Shield Decomposition for Safe Reinforcement Learning in General Partially Observable Multi-Agent Environments

Daniel Melcer
Northeastern University
Boston, MA 02115
melcer.d@northeastern.edu

Christopher Amato*
Northeastern University
Boston, MA 02115
c.amato@northeastern.edu

Stavros Tripakis*
Northeastern University
Boston, MA 02115
stavros@northeastern.edu

Abstract

As Reinforcement Learning is increasingly used in safety-critical systems, it is important to restrict RL agents to only take safe actions. Shielding is a promising approach to this task; however, in multi-agent domains, shielding has previously been restricted to environments where all agents observe the same information. Most real-world tasks do not satisfy this strong assumption. We discuss the theoretical foundations of multi-agent shielding in environments with general partial observability and develop a novel shielding method which is effective in such domains. Through a series of experiments, we show that agents that use our shielding method are able to safely and successfully solve a variety of RL tasks, including tasks in which prior methods cannot be applied.

1 Introduction

Reinforcement learning is gaining popularity as a general method to solve a wide variety of tasks, such as car racing (Wurman et al., 2022), datacenter cooling (Lazic et al., 2018), and robotic warehouse operations (Knight, 2020). However, in all of these domains, a series of bad actions by the controller can lead to catastrophic failure. Reinforcement learning requires extensive exploration in order to learn a policy that can solve a given task (Sutton and Barto, 2018), and deep RL agents may still behave unpredictably, even after convergence (Clark and Amodei, 2016). Therefore, the use of RL without any modifications or external checks would be inappropriate for many potential applications.

In this paper, we focus on shielding, a technique which addresses this issue using ideas from Formal Methods (Alshiekh et al., 2018; Bloem et al., 2015). While shielding is a promising approach, existing methods for multi-agent shielding are limited by assumptions on observability—current shielding methods assume that all controllable agents observe all safety-relevant information (ElSayed-Aly et al., 2021; Carr et al., 2021), or at least the same safety-relevant information as all other agents (Melcer et al., 2022); alternatively, they are restricted to specific domains (Althoff and Dolan, 2014), or provide probabilistic guarantees by modeling the intentions of other agents in the environment (Nakamura and Bansal, 2023). There does not yet exist, to our knowledge, a shielding method that can protect against unsafe actions throughout the entire training and execution process in multi-agent domains with general partial observability; i.e., environments in which each agent may receive a different observation, the agents cannot communicate with each other, and there may not be any single agent that observes all safety-relevant information, where. However, such general domains are exactly the focus of multi-agent reinforcement learning (Gronauer and Diepold, 2022; Albrecht et al., 2023), as real-world tasks often present arbitrary restrictions on observability.

We first discuss the challenge of multi-agent shielding in general partially observable domains, and contribute an abstraction to express safety properties in such domains. We then present an algorithm

*Equal Advising

for shielding in these domains: We begin by introducing a lightweight algorithm to synthesize a shield for some instances of partially observable environments and describe certain conditions that cause this algorithm to fail. Next, we describe an extension to this algorithm that can find a shield in many of these harder instances, using a novel SAT encoding of the problem for efficiency. Finally, we use a series of experiments to show that this family of methods correctly prevents safety violations in a wider variety of environments than prior shielding methods. We also show that in many domains, decentralized shields accomplish this feat without negatively affecting the task-specific performance of the agent. We conclude by discussing several further research directions related to this work. Overall, our method is a significant step toward making multi-agent reinforcement learning methods safe in realistic partially observable settings.

2 Related Work

Several methods have extended shielding (Alshiekh et al., 2018; Bloem et al., 2015) to partially observable single-agent domains. Mazzi et al. (2021) use expert human knowledge to avoid unsafe selections within the POMCP algorithm (Silver and Veness, 2010). Carr et al. (2022); Junges et al. (2021) both describe belief support-based methods for shielding in such environments; i.e., methods that track the set of possible ground truth states. However, these methods assume a single observer, and have no means for coordinating actions among multiple agents to collaboratively enforce safety.

Alternatively, shielding has been extended to multi-agent environments, both with communication (ElSayed-Aly et al., 2021) and without communication (Melcer et al., 2022). However, these methods don't allow for general partial observability—all agents are assumed to observe the same safety-relevant information. There have been early attempts to extend shielding into partially observable multi-agent domains; for example, two-player one-sided POSGs (Carr et al., 2021). However, this method relies on several strong assumptions; for example, one agent must be able to observe all safety-relevant information.

Hsu et al. (2023) presents a survey of methods for safety-critical control, placing a variety of methods such as model-predictive shielding (Bastani, 2020) and control-barrier functions (Wieland and Allgöwer, 2007) into a single unified framework. This survey includes methods for agents to stay safe when interacting with humans or other agents, by modeling their behavior and expected actions (Nakamura and Bansal, 2023). In contrast, our method creates a decentralized shield for the whole system; we synthesize a specific protocol with the assumption that all agents follow it. The survey also discusses methods for enforcing the safety of multi-agent systems in specific domains (Althoff and Dolan, 2014); in contrast, our method is a general framework for any environment a model is available for.

3 Preliminaries

For set A, we use 2^A to denote the powerset of A. The Cartesian product of two sets A_1, A_2, denoted as $A_1 \times A_2$, is the set $\{(a_1, a_2) | a_1 \in A_1 \wedge a_2 \in A_2\}$.

3.1 Environments

A reinforcement learning environment may be complex: it may have an infinite state space, a complex stochastic transition structure, and a reward function which is irrelevant for safe operation. Therefore, shielding works utilize abstractions of the environment—typically finite-state transition systems—taking as input a safety specification over this abstraction (Alshiekh et al., 2018; ElSayed-Aly et al., 2021; Melcer et al., 2022).

We use a slightly different definition of an environment compared to prior works in order to better separate the dynamics of the environment itself from how agents observe and interact with it:

Definition 1 (Environment)**.** An *environment* is a non-deterministic finite transition system $E = (Q, Q_0, \mathbb{A}, \delta)$ where Q is a finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $\mathbb{A}$ is a finite set of actions, and $\delta : (Q \times \mathbb{A}) \to 2^Q$ is the transition function.

We note that $\delta(q, a)$ may be empty for some $q \in Q, a \in \mathbb{A}$—there might be no successor state.

While we will generate a shield that operates over an environment as defined above, the field of reinforcement learning typically focuses on more complex environments; i.e. those with an infinite state space or a reward function. For example, it is common to utilize a Dec-POMDP (Goldman and Zilberstein, 2004) as a very general formalization of the environment, and to develop reinforcement learning algorithms that operate over any Dec-POMDP. This is not an exclusive decision—shielding can operate in tandem with a reinforcement learning process in such environments, rather than replace it. Appendix E gives a definition for Dec-POMDPs, and discusses considerations for creating a useful abstraction for a given environment. Given a Dec-POMDP, the usual shielding workflow would be to create an abstracted environment (as given in Definition 1), synthesize a shield over this abstracted environment, and then train a set of reinforcement learning agents on the Dec-POMDP, while simulating the abstracted environment in lockstep to avoid taking unsafe actions. All environments in our evaluations are Dec-POMDPs for which we create an abstraction for shielding.

A *run* of environment E is a sequence of states and actions $q_0, a_0, q_1, a_1, \ldots$, where $q_0 \in Q_0$, and for all i, (1) $q_i \in Q$, (2) $a_i \in \mathbb{A}$, and (3) $q_{i+1} \in \delta(q_i, a_i)$. This run *visits* states $q_0, q_1, \ldots$. A state is *reachable* if there exists some run that visits it; all initial states are reachable by definition.

Action $a \in \mathbb{A}$ is *legal* at state $q \in Q$ if $\delta(q, a) \neq \emptyset$; q is a *deadlock* if no action is legal at q. An environment with no reachable deadlock states is *deadlock-free*. We can model valid terminal states in a deadlock-free environments by adding self-loop transitions in such states.

3.2 Centralized Shields

Definition 2 (Centralized Shield)**.** Given environment $E = (Q, Q_0, \mathbb{A}, \delta)$, a *centralized shield* for E is a function $\mathsf{CS} : Q \to 2^{\mathbb{A}}$.

For a given state $q \in Q$, $\mathsf{CS}(q)$ is the set of actions *permissible* by the shield at q.

The *shielded environment* $\mathsf{CS}(E)$ is the non-deterministic finite transition system $(Q, Q_0, \mathbb{A}, \delta_{\mathsf{CS}})$; i.e., E with a modified transition function δ_{CS} where for $q \in Q, a \in \mathbb{A}, \delta_{\mathsf{CS}}(q, a) = \delta(q, a)$ if $a \in \mathsf{CS}(q)$, and $\delta_{\mathsf{CS}}(q, a) = \emptyset$ otherwise (making actions prohibited by the shield illegal).

4 Problem Statement

4.1 Decentralized Environments

Existing abstractions for shielding are not expressive enough to describe how an environment is observed and controlled by many agents, especially where agents may have differing observations. We introduce the following structure to capture this information:

Definition 3 (Decentralization Setup)**.** Given environment $E = (Q, Q_0, \mathbb{A}, \delta)$, a *decentralization setup* for E is a tuple $\mathbb{D} = (k, \mathbb{A}_1, \ldots, \mathbb{A}_k, \Omega_1, \ldots, \Omega_k, \mathsf{obs}_1, \ldots, \mathsf{obs}_k)$ where $k \in \mathbb{N}$ is a number of agents; $\mathbb{A}_i$ is an individual action space for each agent $i \in [1..k]$ such that $\mathbb{A} = \mathbb{A}_1 \times \ldots \times \mathbb{A}_k$; Ω_i is a set of possible observations for each agent $i \in [1..k]$; and $\mathsf{obs}_i : Q \to 2^{\Omega_i}$ is a function for each agent that defines the nonempty set of possible observations of agent i for a given state.

This formulation assumes that the set of possible observations for a given agent is independent of other agents' observations at a given state. This is not a limiting assumption: if agents' observations are not independent, the underlying variables in the environment that affect these dependent observations should be explicitly modeled as part of the state. Note that in a fully observable environment, $\forall i \in [1..k], \Omega_i = Q$, and $\mathsf{obs}_i(q) = \{q\}$.

Definition 4 (Decentralized Shield)**.** Given environment $E = (Q, Q_0, \mathbb{A}, \delta)$ and decentralization setup $\mathbb{D} = (k, \mathbb{A}_1, \ldots, \mathbb{A}_k, \Omega_1, \ldots, \Omega_k, \mathsf{obs}_1, \ldots, \mathsf{obs}_k)$ for E, a decentralized shield over E and $\mathbb{D}$ is a tuple $\mathsf{DS} = (\mathsf{DS}_1, \ldots, \mathsf{DS}_k)$ such that $\mathsf{DS}_i : \Omega_i \to 2^{\mathbb{A}_i}$ for all $i \in [1..k]$.

A decentralized shield can also be applied to an environment; $\mathsf{DS}(E)$ is the transition system $(Q, Q_0, \mathbb{A}, \delta_{\mathsf{DS}})$—this is E with the modified transition function δ_{DS} where for $q \in Q, a = (a_1, \ldots, a_k) \in (\mathbb{A}_1 \times \ldots \times \mathbb{A}_k)$, $\delta_{\mathsf{DS}}(q, a) = \delta(q, a)$ when $\forall i \in [1..k], a_i \in \bigcup_{o_i \in \mathsf{obs}_i(q)} \mathsf{DS}_i(o_i)$ and $\emptyset$ otherwise. In other words, the shielded environment allows joint action a when all individual shields DS_i allow its component individual actions a_i for some possible observation of the current state.

