DOI:10.1145/3715958

Evaluating the advantages and potential drawbacks of shielding as a method for safe RL.

BY BETTINA KÖNIGHOFER, RODERICK BLOEM, NILS JANSEN, SEBASTIAN JUNGES, AND STEFAN PRANGER

Shields for Safe Reinforcement Learning

MACHINE LEARNING (ML) has revolutionized computer science and its impact on society. Reinforcement learning (RL)[39] is a prominent ML technique that solves decision-making problems under uncertainty. Figures 1 and 2 illustrate the standard setting of RL at a high level: An *agent* chooses an *action* that is executed in an *environment*. In response to the action, the environment provides a *reward* and an *observation* of the current state. Over a number of *learning episodes*, the agent infers a policy that describes how to behave to meet its *learning objective*, which usually means that the learned policy maximizes the expected reward in the environment. A core concept in RL is the balance between *exploration* and *exploitation*. Exploration involves trying new actions to gather information, while exploitation uses the best-known actions to maximize rewards based on experience.

Deep reinforcement learning (DRL) has elevated RL to complex environments by employing neural network representations of policies.[1] It received significant public attention when AlphaGo solved the board game Go using RL techniques. AlphaGo's successor, AlphaZero, is completely self-taught. RL has also been successfully applied in domains like autonomous driving, intelligent manufacturing, trading, finance, and healthcare.[31]

RL does not provide safety guarantees during learning or deployment.[9] One reason is that the agent needs to *explore* its environment to understand the consequences of its actions. During exploration, *uncertainty* about the environment can lead RL to choose unsafe actions with potentially harmful consequences.[37,39] One approach to increase safety is to penalize unsafe actions during learning. However, even during exploitation, when the agent makes the best decisions based on current knowledge, there is no guarantee that the agent will never choose unsafe actions.

Consequently, over the past few years, *safe reinforcement learning* has attracted significant research ef-

» key insights

- **Reinforcement learning (RL) is a powerful machine learning technique for intelligent sequential decision making. Despite its successes, application in safety-critical systems is impeded by the inherently unsafe actions RL has to take, especially during exploration.**
- **Shielding is an approach toward safe RL that blocks ("shields") actions at runtime based on a model of the world. Shielded RL combines symbolic AI, which provides formal safety guarantees but has lower scalability, with sub-symbolic RL, which offers high scalability but lacks guarantees.**
- **The guarantees provided by the shield are always relative to the world model. The shield uses this model to analyze the consequences of an action, considering all possible future behaviors. However, if the model fails to capture a safety-relevant aspect, the shield cannot avoid safety-critical behavior.**

Figure 1. The agent operates an unmanned aerial vehicle (UAV) tasked with delivering a package without any collisions. Factors such as wind and other aerial vehicles add complexity to this mission.

Figure 2. The reinforcement learning setting.

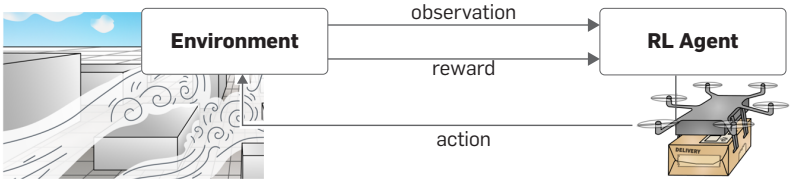
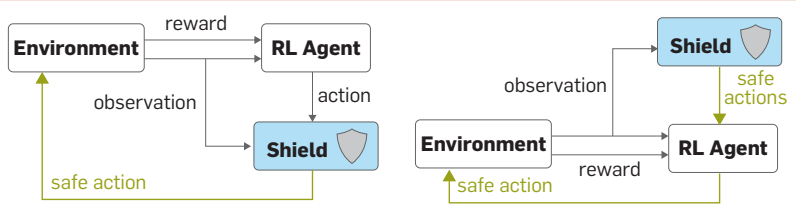


Figure 3. (Left) Post-shielding: The shield prevents unsafe actions from being executed. (Right) Pre-shielding: The shield restricts the choices of the agent.



fort.[16] It comes in three categories: shaping ("engineering") the reward function to encourage the agent to choose safe actions,[25] adding a second cost function ("constraining"),[30] and blocking ("shielding") unsafe actions at runtime.[24] In this article, we focus on the third category—shielding—which provides formal safety guarantees and is thus a particular approach toward verified artificial intelligence (AI).[38]

Integrating shields with reinforcement learning. *Shields* are mechanisms that provide formal safety guarantees by preventing unsafe actions from being executed at runtime. Such a procedure is often referred to as *runtime enforcement*. Integrated into an RL setting, a shield prevents unsafe behavior of the learning agent during both the learning and deployment phases, while restricting the agent as little as possible.

The main approaches to integrating a shield into RL are *pre-shielding* and *post-shielding,* which differ in how they intervene. Figure 3 (left) shows post-shielding: The shield is positioned between the agent and the environment, continuously monitoring the state of the environment and the actions selected by the agent. It evaluates the risk associated with the selected action, classifies the action as safe or unsafe, and replaces any unsafe action with a safe one. Figure 3 (right) illustrates pre-shielding, in which the shield provides a list of all safe actions, allowing the agent to select the best available option. Note that it is formally not possible to compute a shield if no safe action exists. In such cases, the practical solution is to lower the safety requirements, if possible.

Both options have their advantages and disadvantages. Several RL algorithms such as proximal policy optimization[37] and deep Q-learning[29] have *masked* versions[40] that take a list of forbidden actions as extra input, making pre-shielding simple to integrate. In this case, the final decision on which safe action to take remains with the learning algorithm. However, if no masked version of a learning algorithm exists, pre-shielding can be challenging to integrate.

The advantage of a post-shield is that it is generally easy to integrate, as it simply overwrites unsafe actions. However, a naive integration can negatively impact learning performance, for two reasons. First, by overwriting the agent's actions, the shield may slow down the exploration of the environment. To mitigate this, the post-shielding algorithm can often select a "good" alternative for the agent, such as choosing an action similar to the one originally selected. Additionally, some learning algorithms provide a ranked list of actions, allowing the shield to choose the second-best option. Second, a post-shield can disrupt the association between an action and its reward. To address this, the reward should be tied to the action chosen by the shield rather than the agent's original choice.

