Safe Reinforcement Learning via Shielding

Mohammed Alshiekh Roderick Bloem Rudiger Ehlers

Presented by : Shobhit SIngh

June 4, 2025

Contents

Introduction to RL

Markov Decision Process (MDP)

A Markov Decision Process (MDP) is defined as a tuple:

$$\mathcal{M} = (S, s_1, \mathcal{A}, \mathcal{P}, \mathcal{R})$$

- S is a finite set of states
- s_1 is a unique state, representing the initial state $s_1 \in S$
- $\mathcal{A}$ is a set of actions $\{a_1, a_2, \ldots, a_n\}$
- $\mathcal{P} : S \times \mathcal{A} \to \text{DIST}(S)$ is the transition function
- $\mathcal{R} : S \times \mathcal{A} \times S \to \mathbb{R}$ is the reward function

The system works through interactions between an agent and the environment.

How Reinforcement Learning Works

Reinforcement Learning is a process where:

- An agent starts at an initial state s_1
- At each step t, it selects an action $a_t \in \mathcal{A}$
- The environment transitions to a new state s_{t+1} based on $\mathcal{P}(s_t, a_t)$
- The agent receives a numerical reward $r_{t+1} = \mathcal{R}(s_t, a_t, s_{t+1})$
- This process continues, generating a trajectory: $s_1, a_1, r_2, s_2, a_2, r_3, \ldots$

The agent's goal is to learn which actions to take in each state to maximize cumulative rewards.

Rewards and Agent's Objective

- The agent tries to maximize the **expected cumulative reward**

The return G_t from time t is:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Policy (π): A function $\pi : S \to \mathcal{A}$ mapping states to actions
The goal is to find an **optimal policy** π^* that maximizes the expected return from any state.

Why Safety is Important in RL

Standard RL only focuses on maximizing expected reward. But in many real-world applications, this can be dangerous:

- Agents may accept occasional punishments.
- Even a single exploration error could be catastrophic in critical domains
- Agents exploit unintended loopholes in reward functions.

Safety is not guaranteed unless it is explicitly enforced during learning and deployment.

Pre-requisites

Linear Temporal Logic (LTL)

LTL allows expressing temporal properties over traces.

- $X\varphi$ (next): φ holds in the next state
- $F\varphi$ (finally): φ eventually holds
- $G\varphi$ (globally): φ holds always
- $\varphi_1 \, U \, \varphi_2$ (until): φ_1 holds until φ_2

Use: Specify safety conditions in reactive systems.

Safety Games and Automata

A safety automaton: $\varphi_s = (Q, q_0, \Sigma, \delta, F)$

- $\delta : Q \times \Sigma \to Q$ is the transition function
- $F \subseteq Q$ is the set of safe states
- A trace σ induces a run $q_0 q_1 q_2 \ldots$ with $q_{i+1} = \delta(q_i, \sigma_i)$
- σ satisfies φ_s iff $\forall i,\ q_i \in F$

2-Player Alternating Game

Game Structure: A game is a tuple $G = (G, g_0, \Sigma_I, \Sigma_O, \delta, win)$ where:

- G: finite set of game states
- $g_0 \in G$: initial state
- Σ_I, Σ_O: input and output alphabets (environment and system)
- $\delta : G \times \Sigma_I \times \Sigma_O \to G$: transition function
- $win : G^\omega \to \mathbb{B}$: winning condition over infinite plays

Gameplay:

- At each step:
  - Environment chooses $\sigma_I \in \Sigma_I$
  - System chooses $\sigma_O \in \Sigma_O$
  - Next state: $g' = \delta(g, \sigma_I, \sigma_O)$
- The resulting infinite sequence $g_0 g_1 g_2 \ldots$ is a *play*

Strategy and Winning:

- A (memoryless) strategy: $\rho : G \times \Sigma_I \to \Sigma_O$
- The *winning region* $W \subseteq G$: states from which a winning strategy exists

Shielding

Shielding in Reinforcement Learning

Shielding integrates a **correct-by-construction reactive system** into RL.

- The shield monitors agent actions.
- If an action is unsafe (w.r.t. system specifications), it is overwritten.
- Ensures safety with minimal interference.

Goal: Prevent unsafe decisions during both learning and execution.

Preemptive Shielding – Mechanism

At each time step t:

1. The shield computes safe actions $\{a_t^1, a_t^2, \ldots, a_t^k\}$ by filtering out those violating φ_s.
2. The agent selects $a_t \in \{a_t^1, \ldots, a_t^k\}$.
3. The environment executes a_t and returns s_{t+1}, r_{t+1}.

