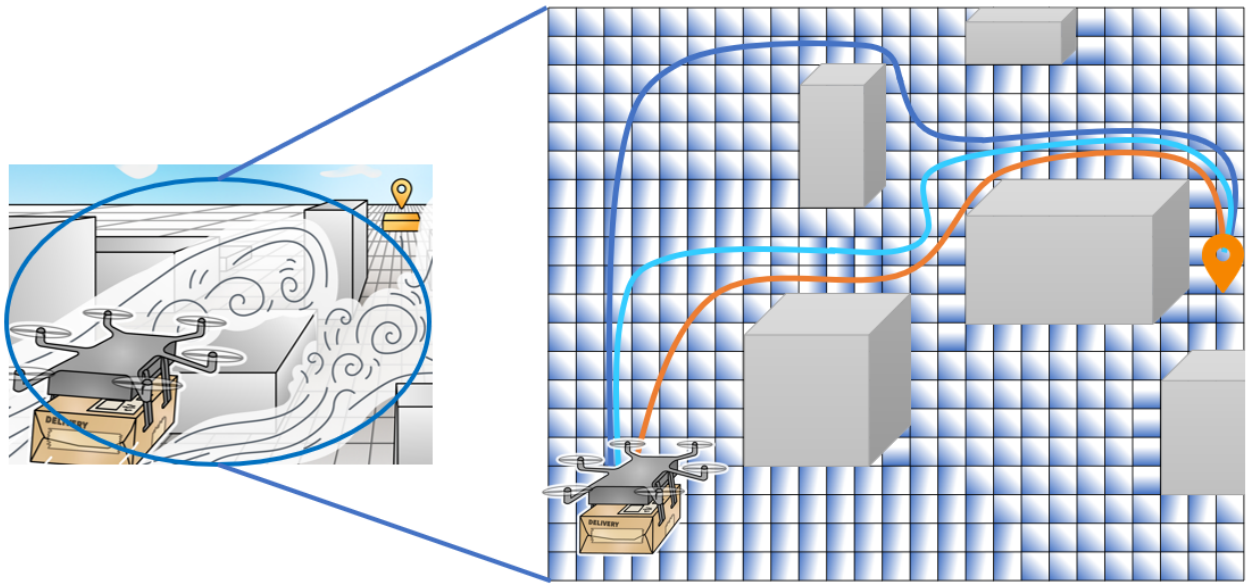
- in many real-world systems it is impossible to avoid risks completely
- however it is possible to give probabilistic guarantees
- this inspired introduction of probabilistic shields
- for probabilistic shields it makes sense to consider finite episodes (in an infinite run, a very unlikely event is bound to eventually happen, so we would be back to absolute guarantees)

Case study: UAV with probabilistic shield

- UAV agent must learn to deliver packages to a specified location
- safety constraint is to avoid collisions with buildings
- wind may affect the movement of the agent
- wind direction is modelled individually for each cell, with a 5% probability that the wind displaces the UAV in the direction of the wind
- evaluation used probabilistic shields with a a finite horizon of 20 time steps and probability thresholds $\epsilon \in \{0.00, 0.03, 0.05\}$, where ϵ limits the risk the agent is allowed to take
- training and reward functions were the same as in the previous experiment

