

Safety and Reliability for Learning Systems

Part IV - Enforcing the correctness of systems that adapt at runtime.

Rüdiger Ehlers, Clausthal University of Technology

Marktobersdorf Summer School, August 2019

This part is based on joint work with Mohammed Alshiekh, Roderick Bloem, Bettina Könighofer, Scott Niekum, Ufuk Topcu, and Min Wen

Reinforcement learning

Reinforcement learning

- A method to allow a system to adapt on its own to an unknown or changing environment.
 - calibration and wear-off compensation for CPS
 - adaptation to environment changes

Reinforcement learning

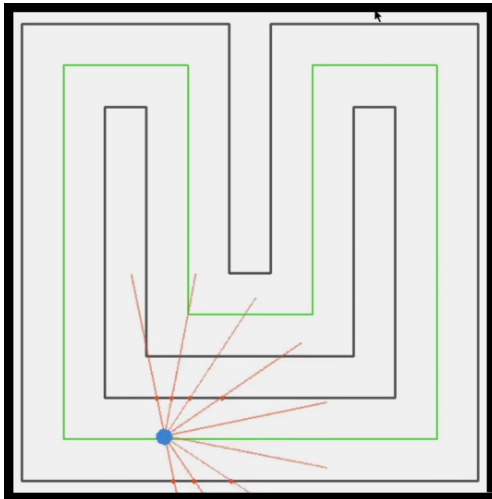
Reinforcement learning

- A method to allow a system to adapt on its own to an unknown or changing environment.
 - calibration and wear-off compensation for CPS
 - adaptation to environment changes

Problems

- Quality assurance of such systems is very difficult: how do you assure the quality of something that is not known at design time?

Example: Car driving



Video and implementation by Bettina Könighofer, TU Graz

Traditional safety in reinforcement learning

Reinforcement learning modes

- Exploration mode (in which the system adapts)
- Exploitation mode (in which the computed control policy is exploited)

Traditional safety in reinforcement learning

Reinforcement learning modes

- Exploration mode (in which the system adapts)
- Exploitation mode (in which the computed control policy is exploited)

Safety

A mean summary: For safety-critical systems, after the end of the exploration mode, the learned behavior should be good enough.

Problems with this definition

- Relies on external measures to make the exploration mode safe
- Does not allow runtime adaptation

Types of approaches

Two streams of approaches

- Constrain the learner to only learn from the set of *safe* behaviors
- Correct the learner's choices.

Types of approaches

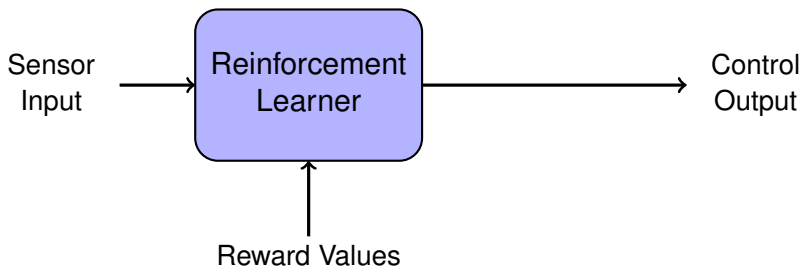
Two streams of approaches

- Constrain the learner to only learn from the set of *safe* behaviors
- Correct the learner's choices.

In the following...

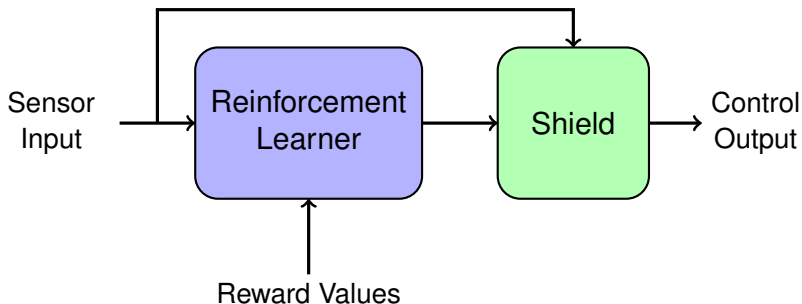
...we consider the second type of approach.