Shield computation and safety guarantees. The essential property of a shield is its ability to provide provable safety guarantees during learning and deployment. To that end, shields are rigorously computed from a formal *specification* of safety-critical properties and an abstract *model* of the environment. This specifica-

tion distinguishes permitted (safe) and forbidden (unsafe) behaviors and is independent of the agent's learning objective. Thus, a shield maintains a clear separation between an agent's safety and performance. Many formal approaches exist to derive shields, leveraging techniques from reactive synthesis,[5] supervisory control,[35] theorem proving,[15] or model checking.[3,11]

The guarantees that a shield provides may be *absolute* or *probabilistic* (see later sections). When using *deterministic models*, the resulting shields treat safety as a qualitative, absolute measure. By taking only actions classified as safe, the specified safety-critical properties are guaranteed to never be violated, regardless of what may occur in the environment in the future. If this condition is not met, an action is deemed unsafe.

In real-world scenarios, absolute safety guarantees may be unrealistic. For example, an unmanned aerial vehicle (UAV) may not be able to rule out the possibility of colliding with other aerial vehicles when they behave highly irregularly. Thus, a low risk of a collision may be acceptable. For these situations, probabilistic models are used to compute shields that consider safety as a *quantitative measure*. An action is deemed unsafe if it induces an *unacceptably high risk* of compromising safety. To classify an action as safe or unsafe, a shield computes the probability of an action violating safety. Actions are classified as safe if this probability is below some threshold. By adjusting this threshold, shields can be generated that are either more liberal or more restrictive toward the agent.

Promise and limitations of shielding as an approach to combine symbolic and sub-symbolic AI. Shielded RL combines model-based methods with DRL, providing provable safety guarantees and high performance. Shielding thus combines the benefits of sub-symbolic and symbolic AI. Neural networks can process large amounts of data to learn excellent approximations of ill-understood functions inductively and offer performance that cannot be matched by other methods. On the other hand, symbolic AI offers deductive, data-independent approaches that are less performant but provide guarantees. However, shielding suffers inherently from the limitations of model-based verification. To provide safety guarantees, one has to employ techniques such as model checking. Though state-of-the-art shielding approaches are able to handle environments with billions of states,[19] real-world applications may demand far larger scalability.

Core questions of shielded reinforcement learning. We will discuss the following core questions of shielded RL:

- What types of *safety guarantees* can be provided by a shield, and under which *assumptions?*
- How can shields be *computed*?
- How can shields be *integrated* in RL?
- What are the *challenges* in shielded RL?

The purpose of this article is to discuss these questions and to shed light on the current state of the art of shielded learning.

Modeling

Shielding requires a formal model of the environment. Here, we assume that the environment consists of a finite number of states and that time is discrete. We model an environment as a Markov decision process (MDP),[8] the standard model used in RL. MDPs have a probabilistic transition function that captures uncertainties in the environment, including the consequences of the agent's actions and the behavior of other agents acting within the same environment. We discuss relaxations of these assumptions in a later section.

Example 1. Figure 4 shows a discrete representation of the environment of a UAV, depicted as a two-dimensional grid. Figure 5 shows part of the MDP for this example, where the states correspond to the cells in the grid and the actions are the labels of the transitions that connect the states. The specification for this example is simple: Do not crash into a building. The red states, labeled 5A and 5B, indicate collisions of the UAV and a building, and thus violations of the safety specification.

The agent's actions label the edges: north (N), east (E), south (S), and west (W). The outcome of an action is uncertain due to the southwest wind, captured by the probabilistic transition function. For example, if the UAV chooses Action N in Cell 4A, it will move to Cell 4B with probability p and to Cell 5B with probability $1 - p$.

Deterministic models. MDP models represent the environment probabilistically. Using deterministic models, one can prevent unsafe behavior regardless of the behavior of the environment by adopting a worst-case view of its behavior. One reason to use such a deterministic model is that not enough data may be available to derive probabilities. Adversarial interaction between the environment

Figure 4. A snippet of the discrete model for the UAV example. The gray-shaded squares represent buildings. The UAV can move from cell to cell. The blue arrows indicate one possible path to the target location.

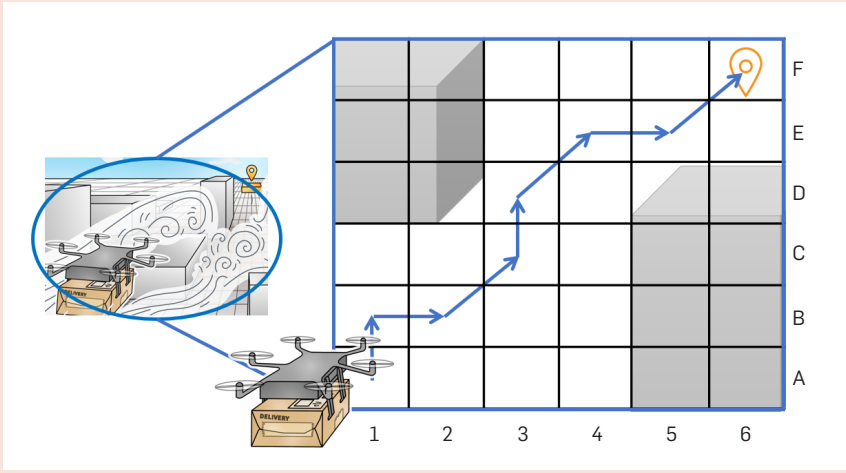


Figure 5. Markov decision process of the environment depicted in Figure 4.