4.2 Decentralized Shield Synthesis

Problem 1 (Decentralized Shield Synthesis)**.** Given a deadlock-free environment $E = (Q, Q_0, \mathbb{A}, \delta)$, a decentralization setup $\mathbb{D} = (k, \mathbb{A}_1, \ldots, \mathbb{A}_k, \Omega_1, \ldots, \Omega_k, \mathsf{obs}_1, \ldots, \mathsf{obs}_k)$ for E, and a set of bad states $Q_{bad} \subseteq Q$, find, if there exists, a decentralized shield $\mathsf{DS} = (\mathsf{DS}_1, \ldots, \mathsf{DS}_k)$ over E and $\mathbb{D}$ such that (1) all states in Q_{bad} are unreachable in $\mathsf{DS}(E)$, and (2) for every reachable state q of $\mathsf{DS}(E)$, for every possible $o_1 \in \mathsf{obs}_1(q), \ldots, o_k \in \mathsf{obs}_k(q)$, it holds that $\mathsf{DS}_1(o_1) \times \ldots \times \mathsf{DS}_k(o_k)$ is a nonempty subset of the legal actions of E at q; i.e., $\forall a \in (\mathsf{DS}_1(o_1) \times \ldots \times \mathsf{DS}_k(o_k)), \delta(q, a) \neq \emptyset$.

Condition (1) ensures that the resulting system is *safe*, as it is impossible for a "bad" state to be reached in any run. To understand condition (2), consider the alternative where Problem 1 simply demanded that $\mathsf{DS}(E)$ were deadlock-free—this would imply that the set of joint actions allowed by the decentralized shield may contain some illegal actions, as long as it contains at least one legal action. This does not work well when the joint action is collectively chosen by many independent agents. Rather, as written, condition (2) means that each agent i can independently choose an individual action from DS_i, and the joint action is guaranteed to be legal. We must still show that a shield that satisfies these conditions results in a deadlock-free environment:

Theorem 1. *Given a decentralized shield* DS *that satisfies Problem 1,* $\mathsf{DS}(E)$ *is deadlock-free.*

A proof sketch of this theorem is given in Appendix J.1.

4.3 Example

Consider a driverless car scenario with many agents, and a complex reward function based on energy efficiency and time to reach the destination. While RL may be able to optimize for this reward, there is no appropriate penalty for a safety violation—it should be "infinitely" high to reflect that a collision absolutely should not occur no matter the time savings, but this presents a practical reward scaling problem for a RL agent, and the agent must still experience this reward to learn it. We create a shield here to enforce a safety constraint, even at the start of training.

For shielding purposes, we create a simpler environment E that only tracks relative agent, pedestrian, and obstacle positions, ignoring non-safety-relevant information such as efficiency or passenger temperature control. The abstracted environment E, and the set of unsafe states Q_{bad}, must be constructed such that if all states in Q_{bad} are successfully avoided, the underlying environment is not in an unsafe state (i.e., a collision). The decentralization setup describes how each car is observed and controlled by individual agents, rather than a centralized controller.

5 Method

Shield synthesis is essentially a controller synthesis problem (Bloem et al., 2015; Ramadge and Wonham, 1987; Pnueli and Rosner, 1989); decentralized shield synthesis is therefore essentially a decentralized controller synthesis problem. Such problems are known to be generally undecidable when agents receive different observations from the environment (Pnueli and Rosner, 1990; Thistle, 2005; Tripakis, 2004). Despite this undecidability, we present an algorithm that successfully synthesizes a decentralized shield in many cases where previous methods, such as those presented in

ElSayed-Aly et al. (2021) and Melcer et al. (2022), make input assumptions that are too prohibitive. Our algorithm always terminates; however, as any terminating algorithm for an undecidable problem must, it sometimes reports failure in cases where a shield may exist.

At a high level, we first synthesize a centralized shield, then decompose it into a decentralized shield.

Problem 2 (Centralized Shield Synthesis)**.** Given a deadlock-free environment $E = (Q, Q_0, \mathbb{A}, \delta)$, and a set of bad states $Q_{bad} \subseteq Q$, find, if there exists, a centralized shield CS for E such that (1) all states in Q_{bad} are unreachable in $\mathsf{CS}(E)$, and (2) for every reachable state q of $\mathsf{CS}(E)$, $\mathsf{CS}(q)$ is nonempty and contains only legal actions at q; i.e., $\forall a \in \mathsf{CS}(q), \delta(q, a) \neq \emptyset$.

Existing methods solve the centralized shield synthesis problem though a simple fixpoint-finding process (Bloem et al., 2015; ElSayed-Aly et al., 2021). The resulting centralized shields have the property that they do not disallow any actions which do not absolutely need to be disallowed. The cited methods define an environment slightly differently from each other, and from how we define an environment; we detail the adaptation of these methods to our setting in Algorithm 2 (Appendix).

Problem 3 (Shield Decomposition)**.** Given an environment $E = (Q, Q_0, \mathbb{A}, \delta)$, a centralized shield CS for E that satisfies the conditions from Problem 2, and a decentralization setup $\mathbb{D} = (k, \mathbb{A}_1, \ldots, \mathbb{A}_k, \Omega_1, \ldots, \Omega_k, \mathsf{obs}_1, \ldots, \mathsf{obs}_k)$ for E, find a decentralized shield $\mathsf{DS} = (\mathsf{DS}_1, \ldots, \mathsf{DS}_k)$ over E and $\mathbb{D}$ such that for all reachable $q \in Q, \forall o_1 \in \mathsf{obs}_1(q), \ldots, \forall o_k \in \mathsf{obs}_k(q), \mathsf{DS}_1(o_1) \times \ldots \times \mathsf{DS}_k(o_k)$ is a non-empty subset of $\mathsf{CS}(q)$.

Theorem 2. *A decentralized shield* DS *that satisfies the requirements of Problem 3 also satisfies the requirements of Problem 1.*

A proof sketch of this theorem is given in Appendix J.2.

5.1 A Naive Algorithm for Centralized Shield Decomposition

One potential solution to Problem 3 is as follows. Given a centralized shield CS for environment $E = (Q, Q_0, \mathbb{A}, \delta)$ and decentralization setup $\mathbb{D} = (k, \mathbb{A}_1, \ldots, \mathbb{A}_k, \Omega_1, \ldots, \Omega_k, \mathsf{obs}_1, \ldots, \mathsf{obs}_k)$, we first find, for each state $q \in Q$, a sequence of sets of individual actions $A_1^q \subseteq \mathbb{A}_1, \ldots, A_k^q \subseteq \mathbb{A}_k$, such that $(A_1^q \times \ldots \times A_k^q) \subseteq \mathsf{CS}(q)$. We refer to such a sequence of sets as a *state-decentralization* for state q. Such a state-decentralization always exists for any reachable state: as $\mathsf{CS}(q)$ is nonempty by assumption, we can choose any element $a = (a_1, \ldots, a_k) \in \mathsf{CS}(q)$, and set $A_i^q = \{a_i\}$ for all i.

Second, for each agent i, we define the function $R_i : \Omega_i \to 2^Q$ where $R_i(o_i) = \{q \in Q | o_i \in \mathsf{obs}_i(q)\}$; i.e., the set of states where agent i may encounter observation o_i. We construct a decentralized shield $\mathsf{DS} = (\mathsf{DS}_1, \ldots, \mathsf{DS}_k)$ where $\mathsf{DS}_i(o_i) = \bigcap_{r \in R_i(o_i)} A_i^r$.

Finally, we check that $\mathsf{DS}_i(o_i) \neq \emptyset$ for every $i \in [1..k]$ and $o_i \in \Omega_i$. If this holds, the algorithm has finished: for every state q and observation $o_1 \in \mathsf{obs}_1(q), \ldots, o_k \in \mathsf{obs}_k(q)$, $(\bigcap_{r \in R_1(o_1)} A_1^r) \times \ldots \times (\bigcap_{r \in R_k(o_k)} A_k^r)$ is trivially a subset of $A_1^q \times \ldots \times A_k^q$, which itself is, by definition, a subset of $\mathsf{CS}(q)$. Otherwise, the algorithm reports failure; the chosen state-decentralizations result in a deadlock. We name this process the *naive algorithm*, detailed in Algorithms 3 and 4 (Appendix).

This algorithm may be improved by restricting its consideration to *maximally permissive* state-decentralizations; i.e., sets of actions $A_1^q \subseteq \mathbb{A}_i, \ldots, A_k^q \subseteq \mathbb{A}_k$ for which $(A_1^q \times \ldots \times A_k^q) \subseteq \mathsf{CS}(q)$, but $\nexists i \in [1..k], a_i \in \mathbb{A}_i$ where $a_i \notin A_i^q$ and $(A_1^q \times \ldots A_i^q \cup \{a_i\} \ldots \times A_k^q) \subseteq \mathsf{CS}(q)$—in other words, state-decentralizations where no individual actions may be added while retaining safety. If there exists any q such that $A^q = (A_1^q, \ldots, A_k^q)$, and shield decomposition succeeds, there exists a safe maximally permissive state-decentralization $A'^q = (A'^q_1, \ldots, A'^q_k)$ where shield decomposition would succeed if we replace A^q with A'^q—$\forall i \in [1..k], A'^q_i \supseteq A_i^q$. Algorithm 6 (Appendix) describes how to compute the set of maximally permissive state-decentralizations $MPD_{\mathsf{CS}}(q)$ for a given state $q \in Q$.

Algorithm 1 Shielded Training Overview

Input
  $E = (Q, Q_0, \mathbb{A}, \delta)$; // Environment
  $\mathbb{D} = (k, \mathbb{A}_1, \ldots, \mathbb{A}_k, \Omega_1, \ldots, \Omega_k, \mathsf{obs}_1, \ldots, \mathsf{obs}_k)$ // Decentralization Setup
  $Q_{bad} : 2^Q$ // A set of unsafe states
  $r_p : \mathbb{R}^-$ // Penalty reward
  $\mathcal{M}$ // RL environment with agents $[1..k]$, actions $\mathbb{A}_1, \ldots, \mathbb{A}_k$, observations $\Omega_1, \ldots, \Omega_k$
  $\pi_i : (\Omega_i \times \mathbb{A}_i)^* \times \Omega_i \to \mathbb{A}_i$; // Initial policies for each $i \in [1..k]$
procedure TrainWithShield$(E, \mathbb{D}, Q_{bad}, \mathcal{M}, (\pi_1, \ldots, \pi_k), r_p)$
  CS := SynthCShield(E, Q_{bad}) // Appendix, Algorithm 2
  $(\mathsf{DS}_1, \ldots, \mathsf{DS}_k)$:= DecomposeCShield$(E, \mathbb{D}, \mathsf{CS})$ // Appendix, Algorithm 3
  while π_i not converged, **in parallel** for each $i \in [1..k]$ **do** // Agents act independently
    o_i := Initial observation
    $h_i := (o_i)$
    while Episode not terminated **do**
      $a_i := \pi_i(h)$
      $safe := \mathsf{DS}_i(o_i)$ // o_i may be abstracted before being passed to DS_i
      if $a_i \in safe$ **then**
        r_i, o_i := Reward, observation after a_i in $\mathcal{M}$ // Joint action is $(a_1, \ldots, a_k)$
        $h'_i := h_i + (a_i, o_i)$
        Train π_i with (h_i, a_i, r_i, h'_i)
      else
        a'_i := Arbitrary element of *safe*
        r_i, o_i := Reward, obs after a'_i in $\mathcal{M}$
        $h'_i := h_i + (a'_i, o_i)$
        Train π_i with (h_i, a'_i, r_i, h'_i) **or** $(h_i, a_i, r_i + r_p, h_i + (a_i, o_i))$ with probability 0.5
      end if
      $h := h'$
    end while
  end while
end procedure

5.1.1 Weaknesses of the Naive Algorithm

Despite the improvements from restricting the set of state-decentralizations, the naive algorithm presented in Section 5.1 has a clear weakness: if the emptiness check fails, the algorithm has failed to synthesize a decentralized shield.

