Safe Reinforcement Learning via Shielding

Mohammed Alshiekh, Roderick Bloem, Rudiger Ehlers, Bettina Konighofer, Scott Niekum, Ufuk Topcu

Presented by Karim Ghonim and Hossam Arafat

Sapienza University

July 2, 2019

Overview

1. Motivation
2. Introduction
3. RL
4. MDP
5. Synthesis of Shields
6. Linear Temporal Logic - LTL
   - Introduction to LTL
   - LTL Safety Specification and automata
7. Synthesis through Safety Games
8. Shielded Reinforcement Learning
9. Experiments

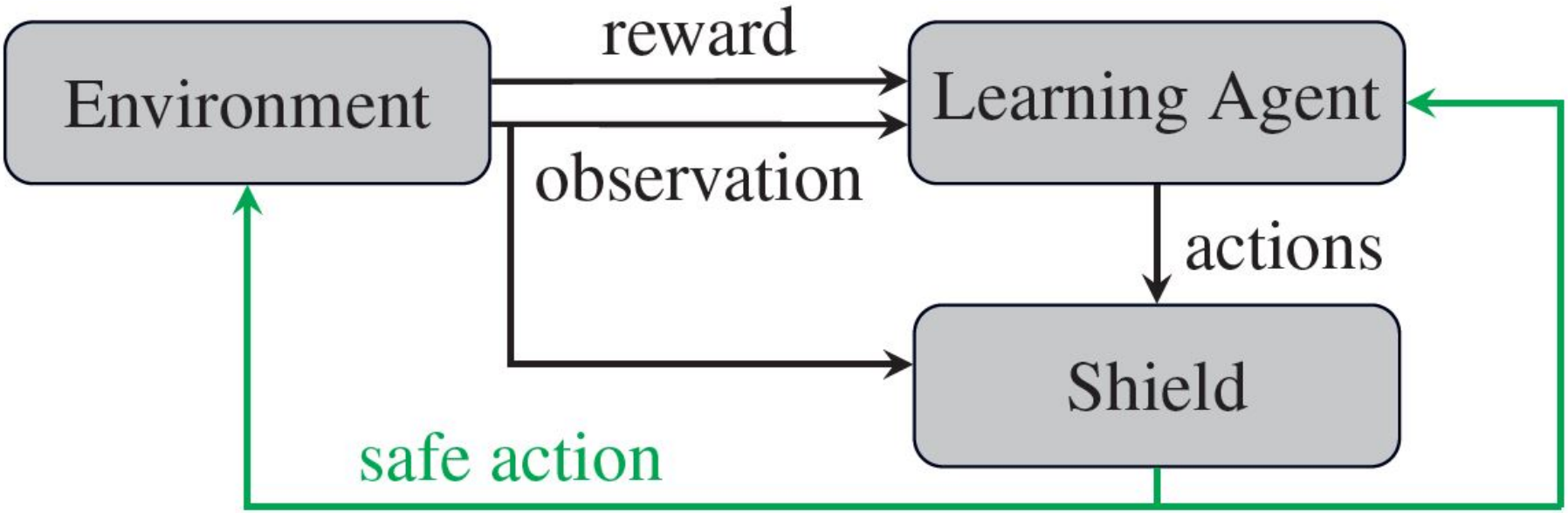
Motivation

- Increasing use of learning-based controllers in physical systems in the proximity of humans strengthens the concern of whether these systems will operate safely
- Safety or more generally correctness during learning and execution of controllers has attracted significantly less attention than optimality and convergence
- How can we let a learning agent do whatever it is doing, and also monitor and interfere with its operation whenever absolutely needed in order to ensure safety?

What is "Safe" Reinforcement Learning?

- A framework that allows applying machine learning to control systems in a way that the correctness of the systems execution is guaranteed
- The shield monitors the actions selected by the learning agent and corrects them if and only if the chosen action is unsafe
- Real-life application
  - Autonomous Vehicles
- How do we define "Correctness"?

Shielded Reinforcement Learning



Reinforcement Learning

- **Agent and Environment**: the agent operates in an environment and obtains a *reward* after every performed action
- **Goal**: an agent must learn an optimal policy through trial-and-error via interactions with an unknown environment modeled by an MDP
- **Q-function**: gives a measure of how good action a is in state s

$$Q^{\pi}(s,a) \equiv r(s,a) + \gamma \sum_{t=1}^{T-1} \gamma^{t-1} r_t$$

where:

Q: Value of taking action a in state s

r: Reward obtained from taking action a in state s

t: The current time step γ: Discount Factor

T: The total number of steps per episode

- **Optimal Policy**: computed as

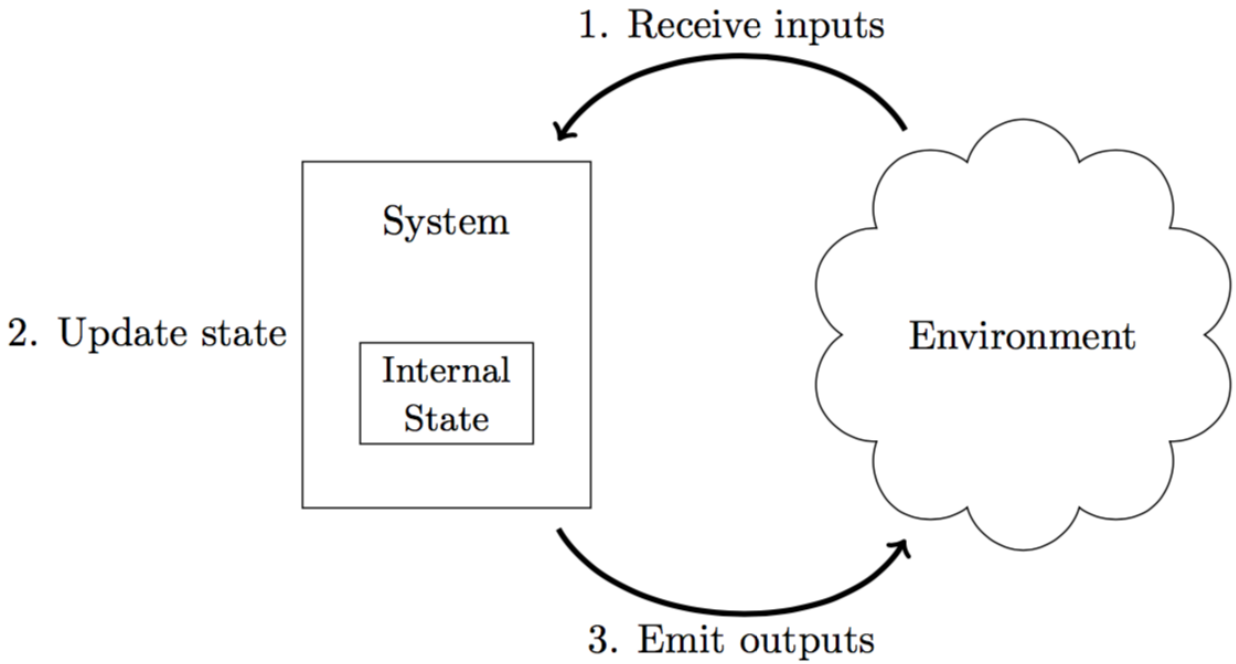
$$\pi^{\star}(s) = \arg\max_{a \in A} Q^{\star}(s,a)$$

Markov Decision Process

- Defined as $M =< S, s_I, A, P, R >$ where
  - S is a set of states
  - s_I is the initial state
  - A is a set of actions
  - P : transition function $P(s_{t+1}|s_t, a_t)$
  - $R : S \times A \times S \rightarrow \mathbb{R}$
- Solution to **MDP** called policy $\pi : S \rightarrow A$ that maximizes the cumulative discounted reward
- **RL** is the task of learning a possibly optimal policy, from an initial state s_I, on an MDP where only S and A are **known**, while P and R are **not known**.

Synthesis of Shields

Given: An LTL formula φ as specification with $Prop = \mathcal{I} \cup \mathcal{O}$.



Obtain: A model of a reactive system that **realizes** φ.

- For every sequence of inputs, the trace of the system satisfies φ.

Finite Reactive Systems

- Defined as $\mathcal{S} =< Q, q_0, \Sigma_I, \Sigma_O, \delta, \lambda >$ where
  - Q is a finite set of states
  - $q_I \in Q$ is the initial state
  - Σ_I : is the input alphabet
  - Σ_O : is the output alphabet
  - $\delta : Q \times \Sigma_I \rightarrow Q$ is the transition function
  - $\lambda : Q \times \Sigma_I \rightarrow \Sigma_O$ is the output function

Linear Temporal Logic - LTL

Allows expressing temporal patterns about a property (**over infinite traces**)

Syntax

$X\varphi$: φ is true in the next moment in time
$G\varphi$: φ is true forever
$F\varphi$: φ is true eventually
$\varphi \mathcal{U} \psi$: φ is true until ψ is true

Example

$$G(level > 0) \wedge G(level < 100) \wedge$$
$$\mathrm{G}((\mathrm{open} \wedge Xclose) \Rightarrow XXclose \wedge XXXclose) \wedge$$
$$\mathrm{G}((\mathrm{close} \wedge Xopen) \Rightarrow XXopen \wedge XXXopen)$$

LTL Introduces

Safety Specification φ^S: something bad should never happen
$G\neg(WaterLevel > 93)$

φ^S defines a set of allowed traces

The reactive system $\mathcal{S}$ realizes φ^S if for every sequence of inputs, the trace of the system realizes φ^S

LTL Safety Specification and **automata**

LTL safety specifications can be translated into a *safety word automaton* that represents it.

