Safe Planning via Model Predictive Shielding

Osbert Bastani
University of Pennsylvania, USA

Abstract

Reinforcement learning is a promising approach to synthesizing policies for robotics tasks. A key challenge is ensuring safety of the learned policy—e.g., that a walking robot does not fall over, or an autonomous car does not run into an obstacle. We focus on the setting where the dynamics are known, and the goal is to prove that a policy trained in simulation satisfies a given safety constraint. We build on an approach called shielding, which uses a backup policy to override the learned policy as needed to ensure safety. Our algorithm, called model predictive shielding (MPS), computes whether it is safe to use the learned policy on-the-fly instead of ahead-of-time. By doing so, our approach is computationally efficient, and can furthermore be used to ensure safety even in novel environments. Finally, we empirically demonstrate the benefits of our approach.

1 Introduction

Reinforcement learning is a promising approach for synthesizing control policies for accomplishing robotics tasks—e.g., it can be used to automatically synthesize policies for challenging planning and control problems (Andrychowicz et al., 2018; Khan et al., 2019), or to compress a computationally expensive search-based planner or optimal controller into a neural network policy that is computationally efficient in comparison (Levine and Koltun, 2013).

A key challenge for using learned policies on real robots is how to ensure safety—e.g., that a autonomous car does not drive into an obstacle, a walking robot does not fall, or a quadcopter does not crash.

Preliminary work. Under review by AISTATS 2020. Do not distribute.

We consider the planning setting, where a policy is learned in simulation, and the goal is to deploy it to control a real robot. We assume that the robot dynamics are known and deterministic. Furthermore, we assume that while the environment may not be known ahead-of-time, perception is accurate, so we know the positions of the obstacles when executing the policy. As a concrete example, consider an autonomous car. We have very good models of car dynamics, and we have good sensors for detecting obstacles. However, we may want the car to drive in many different environments, with different configurations of obstacles (e.g., walls, buildings, and trees). Given a learned policy, our goal is to ensure that the policy does not cause an accident when driving in a novel environment.

One approach to guaranteeing safety is to rely on ahead-of-time verification—i.e., prove ahead-of-time that the learned policy is safe, and then deploy the learned policy on the robot (Verma et al., 2018; Bastani et al., 2018; Ivanov et al., 2019). An alternative approach, called *shielding*, is to synthesize a backup policy and prove that it is safe, and then use the backup policy to override the learned policy as needed to guarantee safety (Perkins and Barto, 2002; Gillula and Tomlin, 2012; Akametalu et al., 2014; Chow et al., 2018; Alshiekh et al., 2018; Zhu et al., 2019). [1]

A shortcoming of existing approaches is that ahead-of-time verification can be computationally intractable for high-dimensional state spaces. However, to handle novel environments, we must encode the environment in the state, which can quickly increase the dimension of the state space. Another shortcoming is that to prove safety for an infinite horizon, existing approaches typically rely on proving that the robot remains in a bounded region of the state space that is known to be safe. However, we often want robots that engage in dynamic behaviors—e.g., driving across a potentially unbounded region of the state space.

We propose an approach to safe reinforcement learning that ensures safety on-the-fly. Our approach, called *model predictive shielding (MPS)*, is based on the con-

[1] The term "shielding" was recently introduced in Alshiekh et al. (2018).

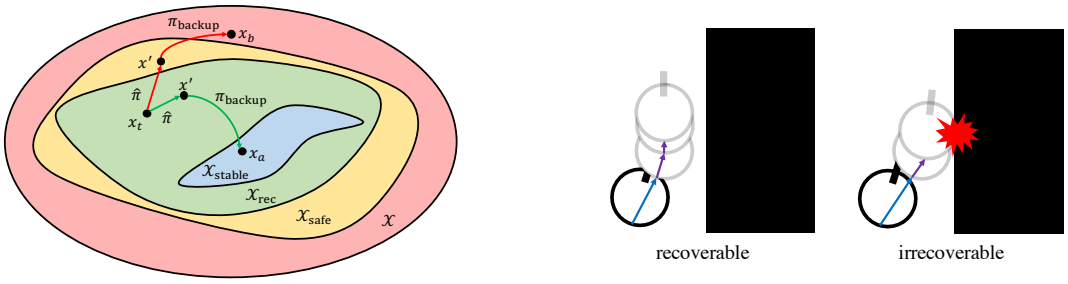

Figure 1: Left: An overview of MPS. At state x_t, MPS simulates the learned policy $\hat{\pi}$ for one step, and then simulate the backup policy π_{backup} for N steps to see if it can drive the robot from $\mathcal{X}_{\text{rec}}$ to $\mathcal{X}_{\text{stable}}$. If so (green trajectory to x_a), it uses $\hat{\pi}$; otherwise (red trajectory to x_b), it uses π_{backup}. Right: Example of recoverable and irrecoverable states for an autonomous car, where the goal is to avoid the black obstacle. We show the simulated trajectory used to check recoverability; the simulation of $\hat{\pi}$ is blue, and the simulation of π_{backup} is purple.

cept of shielding. In contrast to previous approaches, our algorithm determines whether to use the learned policy or the backup policy on-the-fly. At a high level, MPS maintains the invariant that the backup policy can always recover the robot, and only uses the learned policy if it can prove that doing so maintains this invariant. Unlike previous approaches, it checks whether this invariant holds on-the-fly by simulating the dynamics. Intuitively, checking the invariant for just the current state is far more efficient than verifying ahead-of-time that safety holds for all initial states.

While our approach incurs runtime overhead during computation of the policy, each computation is efficient. In contrast, ahead-of-time verification can take exponential time; even though this computation is offline, it can be infeasible for problems of interest.