For example, consider a simple environment with two agents; each agent has two actions, a_0 and a_1. In state s_0, both agents must select the same action; in s_1, the agents must select opposite actions. Agent 1 receives an observation that can distinguish between these cases, but agent 2 does not.

One possible set of maximally permissive state-decentralizations is that agent 1 must always select a_0, and then agent 2 must select a_0 or a_1 in s_1 and s_2, respectively. If this set of state-decentralizations were chosen, the naive algorithm would fail—agent 2 is unable to distinguish between s_0 and s_1, but there are no individual actions allowed in both states. There is an alternative set of state-decentralizations, where agent 2 always selects a_0, and then agent 1 must select a_0 in state s_0, or a_1 in s_1. With this set of state-decentralizations, shield decomposition succeeds.

However, the number of possible combinations of state-decentralizations grows exponentially with the state space, rendering a brute-force approach intractable for any non-trivial environment.

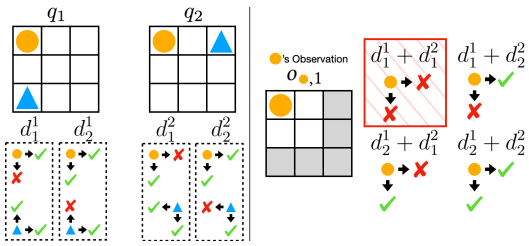

Figure 1: Illustration of the decomposition problem for an environment with a small observation radius. For each state, there are several possible state-decentralizations ($d_1^1, d_2^1 \in MPD_{\mathsf{CS}}(q_1)$); one per state must be chosen in advance ($A^{q_1} = d_1^1$ or d_2^1). Depending on the chosen state-decentralizations, there may not be any safe actions available to the agent. Example further described in Appendix C.

5.2 A Comprehensive Constraint-Based Algorithm

To overcome the limitations of the naive algorithm discussed in Section 5.1.1, we introduce a method to synthesize a decentralized shield by taking advantage of highly efficient SAT solvers.

5.2.1 SAT Solvers

SAT is a well-studied problem, defined as: given a set of boolean variables, and a boolean expression over these variables, find an assignment to the boolean variables such that the expression's value is true, or disprove the existence of such an assignment. This is an NP-complete problem; however, modern SAT solvers can efficiently solve very large instances of the SAT problem (Malik and Zhang, 2009). We use the Kissat solver (Biere et al., 2020) for all instances of SAT described in this paper.

5.2.2 Encoding Shield Decomposition in SAT

The core of the problem is to choose a specific state-decentralization in $MPD_{\mathsf{CS}}(q)$ for every reachable $q \in Q$ (1). If a state-decentralization $A^q = (A_1^q, \ldots, A_k^q) \in MPD_{\mathsf{CS}}(q)$ is chosen, and for some agent $i \in [1..k]$ there exists action $a_i \notin A_i^q$, this means that agent i is not allowed to take action a_i in state q, so a_i should not be a member of $\bigcup_{o_i \in \mathsf{obs}_i(q)} \mathsf{DS}_i(o_i)$, and thus a_i should not be a member of $\mathsf{DS}_i(o_i)$ for any $o_i \in \mathsf{obs}_i(q)$ (2). Finally, there must be an action available for every observation (3). We encode these conditions as the following set of constraints:

$$\bigwedge_{q \in Q} \bigoplus_{d^q \in MPD_{\mathsf{CS}}(q)} A^q = d^q \tag{1}$$

$$\bigwedge_{q \in Q, d^q \in MPD_{\mathsf{CS}}(q), i \in [1..k], a \in (\mathbb{A}_i \setminus d^q[i]), o_i \in \mathsf{obs}_i(q)} A^q = d^q \implies a \notin \mathsf{DS}_i(o_i) \tag{2}$$

$$\bigwedge_{i \in [1..k], o_i \in \Omega_i} \bigvee_{a \in \mathbb{A}_i} a \in \mathsf{DS}_i(o_i) \tag{3}$$

These constraints permit a natural SAT encoding, with one set of variables representing if $A^q = d^q$ for each $q \in Q, d^q \in MPD_{\mathsf{CS}}(q)$, and another set of variables representing $a \in \mathsf{DS}_i(o_i)$ for each $i \in [1..k], o_i \in \Omega_i, a \in \mathbb{A}_i$. By construction, if the solver can satisfy the constraints listed above, the resulting decentralized shield satisfies the requirements of Problem 3. This procedure (referred to as the *SAT-based algorithm*) is shown in Algorithms 3 and 5.

6 Experiments

We first replicate environments found in prior multi-agent shielding works (ElSayed-Aly et al., 2021; Melcer et al., 2022). Gridworld-Collision (**Grid-Col**), introduced by Melo and Veloso (2009), is

a 2-agent gridworld with four maps. In Particle-Momentum, agents move with inertia in an open gridworld, observing either both relative positions and velocities (**Particle-P-V**), or only relative positions (**Particle-P**).

In the above domains, all agents receive identical observations. By contrast, in **Nearby-Obs**, agents are only able to observe other agents up to 2 Manhattan-distance units away; when the agents are farther apart, they are only able to observe their own position. Otherwise, the environment dynamics are similar to Grid-Col.

Finally, we create Flashlight. In this set of environments, agents can only observe squares which are directly adjacent, and are forced to move at every step. Each agent is equipped with a flashlight; if an agent's light is turned on, agents can observe each other up to a 5-unit radius for one time step. Afterwards, the flashlight must recharge for several steps, during which attempts to turn it on will fail. Each agent is able to observe whether its own flashlight is on; if its light is on and the other agent is not visible, this implies that the other agent is more than five units away. We instantiate this environment in both 6x6 (**Flashlight-6**) and 10x10 (**Flashlight-10**) sizes, with a variety of recharge times.

In all of our experiments, there exists an underlying Dec-POMDP that assigns rewards; this Dec-POMDP may be arbitrarily complex. For example, in the Flashlight environments, part of the state space of the Dec-POMDP tracks the current flashlight charge level, in order to determine if an action to turn the light on will succeed. We also create an abstraction of each of these environments for shielding purposes; at minimum, these abstractions ignore rewards, but they may be a more simplified version of the environment. For example, the Flashlight abstractions do not maintain charge level; rather, the action to turn the flashlight on will nondeterministically fail in the abstraction. As this abstraction satisfies the properties in Appendix E, it is straightforward to use this abstraction to enforce safety while running a conventional RL algorithm on the Dec-POMDP.

Extended descriptions of all environments are located in Appendix A.

6.1 Shield Decentralization

We first focus on the ability to synthesize a decentralized shield using a variety of methods. Overall, while current methods are sufficient for shielding in fully observable and simple partially observable domains, they fail in more complex environments, leaving only our method for guaranteeing safety.

As shown in Table 1, all methods are able to produce a decentralized shield for fully observable domains. In order to decompose the centralized shield for Particle-P, we needed to modify our implementation to account for history; an overview of this modification is given in Appendix D. In Nearby-Obs, prior shielding methods are unable to calculate a decentralized shield, as the agents may observe different information. In contrast, both of our algorithms succeed, without even needing to account for history, as all states which can be confused with each other (agents are > 2 units apart) permit the same safe actions (agents have no risk of colliding in one step, so all actions are safe). However, Flashlight-6 and Flashlight-10 contain several states which can be confused with each other, each with different sets of safe actions; our SAT-based methods are the only ones which succeed here. An analysis of the runtime of the decentralization algorithm for each environment is located in Appendix F.

6.2 RL Performance

While the primary purpose of shield synthesis is safety, we would like to ensure that the shields are permissive as well. The goal of a permissive shield is to facilitate successful training of a reinforcement learning agent; therefore, rather than creating a metric which measures properties of the shields in isolation, we train RL agents using each shielding method in a variety of environments, and evaluate agent performance. We show that agents shielded with our method perform about the same, or potentially better compared to centralized-shielded or unshielded agents, while our method ensures safe and decentralized execution in the widest variety of environments.

Note that shielding is independent of the underlying RL algorithm; we believe that our method can be applied to centralized training/decentralized execution methods (Oliehoek et al., 2008) such as MADDPG (Lowe et al., 2017) or QMIX (Rashid et al., 2020) with minimal modification; specific applications of shielding to state-of-the-art MARL methods are an area for future research.

6.2.1 Training Details

In order to best compare our methods to prior results, we replicate the same agent architecture and hyperparameters for Grid-Col, Particle-P, and Particle-P-V as in Melcer et al. (2022). As an overview, Grid-Col uses tabular individual Q-learning. Particle-P and Particle-P-V use Deep Double Q-learning, without any recurrent layer (Van Hasselt et al., 2016). For Nearby-Obs, Flashlight-6, and Flashlight-10, we add a recurrent layer. Further details are located in Appendix G.

For all shielded agents, we use post-posed shielding (Alshiekh et al., 2018). Specifically, if an agent's chosen action a is considered unsafe by the shield, an arbitrary safe action a' is given to the environment, yielding reward r. When training the agent, we randomly ($p = 0.5$) select whether to use action a' and reward r, as actually occurred in the environment, or action a and reward $r + r_p$, where $r_p = -10$ is a penalty reward modifier, so that the agent will be biased to non-arbitrarily choose a safe action in the future. For non-recurrent agents, we add both transitions to the replay buffer. An overview of the shielded training method is given as Algorithm 1.

6.2.2 Results

As shown in Table 1, and in the full results in the Appendix (Tables 5, 6), all agents perform about equally to each other in Grid-Col and Particle-P-V; however, only the shielded agents have zero safety violations in all environments. We note that while the SAT-based method finds a decomposition in a wider variety of environments than the naive method, **if both methods succeed in decomposition, there is no reason to expect the shield produced by one method to perform better than the other**, other than by random chance. This doesn't necessarily apply to shields which incorporate history, as the additional information available for such shields allows for more permissive action selection.

In Particle-P, the shielded agents all achieve better RL performance than the unshielded agents, but otherwise perform similarly to each other. It is initially surprising that decentralized agents would perform so close to agents with a centralized shield, as centralization should intuitively lead to much better performance. One possible explanation is that a decentralized shield consistently permits or prohibits actions, depending solely on the agent's own observation. In contrast, a centralized shield will allow an individual action based on unobserved state information, or actions that other agents take. Because the RL method itself is fully decentralized in both cases, the changes in allowed actions due to centralized shielding may appear as nonstationarity in the environment, hurting performance and balancing out any benefits of extra permissiveness.

In Nearby-Obs, the decentralized-shielded agents perform at least as well as, and potentially better than, the unshielded agents, without incurring any safety violations; full results are listed in Table 7, in the Appendix. Additionally, they hold up quite well against agents that use a centralized shield.

More interesting are the Flashlight-6 and Flashlight-10 domains, where agents have no chance of safe navigation without the flashlight, but can locate the other agent by turning on the flashlight. While it is possible to synthesize a restrictive shield in this domain that does not include history, the results in Table 2 show that including history during shield synthesis often improves performance. Without knowledge of the other agents' position, *any* uncoordinated joint action has the potential for a collision. With history, the agents can guarantee collision-free movement for a short length of time after the flashlight turns off.

Full results and further discussion are located in the Appendix (Tables 8 and 9).

Table 1: A summary of shield decomposition and RL execution results. Random starts are used for all evaluations. 'X' denotes that shield decentralization fails, or violates input assumptions for a given method. Otherwise, we report results for the minimum history length where shield decomposition succeeds (one step for Particle-P, no history for others). Results show average discounted sum of rewards. We report standard error over 50 seeds. Unshielded agents include average sum of safety violations over 100 episodes in parentheses.