“Shielding” a learner



As published by Alshiekh et al. (2018)

“Shielding” a learner



As published by Alshiekh et al. (2018)

Formal definition of a shield

Discrete-time version

Let

- \mathcal{X} be a set of states of a system,
- $x_0 \in \mathcal{X}$ be an initial state,
- \mathcal{I} be a set of control inputs,
- $f : \mathcal{X} \times \mathcal{I} \rightarrow 2^{\mathcal{X}}$ be an evolution function of the environment,
and
- $\psi \subseteq \mathcal{X}^\omega$ be a specification for the system.

A shield is a function $g : \mathcal{X}^* \times \mathcal{I} \rightarrow \mathcal{I}$ such that for all traces $x_0 i_1 x_1 i_2 \dots \in (\mathcal{X} \times \mathcal{I})^\omega$, if for all $j \in \mathbb{N}_{>0}$ we have $i_j = g(x_0 \dots x_{j-1}, i')$ for some $i' \in \mathcal{I}$ and $x_j \in f(x_{j-1}, i_j)$, then $x_0 x_1 x_2 \dots \in \psi$.

In other words

Shield definition

A shield is a *controller* enforcing a specification.

There is however the soft requirement that for as many situations as possible, the shield returns $g(x_1 \dots x_n, i) = i$.

Difficulty of the problem

Difficulty

Thus, shield synthesis is as hard as controller synthesis.

However, by delegating the optimization of an implementation to the reinforcement learner, shields can be much more efficient to compute. Also, it often suffices to shield against the safety hull of a specification.

Difficulty of the problem

Difficulty

Thus, shield synthesis is as hard as controller synthesis.

However, by delegating the optimization of an implementation to the reinforcement learner, shields can be much more efficient to compute. Also, it often suffices to shield against the safety hull of a specification.

Needed

- Environment model
- Specification

Difficulty of the problem

Difficulty

Thus, shield synthesis is as hard as controller synthesis.

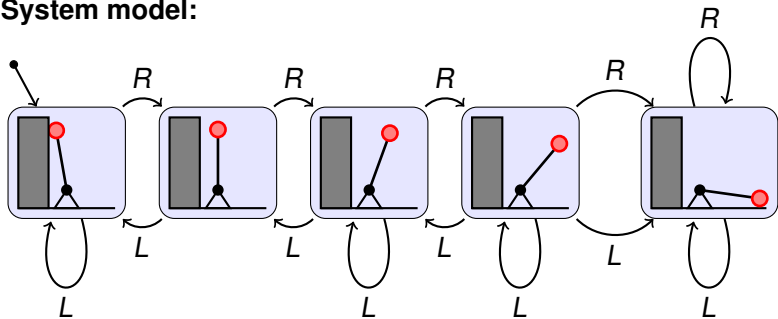
However, by delegating the optimization of an implementation to the reinforcement learner, shields can be much more efficient to compute. Also, it often suffices to shield against the safety hull of a specification.

Needed

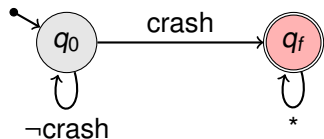
- Environment model ← can be overapproximated
- Specification ← can be underapproximated

A game-based approach to shield synthesis (1)

System model:

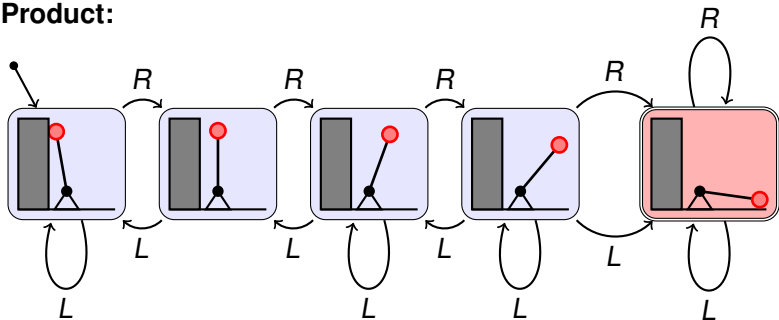


Specification:



A game-based approach to shield synthesis (2)

Product:



Safety games for shielding

Basic idea

What we have is a **game**.

Starting from the initial position, a *play* evolves by the *system player* making repeated action choices from $\{L, R\}$ after which the *environment player* always chooses one outgoing edge.

The system player wins if and only if an error state is never reached.