A remaining challenge is how to construct the backup policy. We propose an approach that decomposes it into two parts: (i) are *stable policy*, which stabilizes the robot near a safe equilibrium point, [2] and (ii) a *recovery policy*, which tries to drive the robot to a safe equilibrium point. The recovery policy can be arbitrary (e.g., trained using reinforcement learning). Additionally, for unstable equilibrium, we propose to use feedbackup control to stabilize the robot. An overview of our approach is shown in Figure 1.

We evaluate our approach on the cart-pole trained to move to the right while avoiding falling, and a bicycle with random obstacles. Furthermore, we demonstrate how our approach outperforms approaches based on ahead-of-time verification.

Example. Consider a walking robot, where the goal is to have the robot run as fast as possible without falling over. The learned policy may perform well at this task, but cannot guarantee safety. We consider equilibrium points where the robot is standing upright at rest. Then, the stable policy stabilizes the robot at these points, and the recovery policy is trained to bring the running robot to a stop. Finally, the MPS algorithm uses the learned policy to run, while maintaining the invariant that the recovery policy can always safely bring the robot to a stop, after which the stable policy can ensure safety for an infinite horizon.

Alternatively, consider an autonomous car—the equilibrium points are states where the car is at rest; the recovery policy could be to slam the brakes, and once the car is at rest, the stable policy could be to keep the car is at rest by continuing to hold the brakes.

A key feature of our approach is that it naturally switches between the learned and backup policies. For example, suppose our algorithm uses the recovery policy to slow down the robot. The robot does not have to come to a stop; instead, our algorithm switches back to the learned policy as soon as it is safe to do so.

Contributions. In summary, our contributions are: we propose a new algorithm for ensuring safety of a learned control policy (Section 2), we propose an approach for constructing a backup policy in this setting (Section 3), along with an extension to handle unstable equilibrium points (Section 4), and we empirically demonstrate the benefits of our approach compared to ones based on ahead-of-time verification (Section 5).

Related work. There has been much recent interest in safe reinforcement learning (Garcıa and Fernández, 2015; Amodei et al., 2016). One approach is to use constrained reinforcement learning to learn policies that satisfy a safety constraint (Achiam et al., 2017; Wen and Topcu, 2018). However, these approaches typically do not guarantee safety.

Existing approaches that guarantee safety typically

[2] The term "stable" refers to how we use feedback stabilization to construct π_{backup}; see Section 4 for details.

rely on proving ahead-of-time that the safety property $\phi_{\text{safe}} = \bigwedge_{x_0 \in \mathcal{X}_0} \bigwedge_{t=0}^{\infty} x_t \in \mathcal{X}_{\text{safe}}$, $x_0 \in \mathcal{X}_0$ are the initial states, $x_{t+1} = f(x_t, \pi(x_t))$ for all $t \geq 0$, and $\mathcal{X}_{\text{safe}}$ are the safe states. One approach is to directly verify that the learned policy is safe (Berkenkamp et al., 2017; Verma et al., 2018; Bastani et al., 2018; Ivanov et al., 2019). However, verification does not give a way to repair the learned policy if it turns out to be unsafe.

An alternative approach, called *shielding*, is use ahead-of-time verification to prove safety for a *backup policy*, and then combine the learned policy with the backup policy in a way that is guaranteed to be safe (Perkins and Barto, 2002; Gillula and Tomlin, 2012; Akametalu et al., 2014; Chow et al., 2018; Alshiekh et al., 2018; Zhu et al., 2019). [3] This approach can improve scalability since the backup policy is often simpler than the learned policy. For example, the backup policy may bring the to a stop if it goes near an obstacle. This approach implicitly verifies safety of the joint policy (i.e., the combination of the learned policy and the backup policy) ahead-of-time.

These existing approaches have two limitations. First, ahead-of-time verification can be computationally infeasible—it requires checking whether safety holds from every state, which can scale exponentially in the state space dimension. Many existing approaches only scale to a few dimensions (Gillula and Tomlin, 2012; Berkenkamp et al., 2017). One solution is to overapproximate the dynamics (Asselborn et al., 2013; Koller et al., 2018). However, for nonlinear dynamics, the approximation error quickly compounds, causing verification to fail even when safety holds.

Scalability is particularly challenging when we want to handle the possibility of novel environments. One way to handle novel environments is to run verification from scratch every time a novel environment is encountered; however, doing so online would be computationally expensive. Our approach to handling novel environments is to encode the environment into the state. However, this approach quickly increases the dimension of the state space, resulting in poor scalability for existing approaches since they rely on ahead-of-time verification. Instead, these approaches typically focus on verifying a property of the robot dynamics in isolation of its environment (e.g., positions of obstacles) or with respect to a fixed environment.

Second, many existing approaches have trouble verifying safety over an infinite time horizon when the desired behavior is "dynamic". One approach is to verify safety over a finite time horizon (Bastani et al., 2018; Chow et al., 2018), but then the robot may not be safe for all time. To prove safety for an infinite time horizon, the typical approach is to try and prove that the robot remains within some constrained region of the state space for an infinite horizon (Gillula and Tomlin, 2012; Akametalu et al., 2014; Berkenkamp et al., 2017). Intuitively, the idea is to compute a set of states $\mathcal{G}$ that is *invariant* under the dynamics—roughly speaking, $f^{(\pi)}(\mathcal{G}) \subseteq \mathcal{G}$, where $f^{(\pi)}$ are the closed-loop dynamics for policy π. Then, as long as the robot starts from $\mathcal{G}$, it would stay in $\mathcal{G}$ forever. Assuming $\mathcal{G} \subseteq \mathcal{X}_{\text{safe}}$, then safety is guaranteed for an infinite horizon.

However, we often want robots that engage in dynamic behaviors such as walking, flying, or driving across a possibly unbounded region of the state space. Approaches relying on invariant sets cannot guarantee safety in these settings, since $\mathcal{G}$ must typically be a bounded set. One solution is to have the user provide an infinite horizon *reference trajectory* that is known to be safe, and then prove that the robot stays near the reference trajectory (Tedrake, 2018). However, the reference trajectory must be manually provided by the user, and is domain specific—e.g., a limit cycle for a walking robot (Kuindersma et al., 2016).