A DFA is a tuple $\varphi^s = \langle Q, q_0, \Sigma, \delta, F \rangle$

Q	is the finite set of states
$q_0 \in Q$	is the initial state
$\sum$	is the input alphabet
$F \subseteq Q$	is the set of safe states
$\delta : Q \times \Sigma \rightarrow S$	is the transition function

A *run* induced by the trace $\sigma = \{\sigma_0, \sigma_1, \sigma_2 ... \sigma_m\}$ is a sequence of states $q = q_0, q_1 ...$ such that $q_{i+1} = \delta(q_i, \sigma_i)\ \forall i \in N$.
A trace σ of a system S satisfies φ^s if the induced run visits only **safe** states, i.e., $\forall i \geq 0. q_i \in F$.

Example: Resource Arbiter

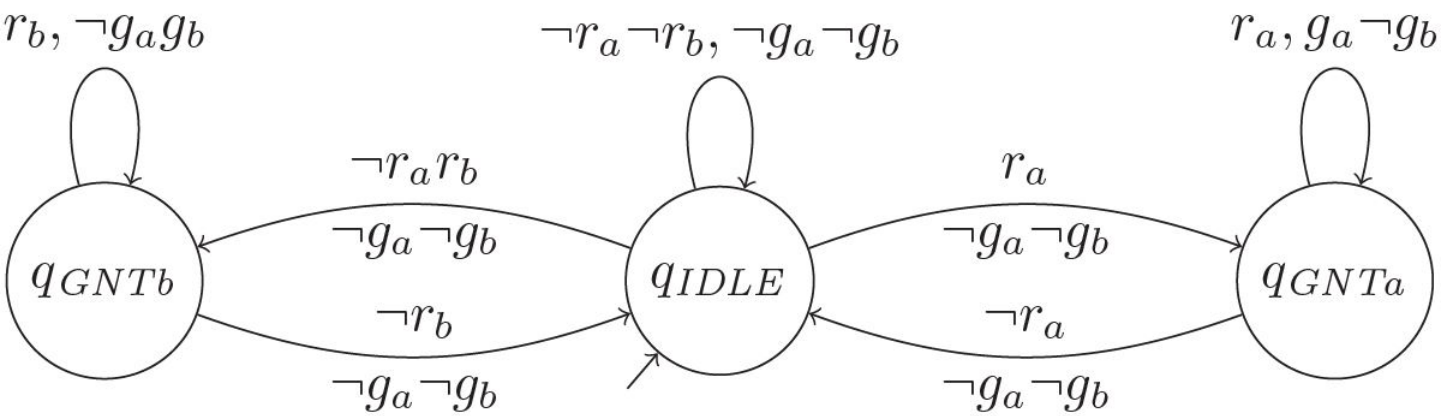
Synthesis of an arbiter:

- Two clients issue **requests** r_1, r_2 for access to shared resources.
- The arbiter produces **grants** g_1, g_2 granting access to the resources.

$$\varphi = G(\neg g_1 \vee \neg g_2) \wedge G(r_1 \Rightarrow F g_1) \wedge G(r_2 \Rightarrow F g_2)$$

The **specification** φ of the arbiter is the conjunction of:

1. **Mutual exclusion:** at no point in time should both g_1 and g_2 be true.
2. **Response:** every request r_i from client i (for $i \in \{1, 2\}$) should eventually be followed by grant g_i for client i.



Abstraction

- Planning ahead does not require the environment dynamics to be completely known in advance
- A (coarse finite-state) abstraction of the environment dynamics is needed to reason about when exactly a specification violation cannot be avoided
- Abstraction has to be conservative w.r.t. the behavior of the real MDP
- Abstraction may have finitely many states even if the MDP has infinitely many states