Solving a game

Solving a game

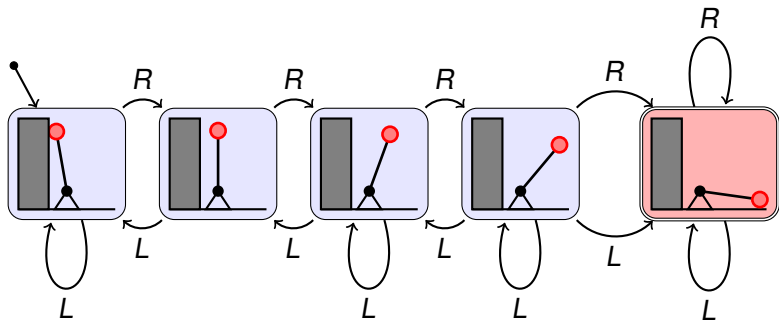
A position in the game is *losing* if:

- it is an error position *or*
- there is no action choice for the system player to avoid a transition to a position that is already known to be losing or is an error position.

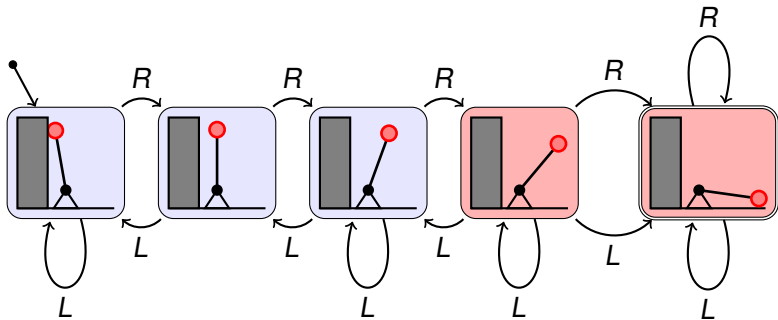
Idea

Starting with the set of error positions as the set of losing positions, *saturate* the set of losing positions using the rules above.

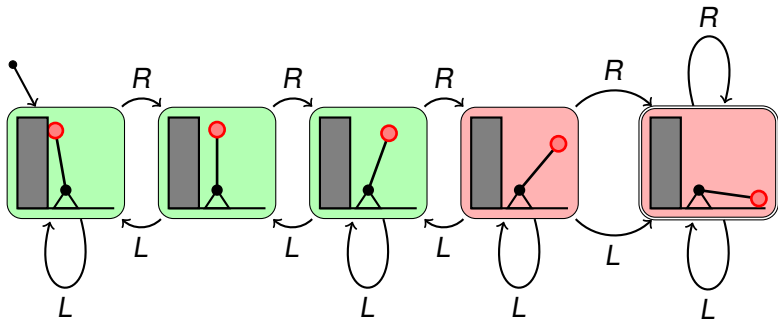
A game-based approach to shield synthesis (3)



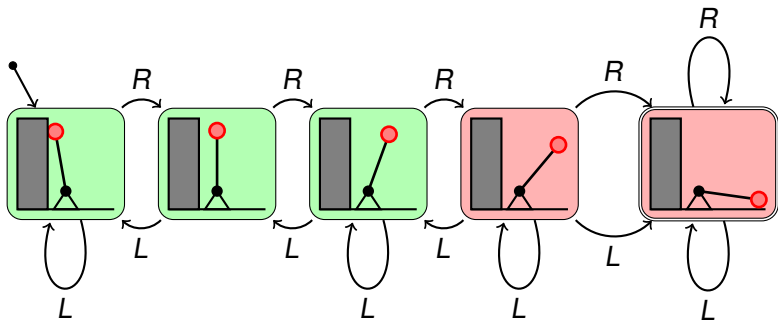
A game-based approach to shield synthesis (3)



A game-based approach to shield synthesis (3)



A game-based approach to shield synthesis (3)