Finally, while we focus on planning, where the dynamics are known, there has also been work on safe exploration, which aims to ensure safety while learning the dynamics (Moldovan and Abbeel, 2012; Gillula and Tomlin, 2012; Akametalu et al., 2014; Turchetta et al., 2016; Wu et al., 2016; Berkenkamp et al., 2017; Dean et al., 2018). These approaches rely on verification, so we believe our approach can benefit them as well.

2 Model Predictive Shielding

Given an arbitrary *learned policy* $\hat{\pi}$ (designed to minimize a loss function), our goal is to minimally modify $\hat{\pi}$ to obtain a safe policy π_{shield} for which safety is guaranteed to hold. At a high level, our algorithm ensures safety by combining $\hat{\pi}$ with a *backup policy* π_{backup} (guaranteed to ensure safety on a subset of states).

As a running example, for the cart-pole, $\hat{\pi}$ may be learned using reinforcement learning to move the cart as quickly as possible to the right, but cannot guarantee that the desired safety property that pole does not fall over. In contrast, π_{backup} may try to stabilize the pole in place, but does not move the cart to the right.

In general, *shielding* is an approach to safety based on constructing a policy π_{shield} that chooses between using $\hat{\pi}$ and using π_{backup}. Our shielding algorithm, called *model predictive shielding (MPS)*, maintains the invariant that π_{backup} can be used to ensure safety. In particular, given a state x, we simulate the dynamics to determine the state x' reached by using $\hat{\pi}$ at

[3] More generally, the shield can simply constrain the set of allowed actions in a way that ensures safety.

x, and then further simulate the dynamics to determine whether π_{backup} can ensure safety from x' using π_{backup}. For now, we describe our approach assuming π_{backup} is given; in Sections 3 & 4, we describe approaches for constructing π_{backup}.

Preliminaries. We consider deterministic, discrete time dynamics $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ with states $\mathcal{X} \subseteq \mathbb{R}^{n_X}$ and actions $\mathcal{U} \subseteq \mathbb{R}^{n_U}$. Given a control policy $\pi : \mathcal{X} \to \mathcal{U}$, $f^{(\pi)}(x) = f(x, \pi(x))$ denotes the closed-loop dynamics. The *trajectory* generated by π from an initial state $x_0 \in \mathcal{X}$ is the infinite sequence of states $x_0, x_1, ...$, where $x_{t+1} = f^{(\pi)}(x_t)$ for all $t \geq 0$.

Shielding problem. We have two goals: (i) given loss $\ell : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$, initial states $\mathcal{X}_0 \subseteq \mathcal{X}$, and initial state distribution d_0 over $\mathcal{X}_0$, minimize

$$L(\pi) = \mathbb{E}_{x_0 \sim d_0}\left[\sum_{t=0}^{T-1} \ell(x_t, u_t)\right],$$

where $x_{t+1} = f(x_t, u_t)$, $u_t = \pi(x_t)$, and $T \in \mathbb{N}$ is a finite time horizon, and (ii) given safe states $\mathcal{X}_{\text{safe}} \subseteq \mathcal{X}$, ensure that the trajectory $x_0, x_1, ...$ generated by π from any $x_0 \in \mathcal{X}_0$ is *safe*—i.e., $x_t \in \mathcal{X}$ for all $t \geq 0$.

To achieve these goals, we assume given two policies: (i) a *learned policy* $\hat{\pi}$ trained to minimize $L(\pi)$, and (ii) a *backup policy* π_{backup}, together with *stable states* $\mathcal{X}_{\text{stable}} \subseteq \mathcal{X}$, such that the trajectory generated by π_{backup} from any $x_0 \in \mathcal{X}_{\text{stable}}$ is guaranteed to be safe. We make no assumptions about $\hat{\pi}$; e.g., it can be a neural network policy trained using reinforcement learning. In contrast, π_{backup} cannot be arbitrary; we give a general construction in Section 3.

The *shielding problem* is to design a policy π_{shield} that combines $\hat{\pi}$ and π_{backup} (i.e., $\pi_{\text{shield}}(x) \in \{\hat{\pi}(x), \pi_{\text{backup}}(x)\}$) in a way that (i) minimizes $L(\pi_{\text{shield}})$, and (ii) the trajectory generated by π_{shield} from any $x_0 \in \mathcal{X}_0$ is safe. Specifically, we must guarantee (ii), but not necessarily (i)—i.e., π_{shield} must be safe, but may be suboptimal. The key challenge is deciding when to use $\hat{\pi}$ and when to use π_{backup}.

Finally, to guarantee safety, we must make some assumption about $\mathcal{X}_0$; we assume $\mathcal{X}_0 \subseteq \mathcal{X}_{\text{stable}}$.

Model predictive shielding (MPS). Our algorithm for computing $\pi_{\text{shield}}(x)$ is shown in Algorithm 1. At a high level, it checks whether π_{backup} can ensure safety from the state $x' = f^{(\pi)}(x)$ that would be reached by $\hat{\pi}$. If so, then it uses $\hat{\pi}$; otherwise, it uses π_{backup}.

More precisely, let $N \in \mathbb{N}$ be given. Then, a state $x \in \mathcal{X}$ is *recoverable* if for the trajectory $x_0, x_1, ...$ generated by π_{backup} from $x_0 = x$, there exists $t \in \{0, 1, ..., N-1\}$ such that (i) $x_i \in \mathcal{X}_{\text{safe}}$ for all $i \leq t$, and (ii) $x_t \in \mathcal{X}_{\text{stable}}$. Intuitively, π_{backup} safely drives the robot into a stable state from x within N steps. In Algorithm 1, IsRecoverable checks whether $x \in \mathcal{X}_{\text{rec}}$.