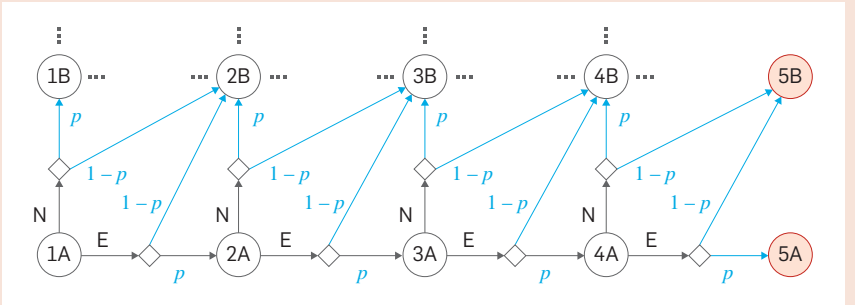
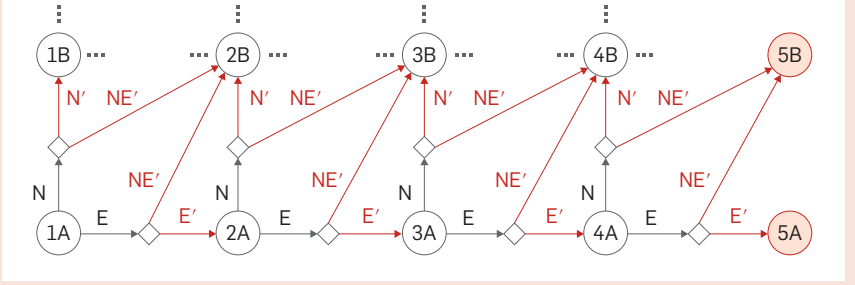


Figure 6. Two-player game representing the adversarial view of the MDP in Figure 5.



and the agent can be modeled as a *two-player game*, where the next state is determined by the actions chosen by both the agent player and the environment player. By removing the probabilities from the edges and replacing them with the environment's choices, we can transform an MDP into a two-player game.

Example 2. Figure 6 depicts the two-player game graph derived from the MDP in Figure 5. The agent-player first selects an action, followed by the environment choosing its action. These decisions determine the subsequent state. For instance, when the agent selects Action N from Cell 3A, the environment can either move the UAV to Cell 3B by choosing Action N' or to Cell 4B by selecting Action NE'.

States and features. The rigor required to compute shields affects the scalability of the approach. Fortunately, shields can often be computed using models with a reduced feature space. By disregarding any features irrelevant to safety, the original model can be pruned to a much smaller model.[19] In the UAV example, such features may include temperature or the designated goal position. Retaining the safety-relevant dynamics of the environment allows the use of model-based techniques to compute the shield, even when the performance-optimal policy cannot be computed with these model-based techniques.

Shielding with Absolute Safety Guarantees

In this section, we discuss how to construct shields that consider safety as an absolute measure using deterministic two-player games in which the environment is treated adversarially (see Figure 6). We will classify actions without any risk as safe, and all other actions as unsafe. This worldview, which does not distinguish between improbable and likely events, may be overly conservative. However, it offers absolute safety guarantees, which can be essential for certain safety-critical applications. It is important to note that future consequences are considered when classifying an action as safe or unsafe: An action taken by an agent is considered safe only if the agent can continue to stay in safe states, regardless of the environment's future actions. Consequently, an action may be deemed unsafe even if a safety-critical state is entered much later in the future.

Safety guarantees. There is a rich body of literature on how to express properties of systems, for instance, using logics such as linear temporal logic.[8] Shielding is typically used to enforce *safety* properties, which state that something bad never happens. In the UAV setting, a safety property could state, "The UAV never collides with another moving or static obstacle." Safety properties can also encompass temporal aspects, such as "The UAV must reach the target within 30 minutes."

For this article, it suffices to note that such properties can be converted to automata and combined with the model via standard product constructions.[8] Without loss of generality, we can assume that the resulting model has at least one *unsafe state* that indicates a violation of the safety guarantee. A shield with absolute safety guarantees ensures that such an unsafe state is never visited, thereby ensuring that no safety property is violated.

Shield computation. The specification of the safety properties that the RL agent must fulfill can easily be converted into a specification for the shield: The shield must 1) ensure that the agent satisfies the safety specification, and 2) allow the agent to select any action that is safe for any future actions of the environment. These requirements can be met by solving a safety game on the model of the environment. Solving a safety game involves computing a *non-deterministic strategy* that defines, for each state, all actions the agent can choose to ensure that an unsafe state is never reached. The methods for solving safety games are well established.[8] We call a state *dangerous* if there exist actions that the environment may choose in the future such that visiting an unsafe state becomes unavoidable, regardless of the actions the agent takes. Vice versa, a state is *safe* if it is not dangerous: The agent can always avoid an unsafe state, no matter what the environment does. In game theory, the set of safe states is referred to as the *winning region*. This set is inductive, meaning that from any safe state, the agent can select an action that leads to another safe state, regardless of the choice taken by the

environment. A shield ensures that the agent remains within the winning region by permitting only those actions that maintain this condition.

The set of dangerous states can be computed recursively by a backward traversal of the graph in time linear in its size. The algorithm keeps a set of dangerous states, which is initialized to the set of unsafe states. At each step, the algorithm adds all states from which, regardless of the action of the agent, there is an environment action that leads to a dangerous state. From such states, the environment can enforce that a dangerous state will be visited. The algorithm continues until a fixpoint is reached.

Example 3. Figure 7 illustrates the safe (white), the dangerous (shaded red), and the unsafe (red) states for the UAV example. The boxes surrounding the dangerous states illustrate the intermediate steps of the algorithm for computing the winning region. Initially, the dangerous states in Box 1 are identified, representing states from which the environment can force a transition to an unsafe state in a single step. For instance (referring back to Figure 6), State 4A is dangerous because both for Action N and for Action E, the environment can choose to visit to State 5B, which is an unsafe state. Similarly, the dangerous states in Box 2 and Box 3 represent states from which the environment can force a transition to an unsafe state in two and three steps, respectively.

Shielding with Probabilistic Guarantees

So far, we have used a simple deterministic model of the environment. This ensures absolute safety when successful, but it fails when it is not possible to guarantee safety for any environment behavior.

Sensors and actuators are examples of systems that fail infrequently, but whose failure may lead to catastrophic results. Not modeling that they can be faulty is overly optimistic, while the worst-case assumption that sensors always fail is too pessimistic to yield useful results. A realistic middle ground is to adopt a probabilistic worldview, assuming that sensors and actuators fail with some probability while ensuring that the probability of reaching bad states remains low.

Finite horizons. In the previous section, we required the agent to remain in the winning region forever. This requirement is not applicable to all probabilistic systems. Since events that happen with a very low probability at any time will happen eventually with probability one, given enough time, restricting ourselves to an infinite horizon would make the definition (almost) as conservative as shields with absolute safety guarantees. Consequently, we often consider a finite horizon of h steps. The horizon can be chosen to be the mission time, the expected battery life, or the time that a safety procedure requires.

Permissive policies versus probabilistic shields. Lifting shielding toward probabilistic guarantees comes with some design choices, which we illustrate with the following example.