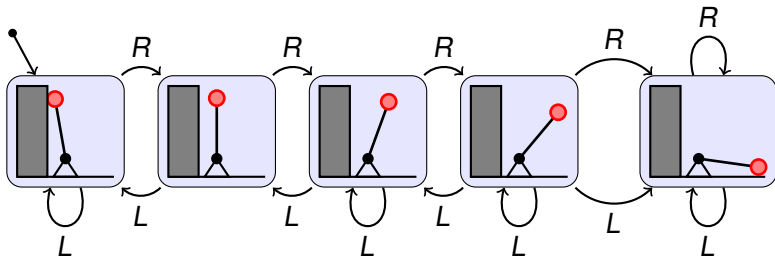


Shield operation

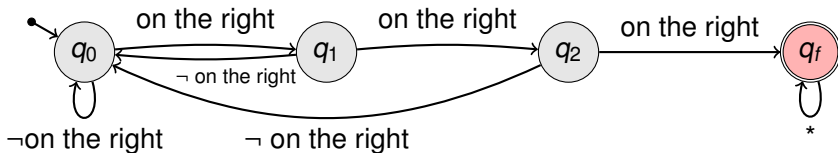
- It keeps track of in which game position the environment is (using sensor input)
- It modifies a learner's output (if needed) to one that is guaranteed to keep the game in the set of winning positions.

A more complex specification (1)

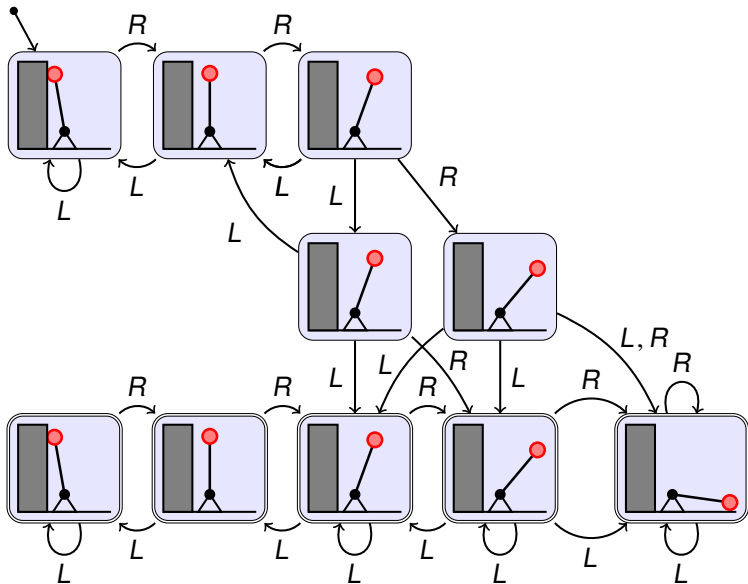
System model:



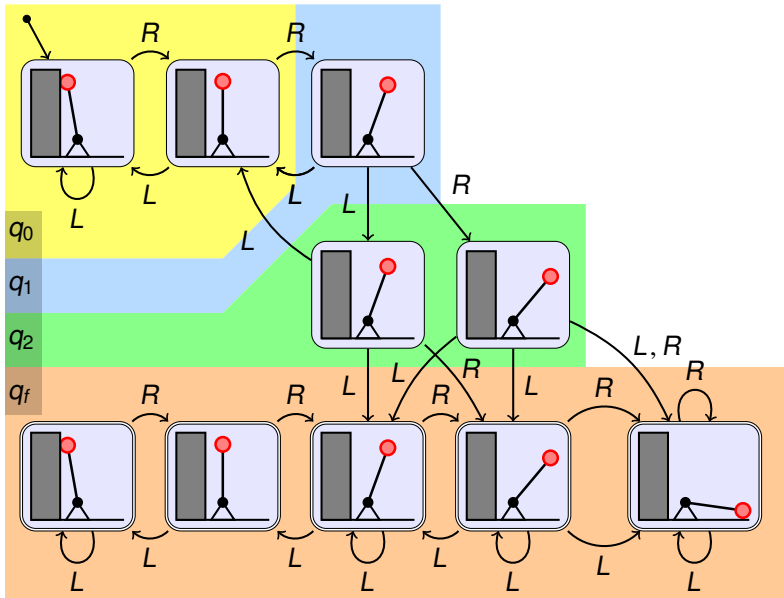
Specification:



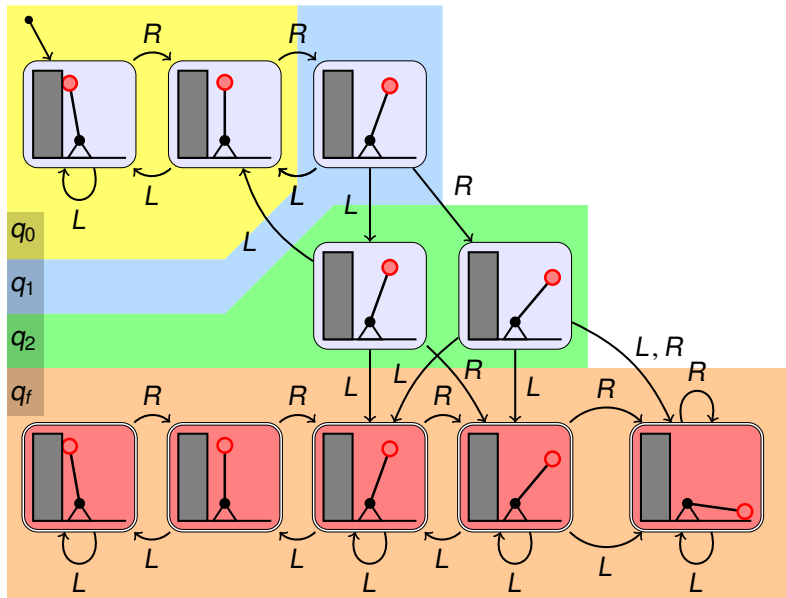
A more complex specification (2)



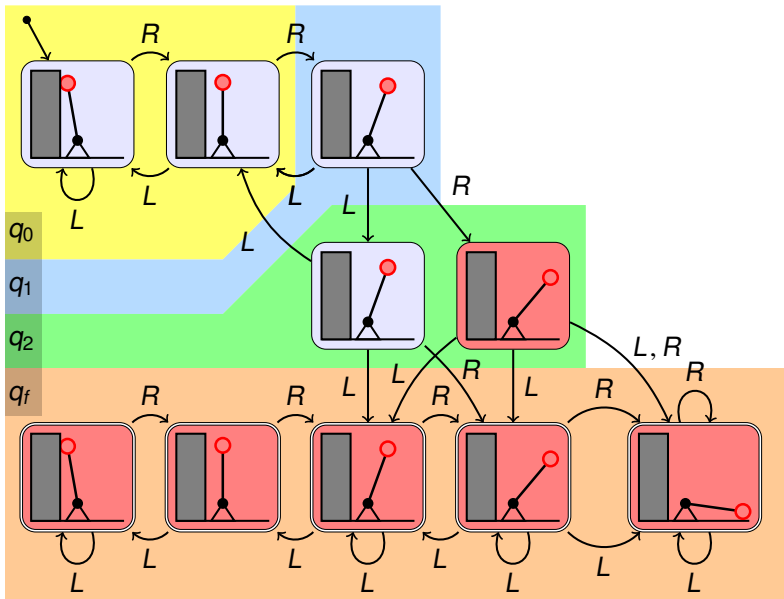
A more complex specification (2)



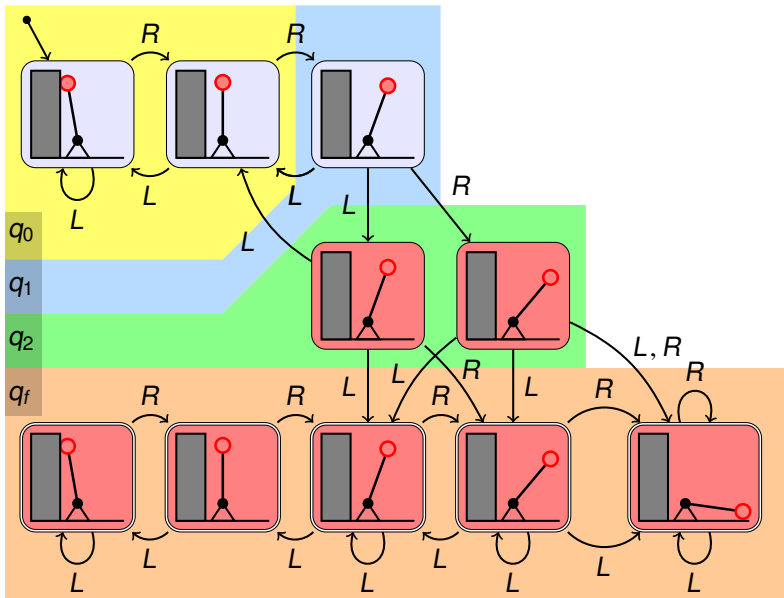
A more complex specification (2)