Algorithm 1 Model predictive shielding (MPS).

```
procedure MPS(x)
    if IsRecoverable(f^(π̂)(x)) then
        return π̂(x)
    else
        return π_backup(x)
    end if
end procedure
procedure IsRecoverable(x)
    for t ∈ {0, 1, ..., N − 1} do
        if x ∈ X_stable then
            return true
        else if x ∉ X_safe then
            return false
        end if
        x ← f^(π_backup)(x)
    end for
    return false
end procedure
```

Then, π_{shield} uses $\hat{\pi}$ if $f^{(\hat{\pi})}(x)$ is recoverable; otherwise, it uses π_{backup}. We have the following:

Theorem 2.1. *The trajectory generated by π_{shield} from any $x_0 \in \mathcal{X}_0$ is safe.*

We give proofs in Appendix A.

Remark 2.2. The running time of our algorithm on each step is $O(N)$ due to the call to IsRecoverable (assuming $\hat{\pi}$ and π_{backup} run in constant time). We believe this overhead is reasonable for many robots; if necessary, we can safely add a time out, and have IsRecoverable return false if it runs out of time.

3 Backup Policies

We now discuss how to construct π_{backup}. Our construction relies on *safe equilibrium points* of f—i.e., where the robot remains safely at rest. Most robots of interest have such equilibria—for example, the cart-pole has equilibrium points when the cart and pole are motionless, and the pole is perfectly upright. Other examples of equilibrium points include a walking robot standing upright, a quadcopter hovering at a position, or a swimming robot treading water.

One challenge is that these equilibria may be unstable; while the approach described in this section technically ensures safety, it is very sensitive to even tiny perturbations. For example, in the case of cart-pole, a tiny perturbation would cause the pole to fall down. We describe how a way to address this issue in Section 4.

At a high level, our backup policy π_{backup} is composed of two policies: (i) a *stable policy* π_{stable} that ensures safety at equilibrium points, and (ii) a *recovery policy* π_{rec} that tries to drive the robot to a safe equilibrium point. Then, π_{backup} uses π_{rec} until it reaches a safe equilibrium point, after which it uses π_{stable}. Continuing our example, for cart-pole, π_{rec} would try to get the pole into an upright position, and then π_{stable} would stabilize the robot near that position. We begin by describing how we construct π_{stable} and π_{rec}, and then describe how they are combined to form π_{backup}.

Stable policy. An *safe equilibrium point* $z \in \mathcal{Z}_{\text{eq}} \subseteq \mathcal{X} \times \mathcal{U}$ is a pair $z = (x, u)$ such that (i) $x = f(x, u)$, and (ii) $x \in \mathcal{X}_{\text{safe}}$. We let

$$\mathcal{X}_{\text{stable}} = \{x \in \mathcal{X} \mid \exists u \in \mathcal{U} \text{ s.t. } (x, u) \in \mathcal{Z}_{\text{eq}}\}.$$

Furthermore, for $(x, u) \in \mathcal{Z}_{\text{eq}}$, we let $\pi_{\text{stable}}(x) = u$; if multiple such u exist, we pick an arbitrary one. Then, π_{stable} and $\mathcal{X}_{\text{stable}}$ satisfy the conditions for the backup policy. As we describe below, we do not need to define π_{stable} outside of $\mathcal{X}_{\text{stable}}$.

Recovery policy. Using $\pi_{\text{backup}} = \pi_{\text{stable}}$ can result in poor performance. In particular, π_{backup} it is undefined outside of $\mathcal{X}_{\text{stable}}$, so $\mathcal{X}_{\text{rec}} = \mathcal{X}_{\text{stable}}$. As a consequence, π_{shield} will keep the robot inside $\mathcal{X}_{\text{stable}}$. However, since $\mathcal{X}_{\text{stable}}$ consists of equilibrium points, the robot will never move.

Thus, we additionally train a *recovery policy* π_{rec} that attempts to drive the robot into $\mathcal{X}_{\text{stable}}$. The choice of π_{rec} can be arbitrary; however, π_{shield} achieves lower loss for better π_{rec}. There is sometimes an obvious choice (e.g., for a autonomous car, π_{rec} may simply slam the brakes), but not always.

In general, we can use reinforcement learning to train π_{rec}. At a high level, we train it to drive the robot from a safe state reached by $\hat{\pi}$ to the closest safe equilibrium point. First, we use initial state distribution d_{rec}; we define d_{rec} by describing how to take a single sample $x \sim d_{\text{rec}}$: (i) sample an initial state $x_0 \sim d_0$, (ii) sample a time horizon $t \sim \text{Uniform}(\{0, ..., N\})$, (iii) compute the trajectory $x_0, x_1, ...$ generated by $\hat{\pi}$ from x_0, and (iv) reject if $x_t \notin \mathcal{X}_{\text{safe}}$; otherwise take $x = x_t$. Second, we use loss $\ell_{\text{rec}}(x, u) = -\mathbb{I}[x \in \mathcal{X}_{\text{stable}}]$, where $\mathbb{I}$ is the indicator function. We can also use a shaped loss—e.g., $\ell_{\text{rec}}(x, u) = \|x - x'\|^2$, where $x' \in \mathcal{X}_{\text{stable}}$ is the closest safe equilibrium point. Then, we use reinforcement learning to train

$$\pi_{\text{rec}} = \arg\min_{\pi} \mathbb{E}_{x_0 \sim d_{\text{rec}}} \left[\sum_{t=0}^{T-1} \ell_{\text{rec}}(x_t, u_t) \right],$$

where $x_{t+1} = f(x_t, u_t)$, $u_t = \pi(x_t)$, and $T \in \mathbb{N}$.