Example 4. Consider the MDP shown in Figure 8. As a safety specification, we want to ensure that the probability of reaching a bad (red) state does not exceed 0.1 within, say, four steps. This specification is satisfied by any policy that takes action B at most once.

The design choice can be exemplified by the following Gedankenexperiment: When an agent is in state $S1$, should the shield allow taking action B? Option 1 is to allow taking action B, as afterward always taking action A will ensure that the specification is satisfied. Option 2 is to only allow taking action B if the agent did not

Figure 7. Categorization of the state space in unsafe (red), dangerous (shaded), and safe (white) states. The numbers show the steps unsafe states. The blue path shows the states traversed when the UAV selects the Action N starting from Cell 2A, while the environment diverts this action to the northeast.

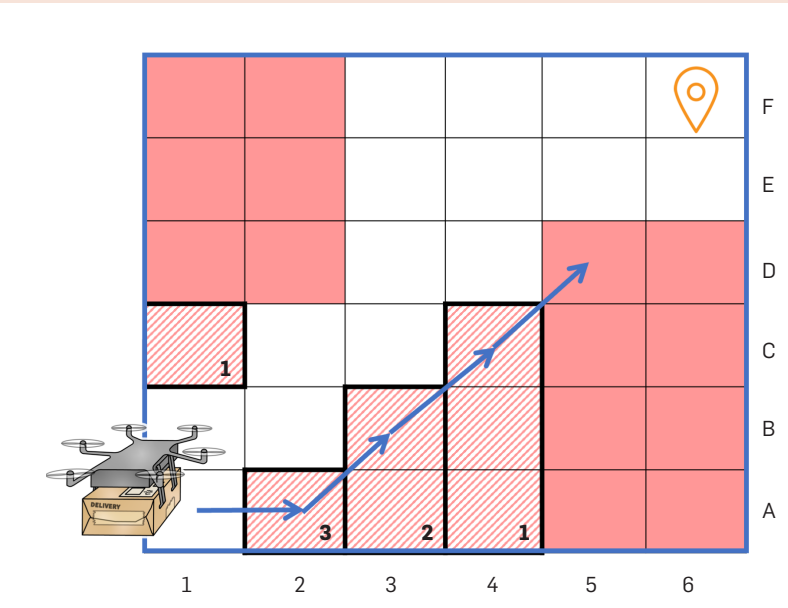
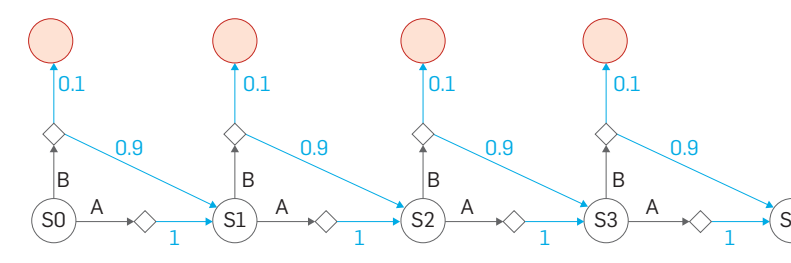


Figure 8. The example MDP used for the Gedankenexperiment.



choose action B in state $S0$. Option 3 is to not allow taking action B.

Option 2 is the perspective that permissive policies[11,22] take. Permissive schedulers tend to be conservative and hard to compute. They prevent actions based on previously taken risks, even when those risks were successfully averted. We focus on Option 1, which aligns with the idea discussed in the previous section: A shield allows any action for which it can keep ensuring that it is possible to satisfy a given specification. The clear downside of following Option 1 is best observed in the previous example: The shield will, in every step, allow taking action B, and thus, a shielded policy may take B in every step and thereby not meet the given specification. Finally, Option 3 is feasible in the given MDP but generally leads to situations in which every action is prevented, even when there are safe policies.

Probabilistic shields. In line with Option 1, we call an action safe if, after taking that action, the minimal probability of reaching an unsafe state within h steps is at most ε. Otherwise, an action is called unsafe. The winning region is the set of all states for which at least one safe action exists.

Shield computation. To construct the shield, we must compute the probability that something bad happens within h steps, assuming that the agent acts optimally in terms of safety in the future. Thus, we are interested in the minimal reachability probabilities across all possible behaviors of the shielded agent. If the exact computation of these probabilities is not feasible, we must obtain *upper bounds*.

Computing (upper bounds on) the optimal reachability probability in an MDP is a standard problem, often also referred to as the *stochastic shortest path problem*.[3] This task requires solving a dynamic program. The minimal reachability probability when taking action a in state s is the expected minimal reachability probability over the successors. The set of dangerous states is then determined by computing the minimal probability of reaching a bad state within h steps.

The resulting dynamic program can be solved in time linear in the number of states, actions, and the horizon. As mentioned above, similar queries have also been defined for more intricate temporal properties and can be solved via adaptions of value iteration, policy iteration, or linear programming.

Equations.

$$x_{5A,h} = x_{5B,h} = x_{5A,h-1} = x_{5B,h-1} = \ldots = 1,$$

$$x_{4A,h} = \min\{\underbrace{p \cdot x_{5A,h-1} + (1-p)x_{5B,h-1}}_{\text{E}}, \underbrace{p \cdot x_{4B,h-1} + (1-p)x_{5B,h-1}}_{\text{N}}, \ldots\},$$

$$x_{3A,h} = \min\{\underbrace{p \cdot x_{4A,h-1} + (1-p)x_{4B,h-1}}_{\text{E}}, \underbrace{p \cdot x_{3B,h-1} + (1-p)x_{4B,h-1}}_{\text{N}}, \ldots\}.$$

Example 5. The equations above give a fragment of a dynamic program to define the minimal probability to reach bad states in Figure 5, where $x_{ij,h}$ describes the minimum probability to reach a bad state from Cell $\langle i,j \rangle$ within h steps.

Given the winning region and the set of safe actions, the shield can be directly deployed. In case a state outside of the winning region is entered during runtime, a fallback strategy needs to be defined, such as selecting a predefined action (for example, braking or landing), or allowing only the safest possible action.

Benefits and Drawbacks

In this section, we discuss the main benefits and potential drawbacks of shielded RL. Compared to standard RL without any mechanism to ensure safety, shielding provides two main advantages:

▸ *Safety assurance.* When using shields with absolute safety guarantees, a shielded learning agent will never violate the critical specification, whether during training or deployment.