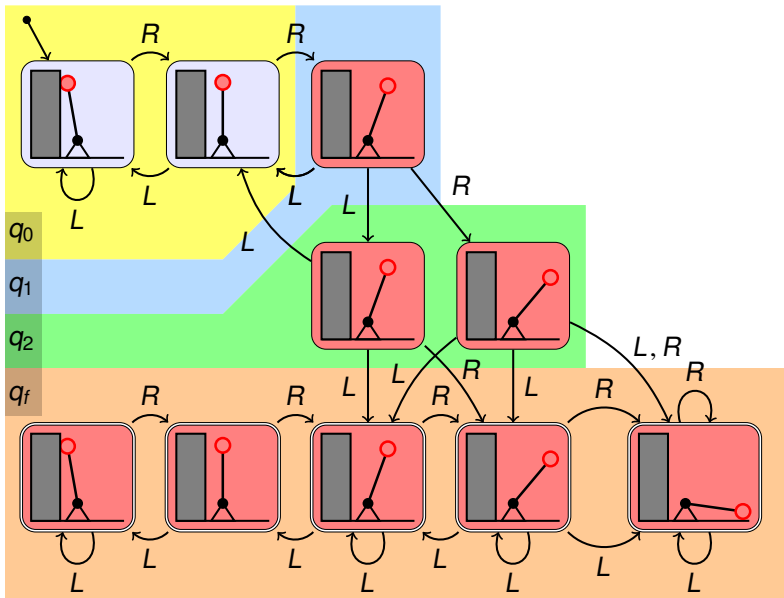
A more complex specification (2)



A more complex specification (2)



A more complex specification (2)



Computing environment descriptions

Necessity of abstractions

Game solving works best with *finite-state* games.

For shielding physical systems, we can thus compute *finite-state* abstractions as the input to the game solving process.

Computing finite-state abstractions

A single-speed vehicle

Workspace limits:

- $x \in [0, 10]$
- $y \in [0, 10]$
- $\omega \in [0, 2\pi]$

System dynamics:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \sin \omega \\ \cos \omega \\ 0 \end{pmatrix} + u \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Computing finite-state abstractions

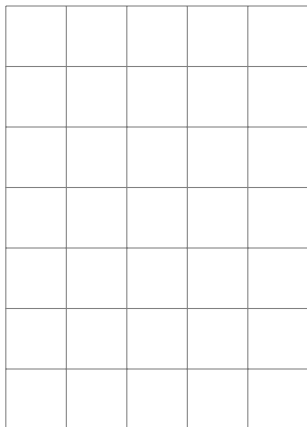
A single-speed vehicle

Workspace limits:

- $x \in [0, 10]$
- $y \in [0, 10]$
- $\omega \in [0, 2\pi]$

System dynamics:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \sin \omega \\ \cos \omega \\ 0 \end{pmatrix} + u \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$



Computing finite-state abstractions

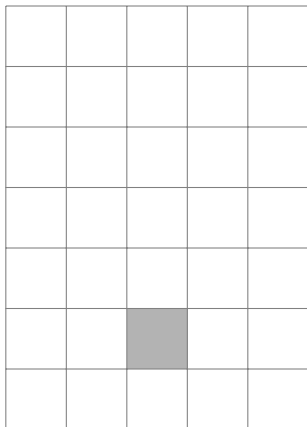
A single-speed vehicle

Workspace limits:

- $x \in [0, 10]$
- $y \in [0, 10]$
- $\omega \in [0, 2\pi]$

System dynamics:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \sin \omega \\ \cos \omega \\ 0 \end{pmatrix} + u \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$



Computing finite-state abstractions

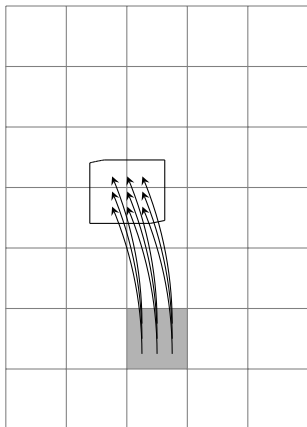
A single-speed vehicle

Workspace limits:

- $x \in [0, 10]$
- $y \in [0, 10]$
- $\omega \in [0, 2\pi]$

System dynamics:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \sin \omega \\ \cos \omega \\ 0 \end{pmatrix} + u \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$