Backup policy. Finally, we have

$$\pi_{\text{backup}}(x) = \begin{cases} \pi_{\text{stable}}(x) & \text{if } x \in \mathcal{X}_{\text{stable}} \\ \pi_{\text{rec}}(x) & \text{otherwise.} \end{cases}$$

By construction, π_{backup} and $\mathcal{X}_{\text{stable}}$ satisfy the conditions for a backup policy.

4 Unstable Equilibrium Points

For unstable equilibria $z \in \mathcal{Z}_{\text{eq}}$, we use feedback stabilization to ensure safety. [4] As in Section 3, π_{backup} is composed of a stable policy π_{stable}, which is safe on $\mathcal{X}_{\text{stable}}$, and a recovery policy π_{rec}, which tries to drive the robot to $\mathcal{X}_{\text{stable}}$. In this section, we focus on constructing π_{stable} and $\mathcal{X}_{\text{stable}}$; we can train π_{rec} as in Section 3. At a high level, we choose π_{stable} to be the LQR for the linear approximation $\tilde{f}$ of the dynamics around z, and then use LQR verification to compute the states $\mathcal{X}_{\text{stable}}$ for which π_{stable} is guaranteed to be safe. We begin by giving background on LQR control and verification, and then describe our construction.

Assumptions. For tractability, our algorithm makes two additional assumptions. First, we assume that the dynamics f is a degree d polynomial. [5] Second, we assume that the safe set is a convex polytope—i.e.,

$$\mathcal{X}_{\text{safe}} = \{x \in \mathcal{X} \mid A_{\text{safe}} x \leq b_{\text{safe}}\},$$

where $A_{\text{safe}} \in \mathbb{R}^{k \times n_X}$ and $b_{\text{safe}} \in \mathbb{R}^k$ for some $k \in \mathbb{N}$.

Remark 4.1. As in prior work (Tedrake, 2009), for non-polynomial dynamics, we use local Taylor approximations; while our theoretical safety guarantees do not hold, safety holds in practice since these approximations are very accurate. Furthermore, as described below, our results easily extend to arbitrary $\mathcal{X}_{\text{safe}}$.

LQR control. Consider linear dynamics $\tilde{f}(x, u) = Ax + Bu$, where $A \in \mathbb{R}^{n_X \times n_X}$ and $B \in \mathbb{R}^{n_X \times n_U}$, with loss $\ell(x, u) = x^\top Q x + u^\top R u$, where $Q \in \mathbb{R}^{n_X \times n_X}$ and $R \in \mathbb{R}^{n_U \times n_U}$. Then, the optimal policy for these dynamics is a linear policy $\pi_{\text{LQR}}(x) = Kx$, where $K \in \mathbb{R}^{n_U \times n_X}$, called the linear quadratic regulator (LQR) (Tedrake, 2018). [6] Additionally, the cost-to-go function (i.e., the negative value function) of the LQR has the form $J(x) = x^\top P x$, where $P \in \mathbb{R}^{n_X \times n_X}$ is a positive semidefinite matrix. Both the LQR and its cost-to-go can be computed efficiently (Tedrake, 2018).

To stabilize the robot near $z \in \mathcal{Z}_{\text{eq}}$, we use the LQR π_{LQR} for the linear approximation $\tilde{f}$ of f around z;

[4] We mean Lypaunov stability (Tedrake, 2018).

[5] In particular, f is a multivariate polynomial over $x \in \mathcal{X}$ with real coefficients.

[6] The LQR is optimal for the infinite horizon problem.

the cost matrices Q, R can each be chosen to be any positive definite matrix—e.g., the identity. Since $\tilde{f}$ becomes arbitrarily accurate close to z, we intuitively expect π_{LQR} to be a good control policy.

LQR verification. We can use LQR verification to compute a region around (x, u) where π_{LQR} is guaranteed to be safe for an infinite horizon (Parrilo, 2000; Tedrake, 2009, 2018). Given a policy π, $\mathcal{G} \subseteq \mathcal{X}$ is *invariant* for π if for any initial state $x_0 \in \mathcal{G}$, the trajectory generated by π from x_0 is contained in $\mathcal{G}$—i.e., if the robot starts from any $x_0 \in \mathcal{G}$, then it remains in $\mathcal{G}$. We have (Tedrake, 2018):

Lemma 4.2. *Let π be a policy. Suppose that there exists $V : \mathcal{X} \to \mathbb{R}$ and $\epsilon \in \mathbb{R}$ satisfying*

$$V(x) \geq V(f^{(\pi)}(x)) \quad (\forall x \in \mathcal{G}_\epsilon = \{x \in \mathcal{X} \mid V(x) \leq \epsilon\}).$$

Then, $\mathcal{G}_\epsilon$ is an invariant set for π.

Here, V is called a *Lyapunov function*. By Lemma 4.2, given a candidate Lyapunov function V, we can use optimization to compute ϵ such that $\mathcal{G}_\epsilon$ is invariant. In particular, given a set $\mathcal{F}$ of functions $f : \mathcal{X} \to \mathbb{R}$, a policy π, and a candidate Lyapunov function V, let

$$\begin{aligned} \epsilon^* = \max_{\lambda \in \mathcal{F}, \vec{\mu} \in \mathcal{F}^k, \epsilon' \in \mathbb{R}} \ & \epsilon \ \text{ subj. to} \\ & V(x) - V(f^{(\pi)}(x)) + \lambda(x)(V(x) - \epsilon') \geq 0 \\ & b_{\text{safe}} - A_{\text{safe}} x + \vec{\mu}(x)(V(x) - \epsilon') \geq 0 \\ & \lambda(x),\ \vec{\mu}(x),\ \epsilon' \geq 0 \end{aligned} \tag{1}$$

where the constraints are required to hold for all $x \in \mathbb{R}^{n_X}$. We have the following (Tedrake, 2018):

Lemma 4.3. *We have (i) $\mathcal{G}_{\epsilon^*}$ is invariant for π, and (ii) π is safe from any $x_0 \in \mathcal{G}_{\epsilon^*}$.*

A standard choice for the candidate Lyapunov function V is the cost-to-go function J of π_{LQR}—i.e., $V(x) = J(x) = x^\top P x$. Indeed, for the linear approximation $\tilde{f}$, J is a Lyapunov function of π_{LQR} on all of $\mathbb{R}^{n_X}$. Thus, J is a promising choice of the candidate Lyapunov function for the true dynamics f.