▸ *Improved sample efficiency.* Shielding often reduces the number of samples needed to learn an optimal policy by preventing the agent from taking unsafe actions, especially when rewards are sparse.[7] Basically, the agent avoids exploring unsafe parts of the state space, allowing it to gather more experiences in the safe states.

Shielding features the following additional characteristics:

▸ *Shields are permissive.* Shields do not hinder the execution of safe actions. Safe actions will not be masked during pre-shielding, and will not be overwritten during post-shielding. As long as the agent does not attempt to choose unsafe actions, it acts as if there is no shield.

▸ *Convergence.* RL algorithms that converge on MDPs also converge in the presence of a shield. Essentially, the joint behavior of the shield and the environment MDP can again be modeled as an MDP, as long as the corrections (post-shielding) and restrictions (pre-shielding) from the shield are fixed and do not change over time.[12]

▸ *Dynamic adaptation.* Shields with probabilistic guarantees may adapt their probability threshold dynamically based on the agent's performance and the state of the environment. For example, if the agent learns new behaviors that approach unsafe regions, the shield can tighten its safety constraints. Conversely, as the agent becomes more proficient, the shield can gradually relax the constraints to allow for more exploration.[7]

Shielding is a flexible approach and can be combined with other approaches to enhance safety or efficiency while learning:

▸ *Guided learning via reward shaping.* In addition to preventing unsafe actions, the shield can modify the reward function to incorporate safety considerations. It can add negative rewards to risky actions or provide additional rewards for actions that maintain safety with high probability. This encourages the agent to learn policies that are both effective and safe.

▸ *Incorporating prior knowledge.*

Shields can incorporate expert knowledge about the task to be learned. Since any objective incorporated into the shield is enforced during runtime, the agent does not necessarily need to learn these properties. Consequently, the agent's reward function can be simpler, potentially accelerating the learning process.[2]

The following considerations should be taken into account when performing shielding:

▸ *Shielding depends on a model.* As with any formal method, the problem must be well understood and articulated. While a shield ensures that a state marked "collision" is never reached, it cannot ensure, for instance, that an object is correctly classified by a vision system. Also, the shield's correctness depends on the model being correct. If the model incorrectly states that a state is not problematic, the shield will not protect against reaching it.

▸ *Possible reduced learning performance.* Although shields often speed up learning, they may also do the opposite, as exploring unsafe states can help the agent to generalize and understand its task. A restrictive shield may hinder thorough exploration of the environment, which is crucial for discovering optimal policies. In particular, changes made by post-shielding may decrease learning performance.

▸ *Computational effort.* An MDP describing a realistic environment may have a very large number of states and transitions, even when it includes only safety-relevant features. Consequently, it may be challenging to efficiently compute a shield from the MDP. These computational aspects remain an ongoing area of research.

Figure 9. A grid world containing two UAVs. The learning agent controls UAV-1 to deliver a package while avoiding collisions with UAV-2.

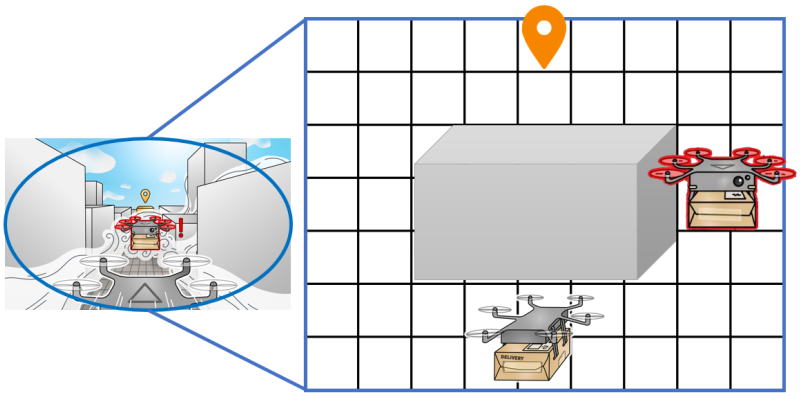
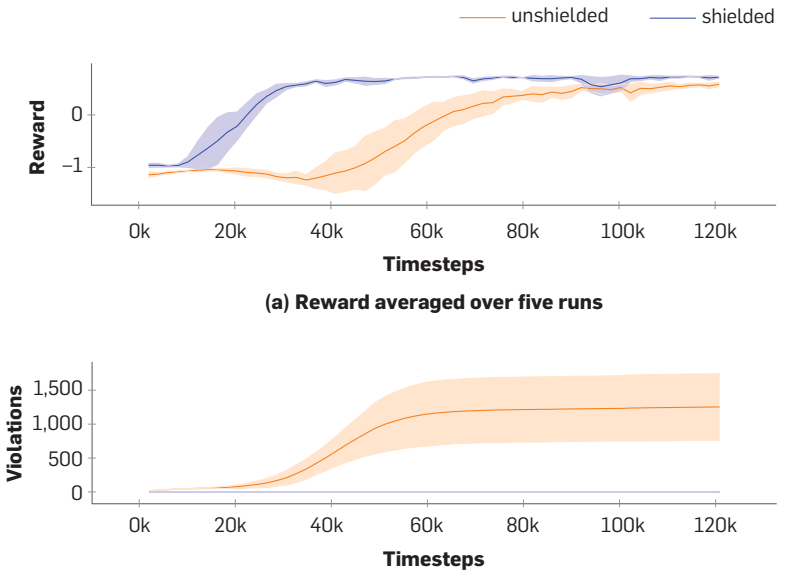


Figure 10. Results for reinforcement learning in the grid world shown in Figure 9.



(a) Reward averaged over five runs

(b) Accumulated safety violations averaged over five runs

Shielding Demonstrations

In this section, we demonstrate the effects of shielding using our running example of a UAV delivering packages. In the first example, we model the environment as adversarial and deploy a pre-shield that provides absolute safety guarantees. In contrast, in the second example we use pre-shields with probabilistic safety guarantees and discuss the results.[a]

Collision avoidance with absolute safety guarantees. Figure 9 illustrates the grid world for the first demonstration. The agent's task is to control UAV-1 to deliver packages to a designated target position. The safety objective is to avoid collisions with UAV-2 (framed in red) and buildings.