Domain	Melcer et al. (2022)	Naive	SAT	Centralized	None (Violations)
Grid-Col	29.4 ± 2.0	29.5 ± 2.4	30.1 ± 2.2	31.1 ± 2.5	28.1 ± 1.9 (0.9)
Particle-P-V	51.3 ± 2.1	53.0 ± 2.4	50.0 ± 1.8	51.7 ± 2.0	48.7 ± 2.0 (0.1)
Particle-P	26.0 ± 3.2	28.8 ± 2.0	27.6 ± 3.1	29.7 ± 3.8	17.0 ± 3.5 (69.2)
Nearby-Obs	X	61.1 ± 5.7	45.0 ± 7.3	61.8 ± 5.4	5.6 ± 8.8 (113.4)
Flashlight	X	X	Table 2	Table 2	Table 2

Table 2: Average discounted sum of rewards and standard error in Flashlight-6 and Flashlight-10, random starting locations, 10 seeds. Prior shielding methods (Melcer et al., 2022) and the naive decomposition algorithm all fail to synthesize a shield.

Size	Recharge	SAT–No History	SAT–1 Step History	Centralized	None (Violations)
6x6	3	65.4 ± 1.1	74.9 ± 0.3	84.7 ± 0.3	83.4 ± 0.2 (7.2)
	4	53.1 ± 1.2	68.5 ± 0.4	83.7 ± 0.2	82.7 ± 0.7 (5.1)
	5	-20.6 ± 14.0	56.0 ± 3.7	81.6 ± 0.9	83.5 ± 0.3 (5.2)
	6	-23.8 ± 15.8	30.0 ± 12.7	76.9 ± 7.0	83.5 ± 0.3 (5.5)
10x10	2	-52.4 ± 0.3	57.2 ± 2.4	16.9 ± 14.3	43.2 ± 11.4 (6.9)
	3	-109.6 ± 7.4	38.1 ± 3.6	14.5 ± 17.3	18.3 ± 20.3 (9.3)

7 Societal Impact

Shielding can be a powerful tool to prevent mistakes as the result of an incorrectly trained RL agent. However, as with other shielding works, there is an inherent risk that the creator or user of an RL system could be overconfident in a shielded RL agent—it is possible for an environment or safety specification to be incorrectly specified, or for there to be a bug in the implementation of the shield synthesis tool. Care must be taken when applying shielding to a given problem, and there should be redundant systems in place for any safety-critical process.

8 Conclusion & Future Work

Communication-free multi-agent shielding was previously limited to domains in which all agents received enough information to deduce the true environment state, or operated on environments with significantly different assumptions on environment structure and agent interaction. In this paper, we developed, to our knowledge, the first decentralized shield synthesis method that allows agents to enforce a safety specification without communication in environments with general partial observability—the first shielding method for the general Dec-POMDP case. These shields allow agents to act safely and are often permissive enough to allow agents to successfully solve difficult reinforcement learning problems under a shielding protocol.

While our approach can scale by using small abstractions for arbitrarily large MARL environments, there is still work to be done to improve the scalability and performance of decentralized shielding when the abstractions themselves grow larger. Symbolic approaches have led to several orders-of-magnitude increases of supported input sizes for model checking (Clarke et al., 2018); the application of these methods to shield synthesis and decentralization is a promising area for future work. Additionally, there are often many different possible shields which are all safe for a given environment,

but some may be more conducive for learning a successful policy. It may be possible to develop metrics for measuring the quality of a shield, or to use information observed during RL training to iteratively improve a shield while maintaining safety.

The methods presented in this paper represent a significant step towards making reinforcement learning safe in realistic partially observable settings.

Acknowledgements

This work has been supported by NSF CCF award #2319500, *FMitF: Track I: Safe Multi-Agent Reinforcement Learning with Shielding*, and used the Discovery cluster, supported by Northeastern University's Research Computing team.

References

Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. 2023. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, Boston, MA. https://www.marl-book.com

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32. AAAI Conference on Artificial Intelligence, New Orleans, LA, 10 pages.

Matthias Althoff and John M. Dolan. 2014. Online Verification of Automated Road Vehicles Using Reachability Analysis. *IEEE Transactions on Robotics* 30, 4 (2014), 903–918. https://doi.org/10.1109/TRO.2014.2312453

Osbert Bastani. 2020. Safe Reinforcement Learning with Nonlinear Dynamics via Model Predictive Shielding. arXiv:1905.10691 [cs.LG]

Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. 2020. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling Entering the SAT Competition 2020. In *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions (Department of Computer Science Report Series B, Vol. B-2020-1)*, Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda (Eds.). University of Helsinki, Alghero, Italy, 51–53.

Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. 2015. Shield synthesis: Runtime enforcement for reactive systems. In *Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings 21*. Springer, International Conference on Tools and Algorithms for the Construction and Analysis of Systems, London, UK, 533–548.

Steven Carr, Nils Jansen, Suda Bharadwaj, M.T.J. Spaan, and Ufuk Topcu. 2021. Safe Policies for Factored Partially Observable Stochastic Games. In *Robotics: Science and System XVII*, Dylan A. Shell, Marc Toussaint, and M. Ani Hsieh (Eds.). Robotics: Science and Systems, Virtual, 11 pages. https://doi.org/10.15607/RSS.2021.XVII.079 Robotics: Science and Systems XVII, 2021 ; Conference date: 12-07-2021 Through 16-07-2021.

Steven Carr, Nils Jansen, Sebastian Junges, and Ufuk Topcu. 2022. Safe reinforcement learning via shielding for pomdps. , 21 pages.

Jack Clark and Dario Amodei. 2016. Faulty reward functions in the wild. https://openai.com/research/faulty-reward-functions

Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (Eds.). 2018. *Handbook of Model Checking*. Springer International Publishing, New York, NY. https://doi.org/10.1007/978-3-319-10575-8

Ingy ElSayed-Aly, Suda Bharadwaj, Christopher Amato, Rüdiger Ehlers, Ufuk Topcu, and Lu Feng. 2021. Safe Multi-Agent Reinforcement Learning via Shielding. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems* (Virtual Event, United Kingdom) *(AAMAS '21)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 483–491.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep Sparse Rectifier Neural Networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 15)*, Geoffrey Gordon, David Dunson, and Miroslav Dudík (Eds.). PMLR, Fort Lauderdale, FL, USA, 315–323. https://proceedings.mlr.press/v15/glorot11a.html

C. V. Goldman and S. Zilberstein. 2004. Decentralized Control of Cooperative Systems: Categorization and Complexity Analysis. *Journal of Artificial Intelligence Research* 22 (Nov. 2004), 143–174. https://doi.org/10.1613/jair.1427

Sven Gronauer and Klaus Diepold. 2022. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review* 55, 2 (Feb 2022), 895–943. https://doi.org/10.1007/s10462-021-09996-w

Kai-Chieh Hsu, Haimin Hu, and Jaime Fernández Fisac. 2023. The Safety Filter: A Unified View of Safety-Critical Control in Autonomous Systems. arXiv:2309.05837 [eess.SY]

Sebastian Junges, Nils Jansen, and Sanjit A. Seshia. 2021. Enforcing Almost-Sure Reachability in POMDPs. In *Computer Aided Verification*, Alexandra Silva and K. Rustan M. Leino (Eds.). Springer International Publishing, Cham, 602–625.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

Will Knight. 2020. AI Helps Warehouse Robots Pick Up New Tricks. https://www.wired.com/story/ai-helps-warehouse-bots-pick-new-skills/

Nevena Lazic, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, MK Ryu, and Greg Imwalle. 2018. Data center cooling using model-predictive control. *Advances in Neural Information Processing Systems* 31 (2018), 10 pages.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *Neural Information Processing Systems (NIPS)* 30 (2017), 12 pages.

Sharad Malik and Lintao Zhang. 2009. Boolean Satisfiability: From Theoretical Hardness to Practical Success. *Commun. ACM* 52, 8 (2009), 76–82.

Giulio Mazzi, Alberto Castellini, and Alessandro Farinelli. 2021. Rule-based Shielding for Partially Observable Monte-Carlo Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 31. International Conference on Automated Planning and Scheduling, Virtual, 243–251.

Daniel Melcer, Christopher Amato, and Stavros Tripakis. 2022. Shield Decentralization for Safe Multi-Agent Reinforcement Learning. *Advances in Neural Information Processing Systems* 36 (2022), 13 pages.

Francisco S. Melo and Manuela Veloso. 2009. Learning of Coordination: Exploiting Sparse Interactions in Multiagent Systems. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2* (Budapest, Hungary) *(AAMAS '09)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 773–780.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning.

Kensuke Nakamura and Somil Bansal. 2023. Online Update of Safety Assurances Using Confidence-Based Predictions. arXiv:2210.01199 [cs.RO]

F. A. Oliehoek, M. T. J. Spaan, and N. Vlassis. 2008. Optimal and Approximate Q-value Functions for Decentralized POMDPs. *Journal of Artificial Intelligence Research* 32 (may 2008), 289–353. https://doi.org/10.1613/jair.2447

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., Vancouver, Canada, 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Amir Pnueli and Roni Rosner. 1989. On the synthesis of a reactive module. In *ACM Symp. POPL*. Principles of Programming Languages, Austin, TX, 12 pages.

A. Pnueli and R. Rosner. 1990. Distributed reactive systems are hard to synthesize. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*. IEEE Symposium on Foundations of Computer Science, St. Louis, MO, 746–757 vol.2. https://doi.org/10.1109/FSCS.1990.89597

P. Ramadge and W. Wonham. 1987. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.* 25, 1 (Jan. 1987), 25 pages.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2020. Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. *J. Mach. Learn. Res.* 21, 1, Article 178 (jan 2020), 51 pages.

David Silver and Joel Veness. 2010. Monte-Carlo Planning in Large POMDPs. In *Advances in Neural Information Processing Systems*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta (Eds.), Vol. 23. Curran Associates, Inc., Vancouver, Canada. https://proceedings.neurips.cc/paper_files/paper/2010/file/edfbe1afcf9246bb0d40eb4d8027d90f-Paper.pdf

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

John G. Thistle. 2005. Undecidability in decentralized supervision. *Systems & Control Letters* 54, 5 (2005), 503–509. https://doi.org/10.1016/j.sysconle.2004.10.002

Stavros Tripakis. 2004. Undecidable Problems of Decentralized Observation and Control on Regular Languages. *Inform. Process. Lett.* 90, 1 (April 2004), 21–28. https://doi.org/10.1016/j.ipl.2004.01.004

Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 30. Association for the Advancement of Artificial Intelligence, Pheonix, AZ, 2094–2100.

Peter Wieland and Frank Allgöwer. 2007. CONSTRUCTIVE SAFETY USING CONTROL BARRIER FUNCTIONS. *IFAC Proceedings Volumes* 40, 12 (2007), 462–467. https://doi.org/10.3182/20070822-3-ZA-2920.00076 7th IFAC Symposium on Nonlinear Control Systems.