The optimization problem (1) is intractable in general. We use a standard modification that strengthens the constraints to obtain tractability; the resulting solution is guaranteed to satisfy the original constraints, but may achieve a suboptimal objective value. First, for some $d' \in \mathbb{N}$, we choose $\mathcal{F}$ to be the set of polynomials in x of degree at most d'. Then, for $\pi = \pi_{\text{LQR}}$, each constraint in (1) has form $p(x) \geq 0$ for some polynomial $p(x)$. We replace each constraint $p(x) \geq 0$ with the stronger constraint that $p(x)$ is a *sum-of-squares (SOS)*—i.e., $p(x) = p_1(x)^2 + ... + p_k(x)^2$ for some polynomials $p_1, ..., p_k$. If $p(x)$ is SOS, then $p(x) \geq 0$ for all $x \in \mathcal{X}$. With this modification, the optimization problem (1) is an *SOS program*; for our choice of $\mathcal{F}$, it can be solved efficiently using semidefinite programming (Parrilo, 2000; Tedrake, 2009, 2018).

Remark 4.4. Our approach is sound—i.e., the solution to our SOS program is guaranteed to satisfy the constraints in (1), so the statement of Lemma 4.3 holds; however, our solution may be suboptimal.

Remark 4.5. For general $\mathcal{X}_{\text{safe}}$, given an equilibrium point $(x, u) \in \mathcal{Z}_{\text{eq}}$, consider a convex polytope $\tilde{\mathcal{X}}_{\text{safe}} = \{x \in \mathcal{X} \mid \tilde{A}_{\text{safe}} x \leq \tilde{b}_{\text{safe}}\}$ satisfying (i) $\tilde{\mathcal{X}}_{\text{safe}} \subseteq \mathcal{X}_{\text{safe}}$, and (ii) $x \in \tilde{\mathcal{X}}_{\text{safe}}$. Then, we can conservatively use $\tilde{A}_{\text{safe}}, \tilde{b}_{\text{safe}}$ in place of $A_{\text{safe}}, b_{\text{safe}}$ when solving the optimization problem (1).

Stable policy. Given a safe equilibrium point $z \in \mathcal{Z}_{\text{eq}}$, let π_{LQR} be the LQR for the linear approximation $\tilde{f}$ around z; then, we let $\pi_z = \pi_{\text{LQR}}$. Furthermore, let ϵ^* be the solution to the SOS variant of the optimization problem (1); then, we let $\mathcal{G}_z = \mathcal{G}_{\epsilon^*}$ be an invariant set of π_z. Now, we choose

$$\pi_{\text{stable}}(x) = \pi_{\rho(x)}(x) \ \text{ and } \ \mathcal{X}_{\text{stable}} = \bigcup_{z \in \mathcal{Z}_{\text{eq}}} \mathcal{G}_z,$$

where $\rho(x)$ is the closest equilibrium point to x—i.e., $\rho(x) = \arg\min_{(x', u') \in \mathcal{Z}_{\text{eq}}} \|x - x'\|$. In other words, $\pi_{\text{stable}}(x)$ uses the LQR for the equilibrium point closest to x, and $\mathcal{X}_{\text{stable}}$ is the set of states in the invariant set of some equilibrium point. We have the following:

Theorem 4.6. *The trajectory generated using π_{stable} from any $x_0 \in \mathcal{X}_{stable}$ is safe.*

Remark 4.7. Computing π_{stable} is polynomial time, but may still be costly—given x, we need to compute the nearest equilibrium point z, and then compute π_z and $\mathcal{G}_z$. In practice, we can often precompute these. For example, for cart-pole, the dynamics are equivariant under translation. Thus, we can compute the $\pi_{z_0}(x) = K_0 x$ and $\mathcal{G}_{z_0}$ for the origin $z_0 = (\vec{0}, \vec{0})$, and perform a change of coordinates to use these for other z. In particular, for any $z = (x', \vec{0}) \in \mathcal{Z}_{\text{eq}}$, we have $\pi_z(x) = K_0(x - x')$ and $\mathcal{G}_z = \{x' + x \mid x \in \mathcal{G}_{z_0}\}$.

5 Experiments

We evaluate our approach on two benchmarks, the cart-pole and the bicycle, showing how it ensures safety in a scalable way. All experiments are run on a 2.9 GHz Intel Core i9 CPU with 32GB memory.

Benchmarks. First, we consider the cart-pole (Barto et al., 1983) with continuous actions. Our goal is for the cart to have positive velocity (i.e., move towards the right), with a target velocity of $v_0 = 0.1$. The safety constraint is that the pole angle does not exceed $\theta_{\max} = 0.15$ rad from the upright position. The initial state distribution is $\text{Uniform}([-0.05, 0.05]^4)$.