The shield dynamically restricts the agent's choices to safe ones.

Preemptive Shielding – Formal Model

The MDP $\mathcal{M} = (S, s_1, A, P, R)$ is transformed into:

$$\mathcal{M}_0 = (S_0, s_1, A_0, P_0, R_0)$$

- $S_0 = S \times Q$: product of MDP and shield states
- $A_0(s, q) \subseteq A$: only safe actions allowed
- P_0: transitions for safe actions only
- R_0: same as R, restricted to A_0

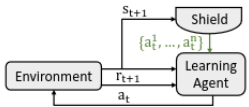


Fig. 6: Preemptive Shielding.

Ensures that unsafe actions are never executed.

Post-Posed Shielding

Workflow:

- Agent picks action a_t.
- If a_t is unsafe w.r.t. φ_s, the shield replaces it with a safe action a'_t.
- The environment executes a'_t and returns s_{t+1}, r_{t+1}.
- The agent updates policy based on a'_t, r_{t+1}.

Key Point: Shield intervenes only after the agent decides, but before execution.

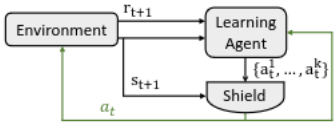


Fig. 7: Post-Posed Shielding.

Enhancing Post-Posed Shielding with Ranked Actions

When the agent provides a ranked list of preferred actions:

- The shield examines the list in order of priority.
- It *selects the first action* that is safe (i.e. leads to a state in the winning region W).
- Only if *all* ranked actions are unsafe does it pick an arbitrary safe fallback.

Note: By honoring the agent's own ranking, the shield minimizes interference and better preserves the learner's intended policy, improving overall learning performance.

Reward Handling in Post-Posed Shielding: Comparison

1. Punishment Approach

- Assign a negative punishment $r'_{t+1} < 0$ to the unsafe action a_t^1.
- Agent learns that selecting a_t^1 at state s_t is unsafe.
- **Pros:**
  - Encourages the agent to avoid unsafe actions over time.
- **Cons:**
  - No guarantee that unsafe actions won't appear in the final policy.
  - Shield must remain active *during* and *after* learning.

2. Reward Passing Approach

- Assign the true reward r_{t+1} to the unsafe action a_t^1.
- Agent updates a_t^1 as if it were executed safely.
- **Pros:**
  - Learning performance unaffected by safety overrides.
  - Agent need not learn to explicitly avoid unsafe actions.
- **Cons:**
  - Unsafe actions may become part of an optimal policy.
  - Shield must remain active *during* both learning and execution.

Advantages of Post-Posed Shielding

- **Transparent to the Agent:** The learning algorithm only observes the MDP state; it never "sees" the shield.
- **Works in Execution Phase:** Can be applied to a fixed (pre-trained) policy without retraining.
- **Minimal Intrusion:** Only corrects actions when they would violate safety—otherwise leaves the agent's choice intact.
- **Easy Integration:** No changes needed to the learning algorithm; simply wrap the shield around the agent.

Synthesis of the Shield

Overview of the Synthesis Process

Now given a

Step 1 : Abstraction of the MDP

Given:

- MDP $\mathcal{M} = (S, s_I, A, P, R)$
- Observer function $f : S \to L$, mapping MDP states to finite abstract labels

We define a deterministic safety word automaton (DSWA):

$$\varphi_M = (Q, q_0, \Sigma, \delta, F)$$

- Q: abstract states tracking observed action-label histories
- q_0: initial state (empty history)
- $\Sigma = A \times L$: alphabet of action-label pairs
- $\delta : Q \times \Sigma \to Q$: transition function
- $F \subseteq Q$: safe states (where observed history is valid)

States and Transitions in the Abstraction

States $q \in Q$:

- Represent possible sets of MDP states (or their labels) compatible with the observed sequence $(a_0, \ell_0), (a_1, \ell_1), \ldots$
- Track how the MDP could evolve without knowing full probabilities

Transition Function $\delta(q, (a, \ell)) = q'$:

- Models the effect of taking action a in a state labeled ℓ
- q' includes labels $f(s')$ of all states s' such that $P(s, a, s') > 0$, for some $s \in S$ with $f(s) = \ell$
- Ensures abstract state updates reflect all possible MDP evolutions

Final States and Safety in the Abstraction

Final States $F \subseteq Q$:

- Each $q \in F$ represents a safe abstract configuration
- All observed action-label traces leading to q are feasible in the original MDP