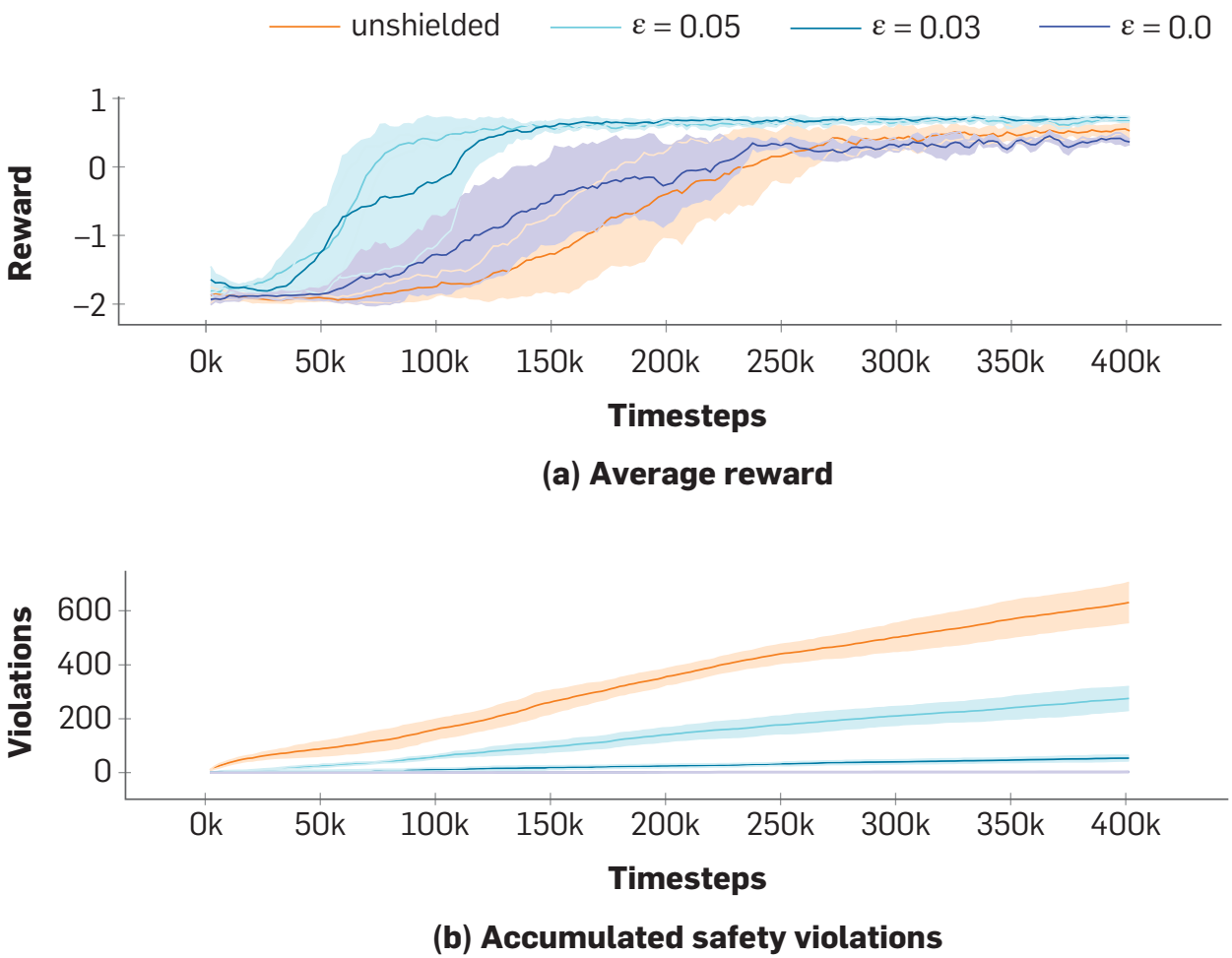
(Könighofer et al. 2025)

Case study: probabilistic shield environment



From Könighofer et al. (2025)

Case study: probabilistic shield violations



From Könighofer et al. (2025)

Tools for incorporating shields

- Pranger and Könighofer (2026) describe an extension to the shield synthesis tool TEMPEST
- `tempestpy` is a Python library which integrates TEMPEST-based shield synthesis directly into the Gymnasium API
- allows shields to be synthesized and deployed within existing RL pipelines
- also extends TEMPEST's algorithmic support to compute sound shields for stochastic multiplayer games while preserving formal safety guarantees
- package also includes symbolic models for MiniGrid and MiniGridSafe, a collection of playground environments which aim to make shielding easily accessible and experimentally transparent

References

- Hunt, N.; Fulton, N.; Magliacane, S.; Hoang, T. N.; Das, S.; and Solar-Lezama, A. 2021. Verifiably Safe Exploration for End-to-End Reinforcement Learning. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, HSCC '21. ACM.

- Krasowski, H.; Thumm, J.; Müller, M.; Schäfer, L.; Wang, X.; and Althoff, M. 2023. Provably Safe Reinforcement Learning: A Theoretical and Experimental Comparison. arXiv:2205.06750.

- Krasowski, H.; Wang, X.; and Althoff, M. 2020. Safe reinforcement learning for autonomous lane changing using set-based prediction. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 1–7. IEEE.

- Könighofer, B.; Bloem, R.; Jansen, N.; Junges, S.; Pranger, S.: Shields for Safe Reinforcement Learning. Commun. ACM 68(11): 80-90 (2025).

- Kupferman, O.; and Vardi, M. Y. 2001. Model Checking of Safety Properties. *Formal Methods in System Design*, 19(3): 291–314.

- Pranger, S.; Könighofer, B.: Easy-to-Use Shielding for Reinforcement Learning. CoRR abs/2606.03804 (2026) https://arxiv.org/abs/2606.03804