Computing finite-state abstractions

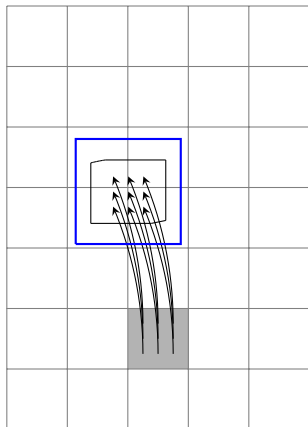
A single-speed vehicle

Workspace limits:

- $x \in [0, 10]$
- $y \in [0, 10]$
- $\omega \in [0, 2\pi]$

System dynamics:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \sin \omega \\ \cos \omega \\ 0 \end{pmatrix} + u \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$



Computing finite-state abstractions

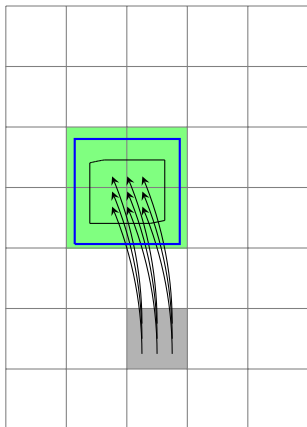
A single-speed vehicle

Workspace limits:

- $x \in [0, 10]$
- $y \in [0, 10]$
- $\omega \in [0, 2\pi]$

System dynamics:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \sin \omega \\ \cos \omega \\ 0 \end{pmatrix} + u \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$



Computing finite-state abstractions

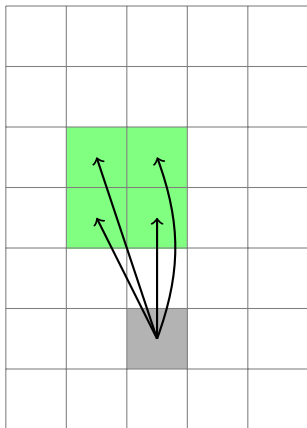
A single-speed vehicle

Workspace limits:

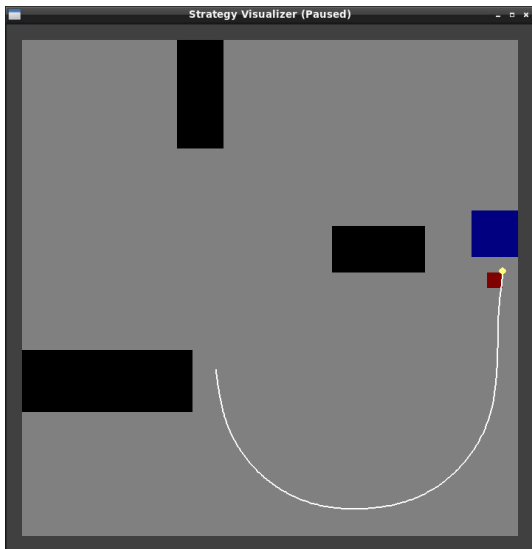
- $x \in [0, 10]$
- $y \in [0, 10]$
- $\omega \in [0, 2\pi]$

System dynamics:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} \sin \omega \\ \cos \omega \\ 0 \end{pmatrix} + u \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$



Abstraction example / demo



Difficulties

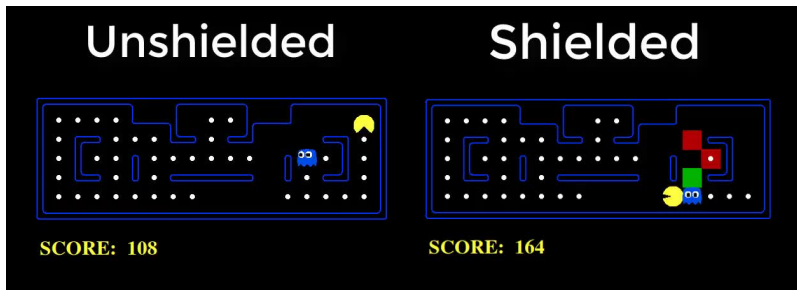
Computation of abstractions

- To account for partially unknown system dynamics, we need to add an error term.
- Computation of abstractions is non-trivial, but tool-support is available:
 - Pessoa - Roy et al., 2011
 - SCOTS - Rungger and Zamani, 2016
 - ROCS - Li and Liu, 2018
- The sizes of abstractions are a **big** problem.

Usage of shields in the field

- Requires perfect *sensing* of the system state (can be partially fixed by using a different type of abstraction)

Comparison shielded/unshielded learning – Pacman



Video and implementation by Bettina Könighofer, TU Graz

Beyond safety

What if the specification is not a *safety* property?