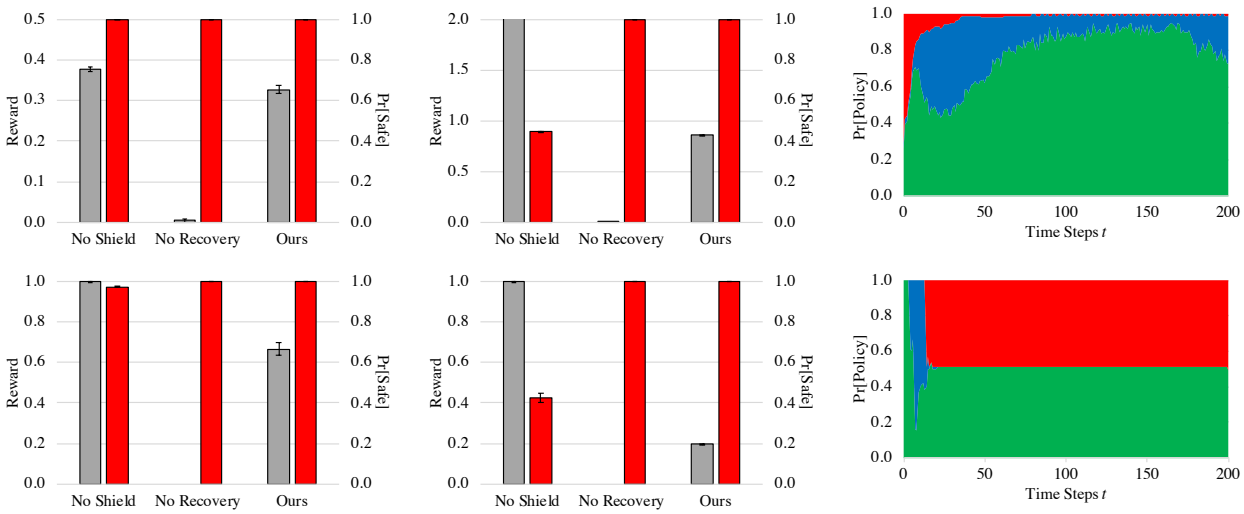

Figure 2: For cart-pole (top) and bicycle (bottom): Reward (gray) and safety probability (red) for original (left) and modified (middle) environments, and probability of using of $\hat{\pi}$ (green), π_{rec} (blue), and π_{LQR} (red) as a function of t on the original cart-pole and bicycle environments (right). For modified cart-pole, we threshold reward at 2.0; "No Shield" achieves larger reward using unsafe behavior. We show means (all) and standard errors (left, middle) over 100 random rollouts.

Second, we consider a bicycle (Taheri, 1992; Pepy et al., 2006), which has a 4-dimensional state space (x, y, v, θ), where (x, y) is the front of the car, v is the velocity, and θ is the heading, and a 2-dimensional action space (a, ϕ), where a is the acceleration and ϕ is the steering angle. We assume that $|a| \leq a_{\max} = 0.25$. The goal is to get from the initial state $(0, 0, 0, 0)$ to the target $x = 1$. In addition, the bicycle must avoid two obstacles, which have x positions 0.4 and 0.7, y positions sampled i.i.d. from Uniform($[-0.05, 0.05]$), and radius 0.05.

Reinforcement learning. We use backpropagation-through-time (BPTT)—i.e., backpropagate through both the policy and the dynamics—to learn $\hat{\pi}$ and π_{rec}. Each policy is a single-layer neural network with 200 hidden units and ReLU activations. For the bicycle, we include the y positions of the obstacles as inputs to the neural network. For each task, we train using a time horizon $T = 200$ steps and a discount factor $\gamma = 0.99$. We use the ADAM optimizer, with a learning rate tuned using cross validation.

Backup policy. For the cart-pole, we use $\rho((x, v, \theta, \omega)) = ((x, 0, 0, 0), 0)$—i.e., stabilize the pole to the origin at the current cart position. We use the degree 5 Taylor approximation around the origin for LQR verification, and degree 6 polynomials for $\mathcal{F}$ in the SOS program. For the bicycle, we do not need LQR verification since its equilibrium points $((x, y, 0, \theta), (0, 0))$ are stable. In prinicple, we could use $\mathcal{X}_{\text{stable}} = \{(x, y, 0, \theta)\}$, but π_{rec} may have difficulty bringing the robot to an exact stop since it is learned. Instead, we use $\mathcal{X}_{\text{stable}} = \{(x, y, v, \theta) \mid |v| \leq a_{\max}\}$ and $\pi_{\text{stable}}((x, y, v, \theta)) = (-|v|, 0)$—i.e., when possible, π_{stable} decelerates the robot to bring it to a stop.

Modified problems. Changes in the planning problem can be a cause of safety failures, since the learned policy $\hat{\pi}$ is tailored to perform well in the original problem. Thus, if the problem changes—e.g., different configurations of obstacles, or longer time horizon—then $\hat{\pi}$ may become unsafe to use. To demonstrate how MPS can ensure safety in the face of such changes, we consider a modification to each robot. First, for the cart-pole, we increase the time horizon—though $\hat{\pi}$ and π_{rec} are trained with a time horizon of $T = 200$, we use them to control the robot for a time horizon of‘ $T = 1000$. Second, for the bicycle, we enlarge the obstacles to have radius 0.2, compared to 0.05 originally.

Results. In Figure 2 (left, middle), we show the reward achieved for both benchmarks and their respective modifications. [7] The reward shown is the actual performance—z for cart-pole (i.e., distance traveled by the cart), and x for the bicycle (i.e., distance traveled towards the target)—rather than the shaped reward used to learn $\hat{\pi}$. We also show the safety probability—i.e., the probability that a random state visited during a random rollout is safe. We show results for (i) $\hat{\pi}$ ("No Shield"), (ii) π_{shield} without π_{rec}—i.e., using $N = 0$ ("No Recovery"), and (iii) π_{shield} using $N = 100$ ("Ours"). All of our shielded policies (i.e., both $N = 0$

[7] As usual, the reward is the negative loss.