Peter R. Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J. Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, Leilani Gilpin, Piyush Khandelwal, Varun Kompella, HaoChih Lin, Patrick MacAlpine, Declan Oller, Takuma Seno, Craig Sherstan, Michael D. Thomure, Houmehr Aghabozorgi, Leon Barrett, Rory

Douglas, Dion Whitehead, Peter Dürr, Peter Stone, Michael Spranger, and Hiroaki Kitano. 2022. Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature* 602, 7896 (Feb. 2022), 223–228. https://doi.org/10.1038/s41586-021-04357-7

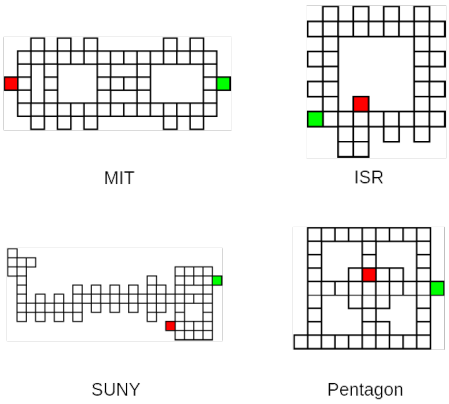

Figure 2: The four maps of Grid-Col and Nearby-Obs. One agent's goal is the red square, the other agent's goal is green. When using the fixed-start environments, agents begin in the opposite agent's goal.

A Environment Descriptions

As mentioned in Section 6, Grid-Col is a commonly used domain (Melo and Veloso, 2009; ElSayed-Aly et al., 2021; Melcer et al., 2022) consisting of two agents, each with five actions—up, left, down, right, and no-op. There are four maps, shown in Figure 2. Agents receive a +100 reward and the episode terminates when they both reach the goal, and a -30 penalty for colliding; collisions are also disallowed by the shield's safety specification. Otherwise, an agent receives a -10 reward when hitting a wall, or a -1 reward for all other time steps. The episode also terminates after 500 time steps without reaching the goal.

Particle-P and Particle-P-V are also pre-existing domains (Melcer et al., 2022) which we replicate for comparison. In these domains, agents can be up to 10 units away from each other. If they stray any further, or collide with each other, this is considered a safety violation by the shield; any unshielded agents which attempt this receive a -30 reward. When agent 1 takes the action corresponding to, for example, "right", the relative x velocity increases; if agent 2 takes the same action, the relative x velocity decreases. Relative velocity is capped at an absolute value of 2 for each axis. The relative position then changes according to the velocity. The goal is for agent 1 to be 9 units below and to the right of agent 2. In the fixed-start environment, agent 1 starts 9 units above, and 9 units to the left of agent 2, with no momentum. In the randomized environment, the agents start at a random relative position, but still with no momentum. Upon both agents reaching the goal, they receive a +100 reward; otherwise, a -1 reward at each step.

The Nearby-Obs domain is identical to Grid-Col except for observability. Flashlight-6 and Flashlight-10 are similar, except that the maps are 6x6 or 10x10 squares respectively, with only exterior walls. We keep the no-op action available, but denote it as unsafe by the shield, and add a -30 penalty if an agent attempts it anyways. Independently of the movement choice, the agents also choose whether to attempt to turn on the flashlight; the agents must still choose to move in a cardinal direction when attempting the flashlight, or face a safety violation for staying still. Each agent is able to observe whether its own flashlight is on—if its light is on and the other agent is not visible, this implies that the other agent is far away. There is no penalty for turning the light on, or for attempting to do so while it is being recharged.

B Shield Synthesis

B.1 CFOS Synthesis Algorithms

Algorithm 2 uses a standard fixpoint-finding procedure to obtain a centralized shield from an environment and set of bad states. The algorithm will always terminate: Q_{Unsafe} cannot grow forever as it will run out of states in Q to add, at which point the loop will end.

Algorithm 2 Synthesize a Centralized Shield

Input
  $E = (Q, Q_0, \mathbb{A}, \delta)$ // Environment
  $Q_{bad} : 2^Q$ // A set of prohibited states
Output
  $\mathsf{CS} : Q \to 2^{\mathbb{A}}$ // Centralized Shield
procedure SynthesizeCShield(E, Q_{bad})
  Initialize $Q_{Unsafe} := Q_{bad}$ // Bad states, plus any states which inevitably lead to them
  repeat
    $\mathsf{CS}(q) := \{a \in \mathbb{A} | \nexists q' \in \delta(q, a), q' \in Q_{Unsafe}\}$ for all $q \in Q$
    $Q_{Unsafe} := Q_{Unsafe} \cup \{q \in Q | \mathsf{CS}(q) = \emptyset\}$
  until Q_{Unsafe} is stable
  if $\exists q \in Q_0, \mathsf{CS}(q) = \emptyset$ **then**
    return fail
  end if
  return CS
end procedure

B.2 Shield Decentralization Algorithms

Algorithm 3 describes the shared parts between the naive and SAT-based decentralization methods. In practice, an implementer of the naive algorithm will simply calculate a single state-decentralization arbitrarily for each state, rather than calculating all state-decentralizations and *then* choosing one arbitrarily, but the two procedures produce identical results.

Algorithm 3 Decompose a Centralized Shield

1: **Input**
2: $E = (Q, Q_0, \mathbb{A}, \delta)$ // Environment
3: $\mathbb{D} = (k, \mathbb{A}_1, \ldots, \mathbb{A}_k, \Omega_1, \ldots, \Omega_k, \mathsf{obs}_1, \ldots, \mathsf{obs}_k)$ // Decentralization Setup
4: $\mathsf{CS} : Q \to 2^{\mathbb{A}}$ // Centralized Shield
5: **Output**
6: $\mathsf{DS} = (\mathsf{DS}_1, \ldots, \mathsf{DS}_k)$ // Decentralized Shield
7: **procedure** DecomposeCShield($E, \mathbb{D}, \mathsf{CS}$)
8: **for** $q \in Q$ **do**
9: $MPD_{\mathsf{CS}}(q) :=$ CalcMPDs($\mathsf{CS}(q)$)
10: **end for**
11: $A :=$ ChooseDecentralizations($E, \mathbb{D}, MPD_{\mathsf{CS}}$) // Several methods; see Algorithms 4, 5
12: **for** $i \in [1..k]$ **do**
13: $\forall o_i \in \Omega_i, \mathsf{DS}_i(o_i) = \mathbb{A}_i$
14: **for** $q \in Q, o_i \in \mathsf{obs}_i(q)$ **do**
15: $\mathsf{DS}_i(o_i) := \mathsf{DS}_i(o_i) \cap A_i^q$
16: **if** $\mathsf{DS}_i(o_i) = \emptyset$ **then**
17: **return fail**
18: **end if**
19: **end for**
20: **end for**
21: **return** $(\mathsf{DS}_1, \ldots, \mathsf{DS}_k)$
22: **end procedure**

Algorithm 4 is the *naive* method for decomposition; this may lead to a set of state-decentralizations that fail on Algorithm 3, Line 17.

Algorithm 4 Naively choose state-decentralization

1: **Input**
2: $E = (Q, Q_0, \mathbb{A}, \delta)$ // Environment
3: $\mathbb{D} = (k, \mathbb{A}_1, \ldots, \mathbb{A}_k, \Omega_1, \ldots, \Omega_k, \mathsf{obs}_1, \ldots, \mathsf{obs}_k)$ // Decentralization Setup
4: $MPD_{\mathsf{CS}} : Q \to 2^{(2^{\mathbb{A}_1}, \ldots, 2^{\mathbb{A}_k})}$ // Maximally Permissive State-Decentralizations
5: **Output**
6: $A^q = (A_1^q \subseteq \mathbb{A}_1, \ldots, A_k^q \subseteq \mathbb{A}_k), \forall q \in Q$ // Chosen State-Decentralizations
7: **procedure** ChooseDecentralizationsNaive($E, \mathbb{D}, MPD_{\mathsf{CS}}$)
8: **for** $q \in Q$ **do**
9: $A^q = (A_1^q, \ldots, A_k^q) :=$ Choose an arbitrary member of $MPD_{\mathsf{CS}}(q)$
10: **end for**
11: **return** $A^q, \forall q \in Q$
12: **end procedure**

In contrast, if Algorithm 5 (the *SAT-based* method) succeeds at finding a set of state-decentralizations, shield decomposition will succeed when using this set.

Algorithm 5 Choose state-decentralizations using a SAT solver

1: **Input**
2: $E = (Q, Q_0, \mathbb{A}, \delta)$ // Environment
3: $\mathbb{D} = (k, \mathbb{A}_1, \ldots, \mathbb{A}_k, \Omega_1, \ldots, \Omega_k, \mathsf{obs}_1, \ldots, \mathsf{obs}_k)$ // Decentralization Setup
4: $MPD_{\mathsf{CS}} : Q \to 2^{(2^{\mathbb{A}_1}, \ldots, 2^{\mathbb{A}_k})}$ // Maximally Permissive State-Decentralizations
5: **Output**
6: $A^q = (A^q_1 \subseteq \mathbb{A}_1, \ldots, A^q_k \subseteq \mathbb{A}_k), \forall q \in Q$ // Chosen State-Decentralizations
7: **procedure** ChooseDecentralizationsSAT$(E, \mathbb{D}, MPD_{\mathsf{CS}})$
8: $C := \{\}$ // A set of boolean constraints to be given to the SAT solver
9: $\forall i \in [1..k], o_i \in \Omega_i, a_i \in \mathbb{A}_i, ActionEnabled_{i,o_i,a_i} : \mathbb{B}$ // Var representing $a_i \in \bigcap_{q \in R_i(o_i)} A^q_i$
10: $\forall q \in Q, d^q \in MPD_{\mathsf{CS}}(q), DecompSelected_{q,d^q} : \mathbb{B}$ // Var representing $A^q = d^q$
11: **for** $i \in [1..k], o_i \in \Omega_i$ **do**
12: $C := C \cup \{\vee_{a_i \in \mathbb{A}_i} ActionEnabled_{i,o_i,a_i}\}$ // Some action is always available
13: **end for**
14: **for** $q \in Q$ **do**
15: $C := C \cup \{\oplus_{d^q \in MPD_{\mathsf{CS}}(q)} DecompSelected_{q,d^q})$ // At least one decomp is chosen
16: **for** $d^q \in MPD_{\mathsf{CS}}(q), i \in [1..k], a_i \in (\mathbb{A}_i \setminus d^q_i)$ **do**
17: // If decentralization d^q is chosen for state q, the agents' available actions will be safe
18: $C := C \cup \{DecompSelected_{q,d^q} \to \neg ActionEnabled_{i,\mathsf{obs}_i(q),a_i}\}$
19: **end for**
20: **end for**
21: $Sol :=$ SATSolver(C)
22: **if** $Sol = UNSAT$ **then**
23: **return fail**
24: **else**
25: **for** $q \in Q$ **do**
26: **for** $d^q \in MPD_{\mathsf{CS}}(q)$ **do**
27: **if** $Sol(DecompSelected_{q,d^q}) = \top$ **then** // True for one d_q (line 15)
28: $A^q := d^q$
29: **end if**
30: **end for**
31: **end for**
32: **return** $A^q, \forall q \in Q$
33: **end if**
34: **end procedure**

For each environment state, the centralized shield specifies a safe set of joint actions; for example, $\{(a_1^1, a_1^2), (a_1^1, a_2^2), (a_2^1, a_1^2)\}$. The goal of Algorithm 6 is to find all maximally-permissive state-decentralizations. In this example, there are exactly two: $(\{a_1^1, a_2^1\}, \{a_1^2\})$ and $(\{a_1^1\}, \{a_1^2, a_2^2\})$. Both state-decentralizations have the property that the Cartesian product of their components is a subset of the safe set of joint actions, and no individual actions can be added while maintaining that property.

The classic method to find a single maximally-permissive state-decentralization (Melcer et al., 2022) starts with one known-safe joint action, and adds all individual actions that it can, while maintaining the safety property. While this works well to find a single maximally permissive state-decentralization, it cannot be used to obtain all state-decentralizations for a given state.