- There exists no *maximally permissive* strategy for the shield in this case.
- We can however compute a *permissive* shield in this case.

Beyond safety

What if the specification is not a *safety* property?

- There exists no *maximally permissive* strategy for the shield in this case.
- We can however compute a *permissive* shield in this case.

Permissive shields: basic idea (liveness)

- Synthesize **a** winning strategy and the set of positions in the game from which the system can win.

Beyond safety

What if the specification is not a *safety* property?

- There exists no *maximally permissive* strategy for the shield in this case.
- We can however compute a *permissive* shield in this case.

Permissive shields: basic idea (liveness)

- Synthesize **a** winning strategy and the set of positions in the game from which the system can win.
- Maintain a *permissiveness counter* (starting with $n > 0$)

Beyond safety

What if the specification is not a *safety* property?

- There exists no *maximally permissive* strategy for the shield in this case.
- We can however compute a *permissive* shield in this case.

Permissive shields: basic idea (liveness)

- Synthesize **a** winning strategy and the set of positions in the game from which the system can win.
- Maintain a *permissiveness counter* (starting with $n > 0$)
- Whenever the winning strategy deviates from the learner's action choice, reduce the counter by one. Change the learner's choice to the choice of the strategy if otherwise a non-winning position could be reached.

Beyond safety

What if the specification is not a *safety* property?

- There exists no *maximally permissive* strategy for the shield in this case.
- We can however compute a *permissive* shield in this case.

Permissive shields: basic idea (liveness)

- Synthesize a winning strategy and the set of positions in the game from which the system can win.
- Maintain a *permissiveness counter* (starting with $n > 0$)
- Whenever the winning strategy deviates from the learner's action choice, reduce the counter by one. Change the learner's choice to the choice of the strategy if otherwise a non-winning position could be reached.
- When the counter is 0, enforce the choices by the winning strategy

Beyond safety

What if the specification is not a *safety* property?

- There exists no *maximally permissive* strategy for the shield in this case.
- We can however compute a *permissive* shield in this case.

Permissive shields: basic idea (liveness)

- Synthesize **a** winning strategy and the set of positions in the game from which the system can win.
- Maintain a *permissiveness counter* (starting with $n > 0$)
- Whenever the winning strategy deviates from the learner's action choice, reduce the counter by one. Change the learner's choice to the choice of the strategy if otherwise a non-winning position could be reached.
- When the counter is 0, enforce the choices by the winning strategy
- When the system reaches its *liveness goal*, reset counter

Note & full LTL

Note

This approach works best if a *non-deterministic* strategy to win the game is used as a starting point.

Note & full LTL

Note

This approach works best if a *non-deterministic* strategy to win the game is used as a starting point.

Full ω -regular specifications

The approach also works for arbitrary ω -regular specifications (when we do not have explicit *liveness goals*).

Starting from a parity game, we maintain a *counter vector* just like in Jurdzinski's small progress measures parity game solving algorithm. The learner's choices are overwritten when one of the counters is 0 or they lead to non-winning positions.

References I

- Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 2669–2678, 2018. URL <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17211>.
- Yinan Li and Jun Liu. ROCS: A robustly complete control synthesis tool for nonlinear dynamical systems. In Maria Prandini and Jyotirmoy V. Deshmukh, editors, *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC 2018, Porto, Portugal, April 11-13, 2018*, pages 130–135. ACM, 2018. doi: 10.1145/3178126.3178153. URL <https://doi.org/10.1145/3178126.3178153>.
- Pritam Roy, Paulo Tabuada, and Rupak Majumdar. Pessoa 2.0: a controller synthesis tool for cyber-physical systems. In Marco Caccamo, Emilio Frazzoli, and Radu Grosu, editors, *Proceedings of the 14th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2011, Chicago, IL, USA, April 12-14, 2011*, pages 315–316. ACM, 2011. doi: 10.1145/1967701.1967748. URL <https://doi.org/10.1145/1967701.1967748>.
- Matthias Rungger and Majid Zamani. SCOTS: A tool for the synthesis of symbolic controllers. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016, Vienna, Austria, April 12-14, 2016*, pages 99–104, 2016. doi: 10.1145/2883817.2883834. URL <https://doi.org/10.1145/2883817.2883834>.