Abstraction

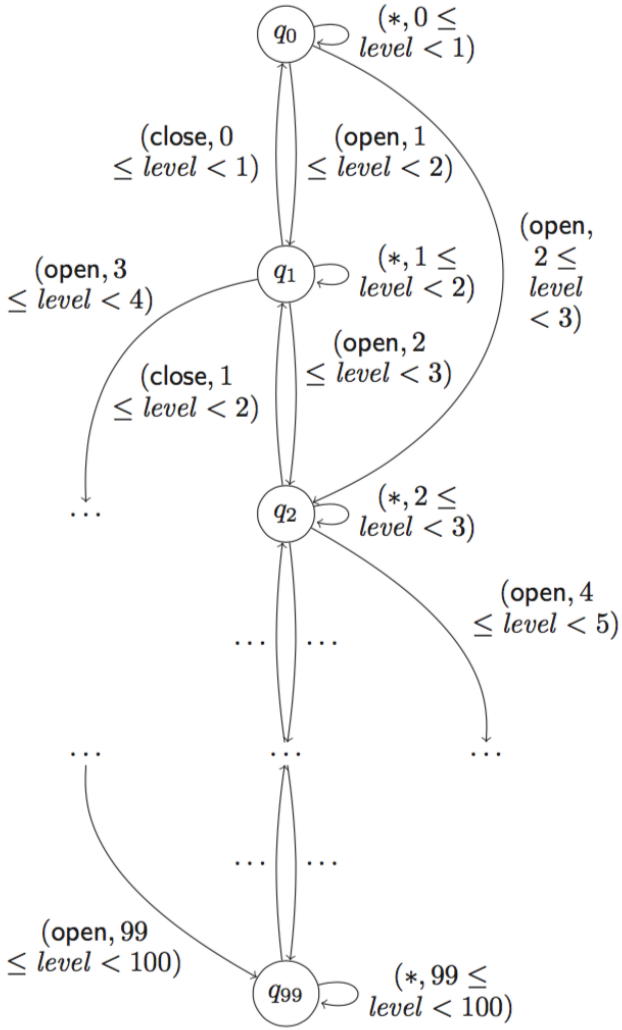
Given an MDP $M =< S, s_I, A, P, R >$ and an *MDP Observer Function* $f : S \to L$

An Abstraction is a tuple $\varphi^{\mathcal{M}} = \langle Q_{\mathcal{M}}, q_{0,\mathcal{M}}, \Sigma, \delta_{\mathcal{M}}, F_{\mathcal{M}} \rangle$

$Q_{\mathcal{M}}$	is the finite set of states
$q_{0,\mathcal{M}} \in Q$	is the initial state
$\Sigma = A \times L$	is the input alphabet
$F \subseteq Q$	is the set of safe states

The transition function, $\delta_{\mathcal{M}}$, specifies the next state as, for every trace $s_0, s_1 ... \in S$ with the corresponding action sequence $a_0, a_1 ... \in A$ of the MDP, for every automaton run $q = q_0, q_1 ... Q$ of $\varphi^{\mathcal{M}}$: $q_{i+1} = \delta_{\mathcal{M}}(q_i, l_i, a_i))$ for $l_i = L(s_i)\ \forall \in \mathbb{N}$

Example


q_0
$(*, 0 \leq level < 1)$
$(\text{close}, 0 \leq level < 1)$
$(\text{open}, 1 \leq level < 2)$
$(\text{open}, 2 \leq level < 3)$
q_1
$(*, 1 \leq level < 2)$
$(\text{open}, 3 \leq level < 4)$
$(\text{open}, 2 \leq level < 3)$
$(\text{close}, 1 \leq level < 2)$
q_2
$(*, 2 \leq level < 3)$
$(\text{open}, 4 \leq level < 5)$
$(\text{open}, 99 \leq level < 100)$
q_{99}
$(*, 99 \leq level < 100)$

Synthesis through Safety Games

- Translate φ^s and $\varphi^{\mathcal{M}}$ to a safety game $\mathsf{G} = (\mathsf{G}, g_0, \Sigma_I, \Sigma_O, \delta, F^g)$ between two players.
- Compute the winning region $\mathsf{W} \subseteq F^g$ of G
- Translate G and W to a reactive system $\mathsf{S} = (Q_S, q_{0,\mathcal{S}}, \Sigma_{I,S}, \Sigma_{O,S}, \delta_S, \lambda_S)$, that constitutes the shield.

Interaction of Shield & Agent

The shield monitors the actions of the agent, and substitutes the selected actions by safe actions whenever this is necessary to prevent the violation of φ^S

Shield

- In each step t, the agent selects an action a_t^1
- The shield forwards a_t to the environment, i.e., $a_t = a_t^1$.
  Only if a_t^1 is unsafe w.r.t. φ^S, the shield selects a different action $a_t \neq a_t^1$ instead.
- The environment executes a_t, moves to s_{t+1} and provides r_{t+1}.
- The agent receives a_t and r_{t+1}, and updates its policy for a_t using r_{t+1}

Rewards for **un**safe Actions

- Assign a punishment $r'_{t+1} < 0$ to a_t^1
  - The agent learns that selecting a_t^1 at state s_t is unsafe, without ever violating φ^S
  - No guarantee that unsafe actions are not part of the final policy
  - Shield has to remain active even after the learning phase
- Assign the reward r_{t+1} to a_t^1
  - Picking unsafe actions can likely be part of an optimal policy by the agent
  - Since an unsafe action is always mapped to a safe one, this does not pose a problem and the agent never has to learn to avoid it
  - Shield is (again) needed during the learning and execution phases

Experiments

Demonstration

Jupyter Notebook