We model UAV-2 adversarially, assuming that it actively tries to collide with UAV-1. This way, the shield's safety guarantees hold regardless of the behavior of UAV-2 (see earlier section, "Shielding with Absolute Safety Guarantees"). We deploy shielded and unshielded RL to search for an optimal and safe policy to deliver the package. During training, the agent is rewarded when reaching the goal and receives a negative reward for violating safety. Additionally, it receives a small negative reward for each step taken.

a Experimental setup: The agents were trained using the maskable proximal policy optimization implementation provided by Stable-Baselines3[34] with the default hyperparameters. The shields were computed using the shield synthesis tool TEMPEST.[33] The details of the experiments can be found in the supplemental material in the ACM Digital Library: https://dl.acm.org/doi/10.1145/3715958

Figure 11. A grid world containing a UAV, controlled by the agent, which must deliver a package. The wind's direction per cell is indicated by varying shading. Paths are illustrated that were learned when using no shield (orange path) or shields with different probability thresholds (blue paths).

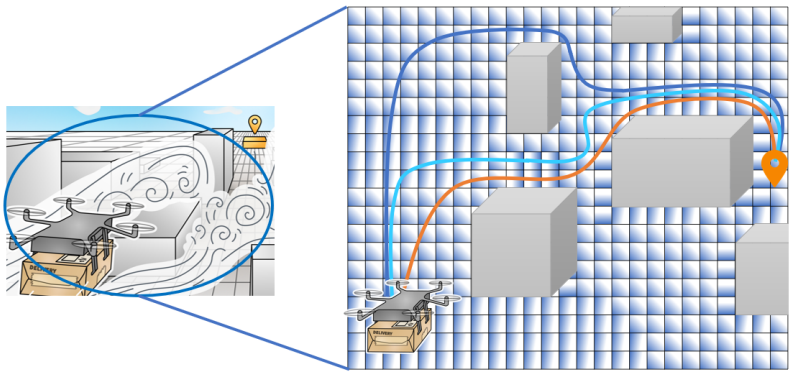
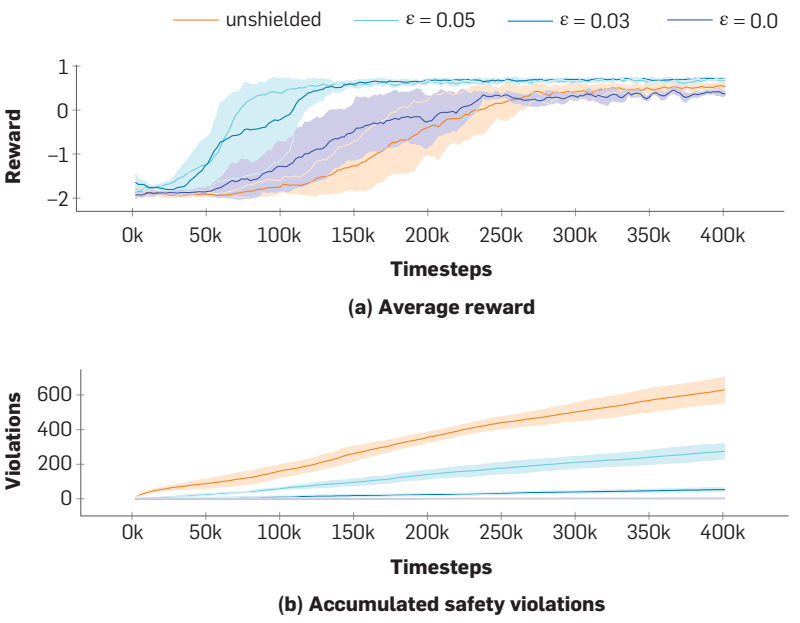


Figure 12. Results for reinforcement learning in the grid world shown in Figure 11.



Results. The learning results are illustrated in Figure 10. The orange curves represent the results for unshielded learning, while the blue curves show the results for shielded learning with absolute safety guarantees. Figure 10a reports the reward, and Figure 10b shows the accumulated safety violations, both averaged over five runs. In this example, shielding clearly enhances learning performance. The challenging part of this learning task is avoiding collisions with UAV-2. Since the shield handles this aspect, the shielded agent quickly learns to complete the package delivery efficiently. While the unshielded agent eventually learns to avoid collisions and to achieve a similar expected reward, the shielded agent never violates safety and converges in a fraction of the time.

Collision avoidance with probabilistic safety guarantees. The grid world for the second demonstration is depicted in Figure 11. In this example, the UAV has to deliver a package without crashing into a building, while wind affects its movement. The wind direction is modeled individually for each cell, with a 5% probability that the wind displaces the UAV in its direction. We compute shields that provide probabilistic safety guarantees (see earlier section, "Shielding with Probablistic Guarantees"), using a finite horizon of 20 time steps and the probability thresholds $\varepsilon \in \{0.00, 0.03, 0.05\}$, where ε limits the risk the agent is allowed to take. As before, we compare unshielded and shielded RL for finding a policy for delivering packages, using a similar reward function.

Results. The orange curves in Figure 12 depict the reward and the number of safety violations observed during training without a shield. The blue curves depict the same metrics when shields with different probability thresholds are applied. As expected, the lower ε is, the fewer safety violations occur. Interestingly, when using either shield that allows the agent to take some risk, the agent learns faster and achieves a higher reward compared to using no shield or a shield that completely forbids taking risks.

Figure 11 illustrates the learned paths for the different learning settings. The unshielded agent learns to take the shortest and narrowest path (orange), which involves a relatively high risk of colliding with a building. In contrast, the shielded agents take longer but safer routes. The shields with $\varepsilon = 0.03$ and $\varepsilon = 0.05$ allow the path between the buildings (light blue), enforcing only a slightly larger distance from the buildings. Thus, most of the agent's flexibility in choosing the path to the target is maintained, while its search space is reduced by prohibiting too risky routes. The longest path (dark blue), which circumnavigates the building, is taken using the shield with $\varepsilon = 0.00$, resulting in a lower final reward.

Further applications. Deep reinforcement learning began gaining significant attention around 2013, with a key breakthrough being the development of DeepMind's Deep Q-Network. The first works on shielded RL emerged in 2018. Much like the early development of RL, shielding

was initially applied in computer games.[17] Only recently has it been applied to prominent application domains such as robotics[20] and autonomous driving,[18] where ensuring safety is critical. These developments represent the first steps in applying shielded RL to real-world challenges.