Safety Enforcement:

- If a transition leads to $q \notin F$, the trace is rejected
- This models safety violations (e.g., invalid action in a given abstract label)

Well-formedness Assumption:

- No state in F allows only transitions that eventually exit F
- Ensures invalid behavior is caught as early as possible

Example: MDP and Automaton

- MDP:
  - $s_0 \xrightarrow{a} s_1,\ f(s_0) = l_0,\ f(s_1) = l_1$
  - $s_1 \xrightarrow{a} s_2,\ f(s_2) = l_2$
- Automaton:
  - $q_0 \xrightarrow{(a,l_0)} q_1$
  - $q_1 \xrightarrow{(a,l_1)} q_2$
  - $q_2 \xrightarrow{(a,l_2)} q_{\text{rej}} \notin F$
- If input trace contains (a, l_3) with l_3 not reachable $\rightarrow$ trace is rejected

Rejection of Impossible Traces

- Intuition: φ_M only accepts traces consistent with the MDP dynamics
- Impossible MDP transitions $\rightarrow$ δ undefined or trace leads outside F
- $\Rightarrow$ Automaton immediately rejects sequences that:
  - Deviate from any feasible execution of M
  - Do not preserve the label-action consistency derived from f
- Hence, accepted traces form a conservative over-approximation of feasible behaviors

Step 2: From Safety LTL to a Safety Automaton

Input: A safety LTL formula φ_s **Translation:** Convert φ_s into a *deterministic safety word automaton*

$$\varphi_s = (Q, q_0, \Sigma, \delta, F)$$

- $\Sigma = 2^I \times 2^O$ is the alphabet of input/output valuations.
- Q is a finite set of monitoring states.
- $q_0 \in Q$ is the initial state (no violation seen yet).
- $\delta : Q \times \Sigma \to Q$ updates the monitor on each step.
- $F \subseteq Q$ are the *safe* states—states from which no bad prefix has occurred.

Acceptance (Safety): An infinite trace $\sigma = \sigma_0\sigma_1 \cdots$ satisfies φ_s iff the induced run

$$q_0 \xrightarrow{\sigma_0} q_1 \xrightarrow{\sigma_1} q_2 \cdots$$

never leaves F, i.e. $\forall i : \; q_i \in F$.

Note: Any violation of the LTL spec causes a transition out of F, enabling early

Step 3: Constructing the Safety Game

Given: Safety automaton φ_s and Abstraction automaton φ_M
We define the two-player safety game

$$G = (\, G,\ g_0,\ \Sigma_I,\ \Sigma_O,\ \delta,\ F_g)$$

with:

$$\begin{aligned} G &= Q \times Q_M, \\ g_0 &= (\, q_0,\ q_{0,M}\,), \\ \Sigma_I &= L, \\ \Sigma_O &= A, \\ \delta\big((q, q_M),\, \ell,\, a\big) &= \big(\, \delta(q, (\ell, a)),\ \delta_M(q_M, (\ell, a))\big), \\ F_g &= (F \times Q_M)\ \cup\ \big(Q \times (Q_M \setminus F_M)\big). \end{aligned}$$

Gameplay: $\ell \in L$ chosen by the environment; $a \in A$ chosen by the system. Safe states $(q, q_M) \in F_g$ occur whenever the safety spec is safe *or* the abstraction has already failed (and thus need not be enforced).

Step 4 : Solving the Safety Game

This "cross–product" lets us reason about both the safety specification and the abstract MDP together.

Safe States F_g:

$$F_g = (F \times Q_M) \cup (Q \times (Q_M \setminus F_M))$$

A combined state is "safe" if

- the safety spec is still in a safe state, or
- the abstraction has already failed (so we no longer require enforcement).

This basically means we have a list of all safe states in the MDP. Now we just need to remove those, which have a forced transition to unsafe state.

Step 4: Solving the Safety Game

Winning Region $W \subseteq F_g$: The subset of F_g from which the system player can **guarantee** (against any environment moves) to stay within F_g forever.

- Computed by standard safety-game algorithms (fixed-point/backward elimination).
- Intuitively, W is the "core" of states where a safe strategy exists.

Key Insight: If we wrap our agent in a shield that only allows transitions within W, then no matter what the environment does (as long as it respects the abstraction), we can never violate the safety specification.

Step 5: Synthesizing the Preemptive Shield

Goal: Turn the safety game G and its winning region W into a reactive "shield" S that filters out unsafe actions.