In contrast, the following algorithm starts with a state-decentralization that allows every joint action. It iterates through unsafe joint actions. If a given state-decentralization $d = (A_1^q, A_2^q, \ldots)$ allows an unsafe joint action $(a_1, a_2, \ldots)$, the algorithm splits d into several state-decentralizations $\{(A_1 \setminus a_1, A_2, \ldots), (A_1, A_2 \setminus a_2, \ldots), \ldots\})$ which are each now safe.

Intuitively, if we let $\mathbb{A}^* \subseteq (\mathbb{A} \setminus \mathbb{A}_{safe})$ be the set of unsafe actions processed by the main loop of the algorithm, the algorithm maintains the invariant that D contains all maximally-permissive state-decentralizations over $\mathbb{A} \setminus \mathbb{A}^*$. When a given unsafe action a^u is processed, for any maximally-permissive state-decentralization d^* for $\mathbb{A} \setminus (\mathbb{A}^* \cup \{a^u\})$, d^* will either also be a maximally-permissive state-decentralization for $\mathbb{A} \setminus \mathbb{A}^*$, or there will exist some d which is a maximally-permissive state-decentralization for $\mathbb{A} \setminus \mathbb{A}^*$ such that $d*$ contains exactly one fewer individual action than d. This algorithm finds all possible candidate state-decentralizations d^*, and then removes all that are not maximally-permissive through the subsumption check on line 20. This process iterates until $\mathbb{A}^* = (\mathbb{A} \setminus \mathbb{A}_{safe})$, meaning that D contains all maximally-permissive state-decentralizations of $\mathbb{A}_{safe}$.

The time complexity of this algorithm is difficult to analyze traditionally; while it would seem that the size of D would grow exponentially as the algorithm proceeds, its size is significantly limited in practice by the subsumption check on line 20. This algorithm consumes a negligible portion of the overall runtime for shield decentralization.

Furthermore, the only input of this algorithm is the set of safe joint actions, plus information about the overall environment—it is not dependent on environment state. Therefore, the output of this algorithm for any one state can be reused among all states which share the same safe joint actions.

Algorithm 6 Find all possible maximally-permissive state-decentralizations

Input
  $E = (Q, Q_0, \mathbb{A}, \delta)$ // Environment
  $\mathbb{D} = (k, \mathbb{A}_1, \ldots, \mathbb{A}_k, \Omega_1, \ldots, \Omega_k, \mathsf{obs}_1, \ldots, \mathsf{obs}_k)$ // Decentralization Setup
  $\mathbb{A}_{safe} \subseteq \mathbb{A}$ // A set of safe joint actions
Output
  $D : 2^{(2^{\mathbb{A}_1}, \ldots, 2^{\mathbb{A}_k})}$ // A set of maximally-permissive state-decentralizations
procedure CalcMPDs$(E, \mathbb{D}, \mathbb{A}_{safe})$
  $D := \{\{\mathbb{A}_i | i \in [1..k]\}\}$
  for $a^u = (a^u_1, \ldots, a^u_k) \in (\mathbb{A} \setminus \mathbb{A}_{safe})$ **do**
    for $d = (d_1, \ldots, d_k) \in D$ **do**
      if $a^u \in (d_1 \times \ldots \times d_k)$ **then**
        $D := D \setminus d$
        for $i \in [1..k]$ **do** // Create state-decentralizations that omit
          if $d_i \setminus \{a^u_i\} \neq \emptyset$ **then** // a specific unsafe action for each agent
            $D := D \cup \{(d_1, \ldots, d_i \setminus \{a^u_i\}, \ldots, d_k)\}$
          end if
        end for
      end if
    end for
    Remove subsumed elements from D
  end for
  return D
end procedure

C Extended Algorithmic Example

Consider the environment shown in Figure 1: a 2-agent 3x3 gridworld, where each agent's observation radius is limited to directly adjacent and diagonal squares.

The environment $E = (Q, Q_0, \mathbb{A}, \delta)$ is defined as follows: Q is all possible states; for this environment, this is $(\mathbb{Z}^3)^4$; i.e., a tuple of four numbers representing the x and y positions of the circle and triangle from 0 to 2, inclusive. Q_0 is the set of initial states; for example, the circle and triangle start at the top-left and bottom-right respectively. $\mathbb{A}$ is the set of joint actions; i.e., all combinations of movements between the circle and triangle. Finally, δ outputs the set of possible next states, given the current state and an action. δ may have arbitrary dynamics; for example, when an agent attempts to leave the boundary, the set of next possible states may include an artificial "sink" state with a self-loop.

So far, the environment doesn't have the concept of multiple agents; it is possible for one agent to control both the circle and triangle, with full observability.

To specify the exact semantics of how agents interact with the environment, we create the decentralization setup $\mathbb{D} = (2, \mathbb{A}_1, \mathbb{A}_2, \Omega_1, \Omega_2, \mathsf{obs}_1, \mathsf{obs}_2)$. The "2" specifies that there are two agents in the environment. $\mathbb{A}_1$ and $\mathbb{A}_2$ are individual action spaces such that $\mathbb{A} = \mathbb{A}_1 \times \mathbb{A}_2$; in our example, $\mathbb{A}_1$ is the component of the action which controls the circle, while $\mathbb{A}_2$ is the component which controls the triangle. Ω_1 and Ω_2 are sets of possible individual observations, while obs_1 and obs_2 map the state space onto this set—for example, all states in which the circle is in the top-left and the triangle is far away are mapped to the same observation.

We assume that these two structures are given, along with the set of unsafe states Q_{bad}. The first step of the decentralized shield generation process is to produce a centralized shield. If there existed any states for which no safe action existed, such states would be marked as unsafe themselves. In this example, no such states exist; the centralized shield is thus very simple—a function that, given a state, returns the set of joint actions which do not immediately lead to a safety violation. The centralized shield would be more complex in environments for which there are states that aren't unsafe themselves, but will inevitably lead to a safety violation.

The next step is to calculate all sets of safe state-decentralizations for each state. This means a set of actions for each agent such that, without communication, it is safe for both agents to choose from this set. For example, when Agent 1 is in the top-left, and Agent 2 is in the top-right, it is safe for Agent 1 to go right or down as long as Agent 2 always goes down. This is not the only safe state-decentralization: it would also be safe for Agent 2 to go left or down, if Agent 1 were to always go down. The important thing is that for this state, one safe state-decentralization is chosen in advance of any training or execution. An enumeration of all safe state-decentralizations is performed by Algorithm 6.

Finally, we must choose one of these state-decentralizations for each state. Note that because multiple states map to the same observation, the safe actions allowed for a given observation is the intersection of the sets of actions allowed for that agent for the chosen state-decentralization of all states which may produce that observation. If in one state, the chosen state-decentralization only allows Agent 1 to go right or down, while in another state, the chosen state-decentralization only allows Agent 1 to go right, then if Agent 1 cannot tell apart these two states (due to them having the same observation), it must always go right. In contrast, if the actions allowed do not overlap, then Agent 1 is stuck with no possible safe action. Our challenge is thus to choose state-decentralizations for which there is always a safe action available for each agent, among all states which produce the same observation.

We encode this selection as a SAT problem. In Figure 3, the first two clauses that we add to the SAT problem relate the chosen state-decentralizations to the actions which are allowed for a given observation: the second clause states that when Agent 1 is in the top-left, and Agent 2 in the top-right, then if the first state-decentralization is selected, Agent 1 is not allowed to move right.

$$DecompSelected_{s_1,d_1^1} \Rightarrow \neg ActionEnabled_{\bullet,o_{\bullet,1},\downarrow}$$
$$DecompSelected_{s_1,d_1^2} \Rightarrow \neg ActionEnabled_{\bullet,o_{\bullet,1},\rightarrow}$$

$$DecompSelected_{s_1,d_1^1} \vee DecompSelected_{s_1,d_2^1}$$
$$DecompSelected_{s_2,d_1^2} \vee DecompSelected_{s_2,d_2^2}$$

$$ActionEnabled_{\bullet,o_{\bullet,1},\downarrow} \vee ActionEnabled_{\bullet,o_{\bullet,1},\rightarrow}$$

Figure 3: Example SAT constraints for the environment from Figure 1. When a specific state-decentralization is selected, some actions are disabled for each agent. A state-decentralization must be selected for each state. Finally, there must be an action which remains enabled for each observation. Corresponding constraints are also added for other states and observations, and for the blue triangle agent.

The third and fourth clauses state that a state-decentralization must be selected for every state; otherwise, the problem is trivially solvable by not selecting any state-decentralization, and allowing every action. Finally, the fifth clause states that Agent 1 should never be stuck with no safe action available when it observes that it is in the top-left, and does not know where Agent 2 is. This is just a small sample of the constraints that would be added to the SAT problem; similar constraints would be added for other states, observations, and agents. If SAT solver succeeds, it gives us a listing of safe individual actions that may be taken after any observation; this is the decentralized shield.

D Incorporating Bounded History in Shield Synthesis

The methods described in this paper only search for shields that do not take history into account. However, consider a domain where agents have momentum, but only observe positions, such as Particle-P. In this domain, $|R_i(o_i)|$ is large: for a given position observation, there are many states, with different velocities, which the agents are unable to distinguish between. Each state in $|R_i(o_i)|$ may have a different set of safe actions. The inclusion of just one step of observation history[1] can disambiguate the states, as there is only one velocity for a given pair of position observations, making it much more likely that a shield exists.

We accomplish this by augmenting the decentralization setup, as described in Algorithm 7. For a given history length H, for each $q \in Q, i \in [1..k]$, the modified observation functions $\mathsf{obs}'_i(q)$ return the set of possible length-$(H+1)$ observation traces for agent i at state q: H steps of historical observations, plus the current observation at q. We then simply provide this modified decentralization setup to the previous algorithm to decompose a shield. Operationally, agents must track the last H observations they encounter; they then provide this limited-horizon history to the decentralized shield to compute the set of safe actions. Our algorithm also handles the beginning of each episode, when agents have not yet encountered $H+1$ observations—the possible initial environment states must be taken into consideration for this.

Algorithm 7 Increasing history length

Input
  $E = (Q, Q_0, \mathbb{A}, \delta)$ // Environment
  $\mathbb{D} = (k, \mathbb{A}_1, \ldots, \mathbb{A}_k, \Omega_1, \ldots, \Omega_k, \mathsf{obs}_1, \ldots, \mathsf{obs}_k)$ // Decentralization setup
  $H : \mathbb{N}$ // Number of history steps in resulting decentralization setup
Output
  $\mathbb{D}' = (k, \mathbb{A}_1, \ldots, \mathbb{A}_k, \Omega'_1, \ldots, \Omega'_k, \mathsf{obs}'_1, \ldots, \mathsf{obs}'_k)$
procedure ITERATEHISTORY$(E, \mathbb{D}, H)$
  $\forall i \in [1..k], q \in Q_0, \mathit{Inits}^0_i(q) := \mathsf{obs}_i(q)$ **if** $q \in Q_0$ **else** $\emptyset$ // Possible obs when horizon $< H$
  $\forall i \in [1..k], q \in Q, \mathit{Hist}^0_i(q) := \mathsf{obs}_i(q)$ // Possible observations when horizon $\geq H$
  for $t \in [1..H]$ **do**
    $\forall i \in [1..k], q \in Q, \mathit{Inits}^t_i(q) := \mathit{Hist}^t_i(q) := \emptyset$
    for $q \in Q, q' \in \cup_{a \in \mathbb{A}} \delta(q, a)$ **do**
      for $i \in [1..k], o'_i \in \mathsf{obs}_i(q')$ **do** // "Push" history to successor states
        $\forall h_i \in \mathit{Inits}^{t-1}_i(q), \mathit{Inits}^t_i(q') := \mathit{Inits}^t_i(q') \cup \textsc{Concat}(h_i, o'_i)$
        $\forall h_i \in \mathit{Hist}^{t-1}_i(q), \mathit{Hist}^t_i(q') := \mathit{Hist}^t_i(q') \cup \textsc{Concat}(h_i, o'_i)$
      end for
    end for
  end for
  for $i \in [1..k]$ **do**
    $\Omega'_i := \Omega_i + (\Omega_i \times \Omega_i) + \ldots + (\Omega_i)^{H+1}$ // $\sum_{t=0}^{H} (\Omega_i)^{t+1}$
    $\forall q \in Q, \mathsf{obs}'_i(q) = (\bigcup_{t \in [0..(H-1)]} \mathit{Inits}^t_i(q)) \cup \mathit{Hist}^H_i(q)$
  end for
  return $(k, \mathbb{A}_1, \ldots, \mathbb{A}_k, \Omega'_1, \ldots, \Omega'_k, \mathsf{obs}'_1, \ldots, \mathsf{obs}'_k)$
end procedure

[1] It is possible to also use action history for further disambiguation, but this is often not necessary in practice.