State of the Art and Challenges

Shields for RL were first introduced in Könighofer et al.,[24] but were originally introduced in the setting of reactive systems.[5] Shielding can be interpreted in the general framework of *supervisory control theory*,[35] which discusses how a high-level controller (a shield) may disable certain actions of lower-level controllers (an RL agent). It should be noted that similar ideas have been introduced independently by others. Notably, Fulton and Platzer introduced the related concepts of *justified speculative control*[15] in continuous settings using a theorem prover as a verification engine.

In this section, we will describe extensions, covering partial observability, continuous systems, multi-agent systems, and properties that go beyond mere safety. We will not give an overview of other methods to guarantee safety in RL.[16] Similarly, for a comprehensive survey on formal runtime enforcement techniques, we refer readers to Falcone et al.[13]

Partially observable environments. So far, we have considered *fully observable* environments, where the shield (and the agent) knows the precise state. This assumption may be unrealistic. A standard approach is to consider *partially observable* environments, where only a part of the state can be observed. Shields are then computed from partially observable Markov decision processes (POMDPs). A challenge for these models is that computing policies is hard or even undecidable.[26] Fortunately, conservative shields can potentially be computed by under-approximating the safe actions. Such shields can be constructed for non-trivial examples, guaranteeing safety and potentially improving the empirical learning rate.[7]

Continuous state and action spaces. Shielding can be extended to systems with continuous state and action spaces. Using continuous systems increases the theoretical complexity and tractability of computing shields. Yet, several papers address this extended problem. For example, Fisac et al.[14] provide a Bayesian method to provide safety guarantees for robot systems. Kochdumper et al.[23] propose a post-shielding approach for cyber-physical systems with nonlinear dynamics using zonotopes, and David et al.[10] proposed a pre-shielding approach for systems with continuous time. Brorholt et al.[6] present an approach to shielding based on discretization, while Rodriguez et al.[36] present a method based on properties written in LTL modulo theories, which allows reasoning over non-Boolean domains. Simplex-based approaches for guaranteed safety in a continuous setting have also been considered.[21,27,32]

To classify an action as safe or unsafe, a shield computes the probability of an action violating safety. Actions are classified as safe if this probability is below some threshold. By adjusting this threshold, shields can be generated that are either more liberal or more restrictive toward the agent.

Multi-agent systems. Shielding multi-agent systems is difficult, as most verification problems are undecidable in a distributed setting. Nevertheless, shields for multi-agent systems have been introduced using runtime verification results, in both centralized and decentralized settings.[4,12,28]

Beyond safety. Safety properties require that something bad never happens; that is, any violation of a safety property consists of a finite "bad" trace. This class is relatively broad, encompassing invariants, such as "a collision never happens," as well as *bounded* properties, such as "every request must be answered within five time units." However, various *unbounded* properties are not safety properties. For example, consider that a UAV should eventually reach its destination. Properties for which no finite bad trace can, by itself, demonstrate that the property is violated are called *liveness properties*.[8] Under certain assumptions, a liveness property can be ensured by a shield that merely prevents an agent from entering a state in which satisfying the property becomes impossible. For instance, a UAV should not drain its battery before reaching its destination, as doing so would prevent the UAV from ever getting there. One such assumption could be that the agent follows an ε-greedy strategy,[7] ensuring that it will eventually explore

any unshielded action. In the absence of such assumptions about the agent's behavior, shielding becomes more challenging, as the shield must enforce specific actions to ensure the satisfaction of the liveness property. The main question is *when* the shield should interfere. Specifically, at any point in time, if the shield intervenes to satisfy the liveness property, this intervention may be premature, as the agent might have satisfied the property on its own at some later point in time. One possible approach to decide when to interfere employs quantitative methods, assigning costs to violations of a property and to the shield's interference.[2]

Conclusion

Truly autonomous and self-learning systems often rely on reinforcement learning, yet the safety of RL systems remains a significant concern. In this article, we discussed shielding, a principled and effective approach to ensuring the safety of RL, even during training. We reviewed and explained how shielding embeds formal guarantees into the behavior of a learning agent, using symbolic reasoning about a given world model. Nevertheless, significant challenges remain: How can we obtain a reliable world model? How can we maintain guarantees in an inherently uncertain and changing world? How can we provide meaningful probabilistic guarantees? And can we explain shields in ways that increase trust in human-AI interactions? Our work explores these challenges, highlighting numerous extensions in the literature, including applications to continuous systems and multi-agent systems.

Acknowledgments

We thank Chao Wang and Robert Koenighofer for the inspiration and the work on the initial formulation of shields; Ruediger Ehlers for his great contributions to shields for reinforcement learning; Ufuk Topcu for many interesting ideas; our many other collaborators and students for ideas, discussions, ideas, and implementations; and Dejan Nickovic, Benedikt Maderbacher, and the anonymous reviewers for many comments that improved the article. Illustrations by Theresa Dachauer. C