Shield as a Reactive System

$$S = (Q_S,\, q_{0,S},\, \Sigma_{I,S},\, \Sigma_{O,S},\, \delta_S,\, \lambda_S)$$

with:

- $Q_S = G$ — each shield state is a pair (q, q_M) from the product game.
- $q_{0,S} = g_0$ — start in the game's initial state.
- $\Sigma_{I,S} = L \times A$ — inputs: the abstract observation $\ell = L(s)$ and the agent's *previous* action a (to advance the game).
- $\Sigma_{O,S} = 2^A$ — outputs: the *set* of all actions the agent is allowed to pick next.
- $\delta_S(g, \ell, a) = \delta(g, \ell, a)$ — update the shield's state exactly as the game would.
- $\lambda_S(g, \ell) = \{\, a \in A \mid \delta(g, \ell, a) \in W \}$ — allow exactly those actions that keep the game inside the winning region.

Intuition for the synthesis

Intuition: At each step, the shield "looks ahead" one move in the game and returns only those actions that are guaranteed safe (i.e. they lead to states from which a safe strategy exists). The agent then chooses freely among these safe actions.

Preemptive Shield: Intuitive Operation

1. **Keeping Track of "Where We Are"**
   The shield maintains its own internal state by observing the current abstract label and the action the agent just took.
2. **Looking One Move Ahead**
   It simulates each possible next action and checks whether it would keep us in the safe region of the safety game.
3. **Filtering the Agent's Choices**
   Only those actions that pass the safety check are returned; unsafe actions are never shown to the agent.
4. **Letting the Agent Play (Safely)**
   The agent picks one action from this safe list, the environment executes it, and the cycle repeats.

Preemptive Shield: Ensuring Safety

- **Step-wise Safety:** Every allowed action a_t satisfies $\delta(q_t, (\ell_t, a_t)) \in W \subseteq F_g$.
- **Inductive Invariant:** Since W is closed under all environment moves and shield updates, once $q_t \in W$, always $q_{t+1} \in W$.
- **Safety Guarantee:** $W \subseteq F_g$ means the safety automaton component of the game never leaves its safe states F. Hence the infinite trace $s_0 a_0 s_1 a_1 \cdots$ always satisfies the safety specification φ_s.

Water Tank Example

Example Setup

- Tank capacity: max 100 liters
- Inflow and outflow: 0–1 liters/sec
- Inflow when valve open: 1–2 liters/sec
- Valve setting must persist at least 3 seconds
- Avoid overflow (over 100L)
- Avoid running dry (below 0L)

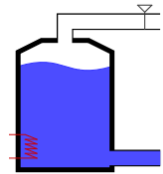


Fig. 3: A hot water storage tank with an inflow, an outflow, and a tank heater.

$$
\begin{aligned}
&\mathsf{G}(\mathit{level} > 0) \\
\wedge\ &\mathsf{G}(\mathit{level} < 100) \\
\wedge\ &\mathsf{G}((\mathit{open} \wedge \mathsf{X}\mathit{close}) \rightarrow \mathsf{XX}\mathit{close} \wedge \mathsf{XXX}\mathit{close}) \\
\wedge\ &\mathsf{G}((\mathit{close} \wedge \mathsf{X}\mathit{open}) \rightarrow \mathsf{XX}\mathit{open} \wedge \mathsf{XXX}\mathit{open})
\end{aligned}
$$

What we want out a shield for this process

- Open inflow valve when tank level is too low, ensuring it reaches at least level 4
- Prevent inflow valve from opening if the tank level is above 93 to avoid overflow risk
- Enforce minimum 3-second delay between state changes to avoid excessive wear on the valve