E Considerations for Creating an Environment Abstraction

For set X, we use $\Delta(X)$ to represent a distribution over X, and $supp(\Delta(X))$ to represent the support of that distribution; i.e., values with non-zero probabilities. $\mathbb{R}$ represents the set of real numbers.

Consider a common specification of a decentralized environment, such as a Dec-POMDP (Goldman and Zilberstein, 2004). Note that there are several closely-related versions of this specification; the same principles apply to other variants.

Definition 5 (Dec-POMDP)**.** A *Dec-POMDP* is a multi-agent environment defined as a tuple $\mathcal{M} = (I = [1..k], S, S_0, (A_1, \ldots, A_k), T, R, (\mathbb{O}_1, \ldots, A_k), (\mathcal{O}_1, \ldots, \mathcal{O}_k), \gamma)$ where I is a set of agents; S is a state space; $S_0 : \Delta(S)$ is the distribution over initial states, A_i is an individual action space for each agent $i \in [1..k]$; $T : S \times A_1 \times \ldots \times A_k \to \Delta(S)$ is a transition probability function, $R : S \to \mathbb{R}$ is a reward function, $\mathbb{O}_i$ is an individual observation space for each agent $i \in [1..k]$, $\mathcal{O}_i : S \to \Delta(\mathbb{O}_i)$ is an individual observation space for each agent $i \in [1..k]$, and γ is a discount factor.

There are several ways to relate an environment abstraction to a Dec-POMDP. As a simple starting point, we say that environment $E = (Q, Q_0, \mathbb{A}, \delta)$ is a *sound* abstraction of Dec-POMDP $\mathcal{M} = (I = [1..k], S, S_0, (A_1, \ldots, A_k), T, R, (\mathbb{O}_1, \ldots, A_k), (\mathcal{O}_1, \ldots, \mathcal{O}_k), \gamma)$ if $\mathbb{A} = (A_1 \times \ldots \times A_k)$ and there exists some state abstraction function $\varsigma : S \to Q$ such that $\forall s \in S, a \in \mathbb{A}, \forall s' \in supp(T(s, a)), \forall \varsigma(s') \in \delta(\varsigma(s), a)$, and $\forall s_0 \in supp(S_0), \varsigma(s_0) \in Q_0$.

The decentralization setup $\mathbb{D} = (k, \mathbb{A}_1, \ldots, \mathbb{A}_k, \Omega_1, \ldots, \Omega_k, \mathsf{obs}_1, \ldots, \mathsf{obs}_k)$ is similarly abstracted. We require that $\forall i \in [1..k], \mathbb{A}_i = A_i$, and that there exists some observation abstraction function $\varrho : \mathbb{O}_i \to \Omega_i$ such that $\forall s \in S, \forall o_i \in \mathcal{O}_i(s), \varrho(o_i) \in \mathsf{obs}_i(\varsigma(s))$.

Finally, let $S_{Bad} \subset S$ be a set of unsafe states. The corresponding set of unsafe states in E is $Q_{bad} = \{\varsigma(s) | s \in S_{Bad}\}$. It can be seen that if one avoids all states in Q_{bad} in E, then it is trivial to also avoid states in S_{Bad} in $\mathcal{M}$.

If S is small, it is possible to create an environment abstraction more or less automatically by discarding reward information and exact transition probabilities: $E = (Q, Q_0, \mathbb{A}, \delta)$, with $Q = S$, $Q_0 : supp(S_0)$, $\mathbb{A} = A_1 \times \ldots \times A_k$, and $\delta(q, a) = supp(T(q, a))$. Similarly, if $\mathbb{O}_i$ are small for all $i \in [1..k]$, the decentralization setup $\mathbb{D} = (k, \mathbb{A}_1, \ldots, \mathbb{A}_k, \Omega_1, \ldots, \Omega_k, \mathsf{obs}_1, \ldots, \mathsf{obs}_k)$ is straightforward to create; for all $i \in [1..k]$, $\mathbb{A}_i = A_i$, $\Omega_i = \mathbb{O}_i$, and $\forall q \in Q, \mathsf{obs}_i(q) = supp(\mathcal{O}_i(q))$.

S may be large, or even infinite—in this case, abstraction is usually an art. The user must decide how much information to discard, leaving enough useful information for the shielding method to work with, without creating too many unique states and observations in the abstracted environment.

For example, consider a large Minecraft-like environment, with continuous positions and several different types of objects. If collision prevention is the only safety objective, a practitioner may discard most information about different objects in the environment, as this will usually only affect reward, not safety. It is straightforward to discretize agent positions; the primary consideration is that any given movement in a certain direction may not cause the agent to move into the next discrete space. This can be handled by ensuring that δ reflects all possible discrete positions that may be the result of a given transition.

To further reduce the state space, it is possible to use relative agent positions as the state space for shield synthesis, rather than absolute positions. Relative agent positions greater than a modest distance from each other do not need to be enumerated; the state space can just include one state to represent that the agents are distant from each other. For example, we might include one state to represent all cases in which agents are at least 10 units of distance from each other; this one state would then include nondeterministic transitions to all states in which agents are 8 or 9 units from each other.

Of course, it is possible to over-abstract an environment. If, for example, we create one abstracted state to represent all states in which agents are 2 units or more from each other, our procedure may

not be able to find a shield—it would be necessary to further refine the abstraction. Methods to automatically perform abstraction creation and refinement are an area for future work.

While this formalism is a simple way to represent an abstraction of a Dec-POMDP, it is not the only useful, or the only correct way. For example, many prior works rely on a trace equivalence property, rather than a state and observation abstraction property (Alshiekh et al., 2018; ElSayed-Aly et al., 2021). It is even possible that a state abstraction which isn't strictly correct can still be useful; if the goal is merely to reduce the number of safety violations, or to eliminate only a subclass of violations, an approximate abstraction is acceptable, and our decentralization method will still work.

F Algorithm Runtime

Our algorithms have real-world performance that is significantly different from their worst-case performance, meaning that a traditional asymptotic complexity analysis is not reflective of the realistic algorithmic runtime. For example, the SAT-based algorithm produces $\sum_{i=1}^{k}|\Omega_i|\times|\mathbb{A}_i|+\sum_{q\in Q}|MPD_{\mathsf{CS}}(q)|$ variables that are given to the solver. This would imply an obscene worst-case runtime of $O(2^{\sum_{i=1}^{k}|\Omega_i|\times|\mathbb{A}_i|+\sum_{q\in Q}|MPD_{\mathsf{CS}}(q)|})$; the algorithm would never terminate except for the most trivial of examples.

While the solver does consume a significant amount of time for more complex examples, there are a number of factors that we believe lead to a generally low runtime, aside from the general advancement in SAT solvers over the past several decades—for example, many of the clauses we add to the solver contain only one or two variables; most SAT solving algorithms handle these cases very efficiently. Similarly, a strict worst-case complexity analysis would add a factor of $\sum_{i=1}^{k}|\Omega_i|$ to many of the algorithms' runtimes. While this factor is technically possible if every state had $O(|\Omega_i|)$ possible observations for each $i \in [1..k]$ on average, this is not usually the case for real-world environments and decentralization setups.

We therefore focus on algorithm runtime, as presented in Table 3.

Table 3: Time required for the decentralization process itself, followed by total time taken in disk access, serialization, and deserialization. All times are measured in seconds, and measured on a M1 Max Macbook Pro. "X" represents that shield decentralization completes, but results in failure, for the selected environment and decentralization method. "-" indicates that a given shield decentralization method was not attempted for this environment. Note that performance optimization was done on a best-effort basis; these times are for shield decentralization, but these times are often dwarfed by the runtime of currently existing centralized synthesis tools.

Domain	Naive (0 History)	Naive (1 History)	SAT (0 History)	SAT (1 History)	SAT (2 History)
Grid-Col	0.3, 0.6	26.9, 2.1	0.8, 1.1	63.6, 2.0	-
Particle-P	X	1.3, 0.1	X	2.4, 0.1	-
Particle-P-V	0.1, 0.1	1.6, 0.1	0.1, 0.1	2.9, 0.1	-
Nearby-Obs	0.3, 0.6	23.8, 0.7	0.6, 0.6	55.5, 0.7	-
Flashlight-6	X	X	<0.1, <0.1	3.0, 0.1	273, 2.2
Flashlight-10	X	X	0.6, 0.4	52.9, 0.7	-

We can also compare the runtimes to the number of variables that are produced during the SAT-based decentralization process. There are two families of variables, we track numbers of each of these separately. The totals are listed in Table 4.

Table 4: Number of SAT variables produced by our method

Environment	$\|Q\|$	History Length	$a \in \mathsf{DS}_i(o_i)$ $(\sum_{i\in[1..k]}\|\Omega_i\|\times\|\mathbb{A}_i\|)$	$A^q = d^q$ $(\sum_{q\in Q}\|MPD_{\mathsf{CS}}(q)\|)$	Total Variables
Grid-Col	52670	0	526700	55384	582084
		1	12128840	1284224	13413064
Particle-P-V	7824	0	78240	11180	89420
		1	645360	93832	739192
Particle-P	7824	0	3600	11180	14780
		1	81840	11796	93636
Flashlight (6x6)	2400	0	11250	3856	15106
		1	388970	214020	602990
		2	13628090	12213716	25841806
Flashlight (10x10)	39600	0	80800	46496	127296
		1	2904920	2858920	5763840

G Additional Training Details

We replicate the hyperparameters from Melcer et al. (2022) as closely as possible. In Grid-Col, we use tabular individual Q-learning with $\gamma = 0.9$. We use ϵ-greedy action selection with ϵ annealed from 1 to 0 over 2.5 million time steps during training, and $\epsilon = 0$ during evaluation. We use 50 random seeds. Note that our methodology diverges from ElSayed-Aly et al. (2021); Melcer et al. (2022) by using $\epsilon = 0$ during evaluation (as opposed to $\epsilon = 0.05$) to be fairer to unshielded agents—if a policy had learned to avoid a specific action, it may have been forced to take it anyways during evaluation, while shielded agents were still able to block such an action. Additionally, we use discounted rewards as the evaluation metric, rather than undiscounted rewards.

For Particle-P and Particle-P-V, we use Deep Double Q-learning (Mnih et al., 2013; Van Hasselt et al., 2016) with a simple network architecture. The agent's network consists of a linear layer from the input space (one-hot encoding of agent positions) to 128 features, a ReLU activation (Glorot et al., 2011), then additional linear layers of size 128, 64, and 5 (the action space size), all with ReLU activation except for the last layer. We set the replay buffer size to be 10^5, and optimize the network using default parameters of the PyTorch Adam optimizer, with a batch size of 32 (Kingma and Ba, 2014; Paszke et al., 2019). Otherwise, all hyperparameters are the same as in Grid-Col. We use 50 seeds as well for these runs.