References

1. Arulkumaran, K. et al. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine 34*, 6 (2017), 26–38.
2. Avni, G. et al. Run-time optimization for learned controllers through quantitative games. In *Computer Aided Verification (CAV), vol. 11561 of Lecture Notes in Computer Science*. Springer (2019), 630–649.
3. Baier, C. and Katoen, J. *Principles of Model Checking*. MIT Press (2008).
4. Bharadwaj, S. et al. Synthesis of minimum-cost shields for multi-agent systems. In *American Control Conf.* IEEE (2019), 1048–1055.
5. Bloem, R. et al. Shield synthesis: Runtime enforcement for reactive systems. In *Intern. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), vol. 9035 of Lecture Notes in Computer Science*. Springer (2015), 533–548.
6. Brorholt, A.H. et al. Shielded reinforcement learning for hybrid systems. In *Symposiom on Bridging the Gap Between AI and Reality (AISoLA), vol. 14380 of Lecture Notes in Computer Science*. Springer (2023), 33–54.
7. Carr, S. Safe reinforcement learning via shielding under partial observability. In *Conf. on Artificial Intelligence (AAAI)* AAAI Press, (2023), 14748–14756.
8. Clarke, E.M. et al., (Eds.). *Handbook of Model Checking*. Springer (2018).
9. Dalrymple, D. et al. Towards guaranteed safe AI: A framework for ensuring robust and reliable AI systems. CoRR abs/2405.06624 (2024).
10. David, A. et al. Uppaal stratego. In *Intern. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), vol. 9035 of Lecture Notes in Computer Science*, Springer, (2015), 206–211.
11. Dräger, K. et al. Permissive controller synthesis for probabilistic systems. *Logical Methods in Computer Science 11*, 2 (2015).
12. Elsayed-Aly, I. et al. Safe multi-agent reinforcement learning via shielding. In *Intern. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS)*. ACM (2021), 483–491.
13. Falcone, Y., Fernandez, J., and Mounier, L. What can you verify and enforce at runtime? *Intern J on Software Tools for Technology Transfer 14*, 3 (2012), 349–382.
14. Fisac, J.F. et al. A general safety framework for learning-based control in uncertain robotic systems. *IEEE Transactions on Automatic Control 64*, 7 (2019), 2737–2752.
15. Fulton, N. and Platzer, A. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*. AAAI Press (2018).
16. García, J. and Fernández, F. A comprehensive survey on safe reinforcement learning. *J of Machine Learning Research 16* (2015), 1437–1480.
17. Giacobbe, M. et al. Shielding atari games with bounded prescience. In *AAMAS '21*. International Foundation for Autonomous Agents and Multiagent Systems (May 2021), 1507–1509.
18. Hu, H., Nakamura, K., and Fisac, J. F. Sharp: Shielding-aware robust planning for safe and efficient human-robot interaction. *IEEE Robotics and Automation Letters 7*, 2 (2022), 5591–5598.
19. Jansen, N. et al. Safe reinforcement learning using probabilistic shields (invited paper). In *Intern. Conf. on Concurrency Theory (CONCUR), vol. 171 of LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020), 3:1–3:16.
20. Ji, G. et al. Towards safe control of continuum manipulator using shielded multiagent reinforcement learning. *IEEE Robotics and Automation Letters 6*, 4 (2021), 7461–7468.
21. Johnson, T.T. et al. Real-time reachability for verified simplex design. *ACM Transactions on Embedded Computing Systems (TECS) 15*, 2 (2016), 1–27.
22. Junges, S. et al. Safety-constrained reinforcement learning for mdps. In *Intern. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), vol. 9636 of Lecture Notes in Computer Science*. Springer (2016), 130–146.
23. Kochdumper, N. et al. Provably safe reinforcement learning via action projection using reachability analysis and polynomial zonotopes. CoRR abs/2210.10691 (2022).
24. Könighofer, B. et al. Shield synthesis. *Formal Methods in System Design 51*, 2 (2017), 332–361.
25. Laud, A. and DeJong, G. The influence of reward on the speed of reinforcement learning: An analysis of shaping. In *Intern. Conf. on Machine Learning, (ICML)* AAAI Press, (2003), 440–447.
26. Madani, O., Hanks, S., and Condon, A. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence 147*, 1-2 (2003), 5–34.
27. Maderbacher, B. et al. Provable correct and adaptive simplex architecture for bounded-liveness properties. In *Symp. on Model Checking Software (SPIN), vol. 13872 of Lecture Notes in Computer Science*, Springer, (2023), 141–160.
28. Melcer, D., Amato, C., and Tripakis, S. Shield decentralization for safe multi-agent reinforcement learning. In *Annual Conf. on Neural Information Processing Systems* (2022).
29. Mnih, V. et al. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (2015), 529–533.
30. Moldovan, T. M. and Abbeel, P. Safe exploration in markov decision processes. In *Intern. Conf. on Machine Learning* (2012).
31. Mousavi, S.S., Schukat, M., and Howley, E. Deep reinforcement learning: An overview. In *Proceedings of SAI Intelligent Systems Conf. 2016: Volume 2*. Springer (2018), 426–440.
32. Phan, D.T. et al. Neural simplex architecture. In *NASA Formal Methods - 12th Intern. Symp., NFM 2020, Proceedings vol. 12229 of Lecture Notes in Computer Science*. R. Lee, S. Jha, and A. Mavridou, (Eds.). Springer (2020), 97–114.
33. Pranger, S. et al. TEMPEST - synthesis tool for reactive systems and shields in probabilistic environments. In *Automated Technology for Verification and Analysis 2021, vol. 12971 of Lecture Notes in Computer Science*. Springer (2021), 222–228.
34. Raffin, A. et al. Stable-baselines3: Reliable reinforcement learning implementations. *J. of Machine Learning Research 22* (2021), 268:1–268:8.
35. Ramadge, P.J. and Wonham, W.M. Supervisory control of a class of discrete event processes. *SIAM J. on Control and Optimization 25*, 1 (1987), 206–230.
36. Rodriguez, A. et al. Shield synthesis for LTL modulo theories. CoRR abs/2406.04184 (2024).
37. Schulman, J. et al. Proximal policy optimization algorithms. CoRR abs/1707.06347 (2017).
38. Seshia, S.A., Sadigh, D., and Sastry, S.S. Toward verified artificial intelligence. *Communications of the ACM 65*, 7 (2022), 46–55.
39. Sutton, R.S. and Barto, A.G. *Reinforcement Learning - An Introduction*. MIT Press (1998).
40. Tang, C. et al. Implementing action mask in proximal policy optimization (PPO) algorithm. *Information & Communications Technology Express 6*, 3 (2020), 200–203.

Bettina Könighofer is an assistant professor at Graz University of Technology, Graz, Austria.

Roderick Bloem (roderick.bloem@tugraz.at) is a professor at Graz University of Technology, Graz, Austria.

Nils Jansen is a professor at Ruhr University Bochum, Germany, and an associate professor at Radboud University, Nijmegen, the Netherlands.

Sebastian Junges is an assistant professor at Radboud University, Nijmegen, the Netherlands.

Stefan Pranger is a Ph.D. student at Graz University of Technology, Graz, Austria.

This research was funded in whole, or in part, by the Austrian Science Fund (FWF) [10.55776/S114] and [10.55776/W1255], by the State Government of Styria, Austria – Department Zukunftsfonds Steiermark, via the ERC Starting Grant 101077178 (DEUCE), and via the NWO Veni Grant 222.147 (ProMiSe). For the purpose of open access, the author has applied a CC BY public copyright license to any Author Accepted Manuscript version arising from this submission.