Water Tank Safety Automata

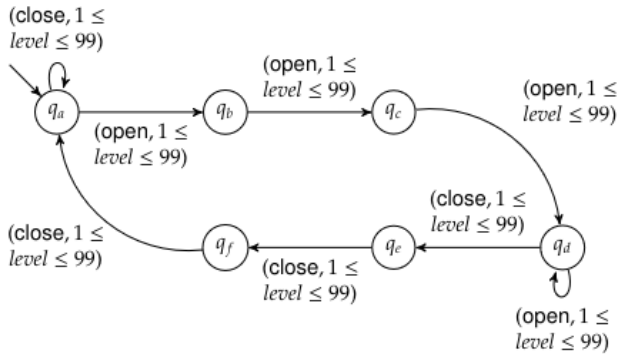


Figure: Safety Automaton made from LTL

MDP Abstraction

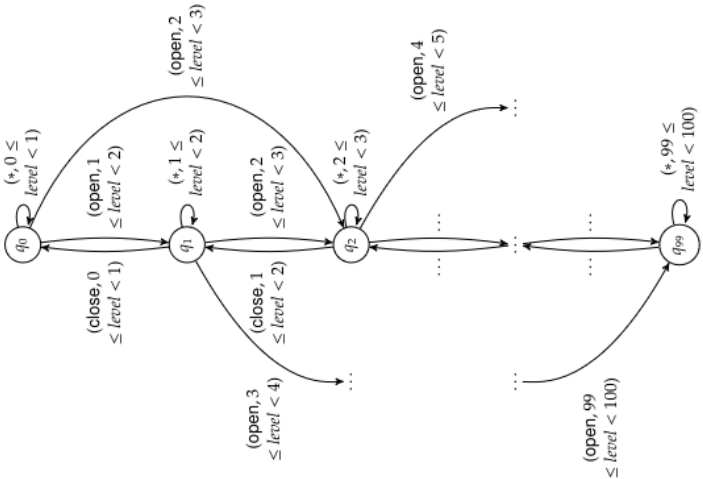
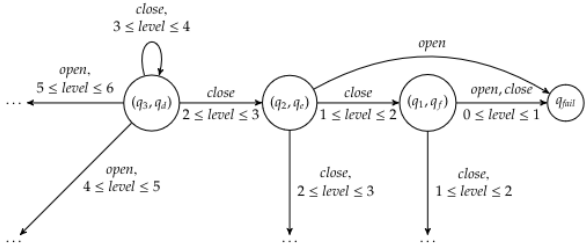


Figure: Abstract MDP

Safety Game



We can see that, in state (q3, qd), the system should not choose action close, as otherwise the system cannot avoid to reach q(fail). It could be the case that q(fail) is actually not reached (when the environment chooses to let the level stay the same for a step), but we cannot be sure because we have to consider all evolutions of the environment to be possible that are consistent with our abstraction. Thus, the shield needs to deactivate the close action in state (q3, qd).

Experiment

Experiments

- Two environments:
  - **9x9 Grid:** bombs, walls, colored regions
  - **15x9 Grid:** opponent agent and two colored targets
- **Agent Objective:** visit all colored regions in the correct order
- **Safety Specifications:**
  - φ_s^1: avoid walls and moving agent (used in both grids)
  - φ_s^2: avoid staying on a bomb for >2 steps (9x9 only)

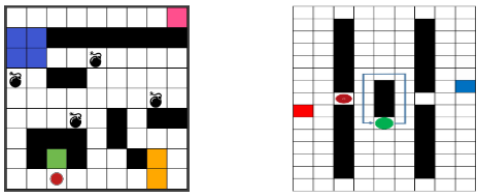


Figure: 9x9 and 15x9 grid world environments

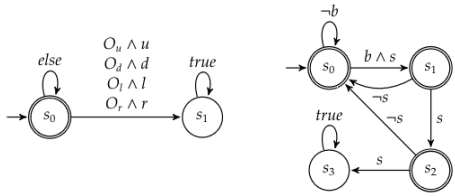


Fig. 10: DFAs for φ_1^s (left) and φ_2^s (right).

Figure: DFAs for safety specifications φ_s^1 and φ_s^2

Shielded vs Unshielded Learning: Results

- **Shield Synthesis Time:**
  - 9x9 grid: $\varphi_s^1 \wedge \varphi_s^2 \rightarrow$ 2 seconds
  - 15x9 grid: φ_s^1 only $\rightarrow$ 0.6 seconds
- **Learning Outcomes (Fig. 11):**
  - Shielded agents avoid unsafe actions and learn faster
  - Only unshielded agents receive negative rewards
  - Post-posed shield with $|\text{rank}_t| = 3$ and no penalty yields optimal reward
  - Penalty-based learning or $|\text{rank}_t| = 1$ leads to suboptimal policies

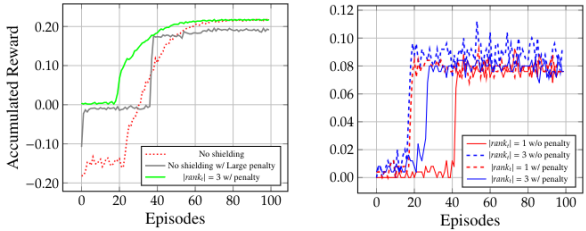


Fig. 11: The accumulated reward per episode for the 9x9 (left) and the 15x9 (right) grid worlds