In Nearby-Obs, we modify the agents to use a DRQN architecture. The neural network is similar to the network used for Particle, except that between the two linear layers of size 128, we insert a GRU layer of size 128. We train with a sequence length of 4 and a batch size of 8; otherwise, all hyperparameters are the same as with Particle-P and Particle-P-V; we did not perform any architecture or hyperparameter search to arrive at these values. We evaluate each configuration with 50 random seeds.

Flashlight-6 and Flashlight-10 use the same agent architecture and hyperparameters as Nearby-Obs, but with 10 random seeds.

Note that we introduce seeded randomness during the shield decentralization process so that the generated shields are diverse over different random seeds of the same experiment. The naive shield generation method often fails when this randomness is introduced, even in cases where it succeeds without randomness; if this occurs, we substitute a non-random shield generated with the naive algorithm.

H Reinforcement Learning Results

We include full results for all environments. Table 5 shows results from Grid-Col for each method. Table 6 shows results from Particle-P using a history length of one, and Particle-P-V without history. Table 7 shows results for Nearby-Obs for each method; note that the method in Melcer et al. (2022) cannot produce a shield for this method due to its partial observability. Tables 8 and 9 show results for Flashlight-6 and Flashlight-10, respectively. Note that our SAT-based decomposition is the only decentralized method that succeeds in this environment.

Aside from the results described here, shielded agents avoid taking unsafe actions during training, compared to the hundreds of thousands of safety violations that unshielded agents take during training. While many of these violations during training are undoubtedly attributable to high ϵ values during ϵ-greedy action selection, we note that shielding allows a RL practitioner to be more flexible with their selection of training hyperparameters, without needing to worry about the hyperparameters' effect on safety violations.

Table 5: Results in Grid-Col; average discounted returns and standard error over 50 seeds, and average total safety violations over 100 evaluation runs in parentheses (omitted for configurations with no violations). History was not used.

Start	Map	Melcer et al. (2022)	Naive	SAT	Centralized	No Shield
Fixed	ISR	38.4 ± 0.6	38.9 ± 0.6	38.1 ± 0.6	38.4 ± 0.7	39.9 ± 0.9 (4.0)
	MIT	2.2 ± 0.1	2.0 ± 0.1	1.8 ± 0.1	2.1 ± 0.2	1.8 ± 0.1
	Pentagon	31.1 ± 0.5	30.7 ± 0.6	32.6 ± 0.5	31.9 ± 0.6	35.4 ± 0.7
	SUNY	24.4 ± 0.4	23.4 ± 0.4	24.0 ± 0.4	22.6 ± 0.3	23.1 ± 0.4
Rand	ISR	29.4 ± 2.0	29.5 ± 2.4	30.1 ± 2.2	31.1 ± 2.5	28.1 ± 1.9 (0.9)
	MIT	15.1 ± 1.6	19.7 ± 2.4	18.0 ± 2.2	19.1 ± 2.3	15.7 ± 1.8
	Pentagon	32.4 ± 1.9	37.2 ± 2.2	34.1 ± 2.2	39.6 ± 2.8	31.8 ± 2.1 (0.3)
	SUNY	11.9 ± 2.5	8.0 ± 1.9	14.8 ± 2.4	10.6 ± 2.4	11.6 ± 2.0

Table 6: Results in Grid-Col; average discounted returns and standard error over 50 seeds, and average total safety violations over 100 evaluation runs in parentheses (omitted for configurations with no violations). Particle-P uses 1 step of history, Particle-P-V uses no history.

Environment	Start	Melcer et al. (2022)	Naive	SAT	Centralized	No Shield
Particle-P	Fixed	33.6 ± 0.0	33.6 ± 0.0	33.6 ± 0.0	33.1 ± 0.5	33.6 ± 0.0
	Rand	26.0 ± 3.2	28.8 ± 2.0	27.6 ± 3.1	29.7 ± 3.8	17.0 ± 3.5 (69.2)
Particle-P-V	Fixed	33.6 ± 0.0	33.6 ± 0.0	33.6 ± 0.0	33.6 ± 0.0	33.6 ± 0.0
	Rand	51.3 ± 2.1	53.0 ± 2.4	50.0 ± 1.8	51.7 ± 2.0	48.7 ± 2.0 (0.1)

Table 7: Results in Nearby-Obs, average discounted returns and standard error over 50 random seeds, and average total safety violations over 100 runs in parentheses (omitted for configurations with no violations). History was not used.

Start Type	Map	Naive	SAT	Centralized	No Shield
Fixed	ISR	80.9 ± 0.2	78.6 ± 2.6	83.9 ± 0.0	81.2 ± 2.7
	MIT	-49.0 ± 0.0	-49.0 ± 0.0	-49.0 ± 0.0	-47.0 ± 2.0
	Pentagon	-2.4 ± 8.5	48.5 ± 7.4	35.2 ± 8.6	-46.5 ± 11.8 (1482.0)
	SUNY	2.4 ± 8.6	-0.6 ± 8.5	2.5 ± 8.6	14.7 ± 8.7
Random	ISR	61.1 ± 5.7	45.0 ± 7.3	61.8 ± 5.4	5.6 ± 8.8 (113.4)
	MIT	-42.4 ± 3.7	-42.2 ± 3.9	-44.2 ± 3.4	-43.5 ± 6.6 (159.8)
	Pentagon	-31.3 ± 6.1	-25.1 ± 9.3	-6.0 ± 9.2	-42.0 ± 4.3 (56.9)
	SUNY	-49.0 ± 0.0	-49.0 ± 0.0	-49.0 ± 0.0	-49.0 ± 0.0 (146.4)

Table 8: Flashlight-6 results; average discounted returns and standard error over 10 random seeds for varying recharge times (RT), and average sum of safety violations over 100 testing episodes in parentheses (omitted for configurations with no violations).

Start	RT	SAT (0 History)	SAT (1 History)	SAT (2 History)	Centralized	No Shield
Fixed	3	65.7 ± 1.5	69.2 ± 1.5	71.1 ± 0.8	78.6 ± 0.0	78.6 ± 0.0
	4	58.7 ± 1.5	66.5 ± 1.9	69.1 ± 1.7	78.1 ± 0.5	78.6 ± 0.0
	5	-41.6 ± 60.5	52.0 ± 11.6	65.1 ± 2.5	78.6 ± 0.0	78.6 ± 0.0
	6	-40.6 ± 41.9	16.4 ± 13.9	50.5 ± 12.3	78.6 ± 0.0	77.7 ± 0.9 (10.0)
Rand	3	65.4 ± 1.1	74.9 ± 0.3	74.4 ± 0.4	84.7 ± 0.3	83.4 ± 0.2 (7.2)
	4	53.1 ± 1.2	68.5 ± 0.4	72.0 ± 0.5	83.7 ± 0.2	82.7 ± 0.7 (5.1)
	5	-20.6 ± 14.0	56.0 ± 3.7	67.1 ± 0.7	81.6 ± 0.9	83.5 ± 0.3 (5.2)
	6	-23.8 ± 15.8	30.0 ± 12.7	62.9 ± 1.7	76.9 ± 7.0	83.5 ± 0.3 (5.5)

Table 9: Flashlight-10 results; average reward and standard error over 10 random seeds for varying recharge times (RT), and average sum of safety violations over 100 testing episodes in parentheses (omitted for configurations with no violations).

Start Type	RT	SAT (0 History)	SAT (1 History)	Centralized	No Shield
Fixed	2	-49.0 ± 0.0	45.7 ± 0.9	55.3 ± 0.0	54.4 ± 0.8 (10.0)
	3	-49.0 ± 0.0	-34.9 ± 10.5	55.3 ± 0.0	54.4 ± 0.8 (10.0)
	4	-49.8 ± 0.8	-52.5 ± 1.4	55.3 ± 0.0	55.3 ± 0.0
Random	2	-52.4 ± 0.3	57.2 ± 2.4	16.9 ± 14.3	43.2 ± 11.4 (6.9)
	3	-109.6 ± 7.4	38.1 ± 3.6	14.5 ± 17.3	18.3 ± 20.3 (9.3)
	4	-171.7 ± 12.8	-55.1 ± 0.3	38.9 ± 8.0	49.8 ± 11.2 (6.4)

I Comparison of Shielding Methods

We include Table 10 to directly compare the domains of several shielding methods in terms of observability, and the number and types of agents.

Method	Obs	Agents	Brief Description
Alshiekh et al. (2018)	Full	Single	First to apply Shielding to RL
Mazzi et al. (2021)	Partial	Single	Enforces safety in context of POMCP algorithm
Carr et al. (2022)	Partial	Single	Uses belief support to handle partial observability
Junges et al. (2021)	Partial	Single	Another belief support method
ElSayed-Aly et al. (2021)	Full	Communicate	Local centralized shields allow for scalability
Melcer et al. (2022)	Full	Cooperative	Avoids communication by decentralizing safe action sets
Carr et al. (2021)	Full	Adversarial	Models adversary as part of nondeterministic env
Ours	Partial	Cooperative	Decentralizes both observations and actions

Table 10: Comparison of the shielding methods mentioned in Section 2; in particular, which domains each method operates in. Note that "Full" observability specifically references observability of safety-relevant, rather than reward-relevant, information. Unless specifically noted, cooperative methods do not use communication. As all shielding methods operate on an abstraction of the environment, information irrelevant for safety may be partially observable.

J Proof Sketches

J.1 Decentralized Shields that Satisfy Problem 1 are Deadlock-Free

Proof Sketch $\forall i \in [1..k], q \in Q, \mathsf{obs}_i(q) \neq \emptyset$, and $\mathsf{DS}_i(o_i)$ is nonempty for every $o_i \in \mathsf{obs}_i(q)$ when q is reachable, so it follows that $\bigcup_{o_i \in \mathsf{obs}_i(q)} \mathsf{DS}_i(o_i)$ is also nonempty. Because this is true for every i, it holds that $(\bigcup_{o_1 \in \mathsf{obs}_1(q)} \mathsf{DS}_1(o_1)) \times \ldots \times (\bigcup_{o_k \in \mathsf{obs}_k(q)} \mathsf{DS}_k(o_k))$ is also nonempty. Because the application of a decentralized shield to the environment preserves legality for these actions, there is at least one legal action for every reachable state of $\mathsf{DS}(E)$, and it is thus deadlock-free. □

J.2 Problems 2 and 3 solve Problem 1

Since for all reachable $q, \forall o_1 \in \mathsf{obs}_1(q), \ldots, \forall o_k \in \mathsf{obs}_k(q), \mathsf{DS}_1(o_1) \times \ldots \times \mathsf{DS}_k(o_k)$ is a non-empty subset of $\mathsf{CS}(q)$, and since $\mathsf{CS}(q)$ is a subset of the legal actions at q, it follows that $\mathsf{DS}_1(o_1) \times \ldots \times \mathsf{DS}_k(o_k)$ is also a subset of the legal actions at q. Additionally, $(\bigcup_{o_1 \in \mathsf{obs}_1(q)} \mathsf{DS}_1(o_1)) \times \ldots \times (\bigcup_{o_k \in \mathsf{obs}_k(q)} \mathsf{DS}_k(o_k)) \subseteq \mathsf{CS}(q)$; therefore, the legal actions at q for $\mathsf{DS}(E)$ are a subset of the legal actions at q for $\mathsf{CS}(E)$. Thus, all states which are unreachable in $\mathsf{CS}(E)$, such as those in Q_{bad}, are also unreachable in $\mathsf{DS}(E)$. □