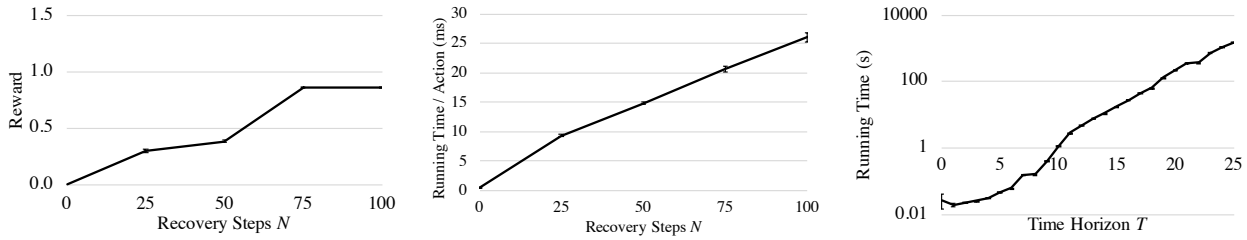

Figure 3: Reward (left) and time per action (middle) on modified cart-pole for π_{shield} as a function of $N \in \{0, 25, 50, 75, 100\}$, and time to verify the cart-pole policy from (Bastani et al., 2018) as a function of N (right). We show means and standard errors over 10 random rollouts (left, middle) or 4 runs (right).

and $N = 100$) achieve perfect safety. The learned policy $\hat{\pi}$ achieves good safety probability (1.0 for cart-pole, 0.97 for bicycle) on the original environments. However, as expected, it performs very poorly on the modified environments (0.45 on cart-pole, 0.42 on bicycle), since it was not trained to account for these modifications. While our shielded policy (with $N = 100$) achieves somewhat reduced reward compared to $\hat{\pi}$, it always achieves perfect safety.

Policy usage. In Figure 2 (right), for π_{shield} with $N = 100$, we show the probability of using $\hat{\pi}$, π_{rec}, and π_{LQR} as a function of time t, on the original cart-pole and bicycle environments. For cart-pole, π_{shield} initially uses π_{LQR} to upright the pole, and then proceeds to use a combination of $\hat{\pi}$ and π_{rec}. For the modified environment (plot omitted), π_{shield} inevitably switches to using π_{LQR} only—$\hat{\pi}$ acts pathologically (and unsafely) for states with large z, since it was not trained on these states. For the bicycle, in about half the rollouts, π_{shield} switches to π_{LQR} and does not make further progress, likely because the obstacle was blocking the way. For the remaining rollouts, π_{shield} uses $\hat{\pi}$ for most of the rollout. For the modified environment (plot omitted), π_{shield} almost always uses π_{LQR}.

Need for a recovery policy. Our results demonstrate the importance of using π_{rec}—when $N = 0$, π_{shield} only uses π_{LQR}, and makes no progress. In Figure 3 (left), we additionally show how reward varies as a function of the recovery steps N for modified cart-pole. There is a large improvement even for $N = 25$; performance then levels off, with $N = 75$ and $N = 100$ achieving similar reward.

Running time. We study the running time of π_{shield}—i.e., how long it takes to compute a single action $u = \pi_{\text{shield}}(x)$. In Figure 3, we show how the average running time varies as a function of the recovery steps N, on cart-pole. As expected, for $N = 100$, the running time is about $100\times$ the running time of $\hat{\pi}$ (26.0ms vs. 0.2ms), since it simulates the dynamics for 100 steps. We believe this overhead is an acceptable cost for guaranteeing safety. The worst case running time is linear in N, but can be sublinear if π_{rec} reaches a stable state in fewer than N steps.

Comparison to ahead-of-time verification. We compare our approach to ahead-of-time verification. One challenge is that these techniques typically only perform verification for a bounded state space or for a bounded time horizon T. In Figure 3, we show how the running time of a state-of-the art verification algorithm scales as a function of T for verifying a cart-pole policy (Bastani et al., 2018). The y-axis is log-scale—thus, verification is exponential in T. Even for $T = 25$, it takes about 24 minutes to perform verification. Though this computation is offline, it quickly becomes intractable for large T. As a rough estimate, the running time grows $10^2\times$ from $T = 10$ to $T = 20$; extrapolating this trend, the running time for $T = 200$ would be over 10^{30} years. In contrast, our approach not only ensures safety for an unbounded horizon, but is also substantially more computationally feasible.

We also compare to verifying safety by overapproximating the dynamics. A common approach is to use a linear approximation of the closed-loop dynamics and use upper bounds on the Lipschitz constant to upper bound the approximation error; then, ellipsoids can be used to overapproximate the dynamics (Koller et al., 2018). We have tried using this approach to verify our learned cart-pole policy. However, the error due to the overapproximation quickly becomes unmanageable—e.g., we could not prove safety even for $T = 3$.

6 Conclusion

We leave much room for future work—e.g., extending our approach to handle unknown dynamics, partial observability, and multi-agent robots.

Acknowledgements

This work was supported in part by NSF Award CCF-1910769.

References

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *ICML*, 2017.

Anayo K Akametalu, Shahab Kaynama, Jaime F Fisac, Melanie Nicole Zeilinger, Jeremy H Gillula, and Claire J Tomlin. Reachability-based safe learning with gaussian processes. In *CDC*, pages 1424–1431. Citeseer, 2014.

Mohammed Alshiekh, Roderick Bloem, Rudiger Ehlers, Bettina Konighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *AAAI*, 2018.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

Leonhard Asselborn, Dominic Gross, and Olaf Stursberg. Control of uncertain nonlinear systems using ellipsoidal reachability calculus. *IFAC Proceedings Volumes*, 46(23):50–55, 2013.

Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5): 834–846, 1983.

Osbert Bastani, Pu Yewen, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in neural information processing systems*, 2018.

Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in Neural Information Processing Systems*, pages 908–918, 2017.

Yinlam Chow, Ofir Nachum, and Edgar Duenez-Guzman. A lyapunov-based approach to safe reinforcement learning. In *NeurIPS*, 2018.

Sarah Dean, Stephen Tu, Nikolai Matni, and Benjamin Recht. Safely learning to control the constrained linear quadratic regulator. *arXiv preprint arXiv:1809.10121*, 2018.

Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *JMLR*, 2015.

Jeremy H. Gillula and Claire J. Tomlin. Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor. In *ICRA*, 2012.

Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *HSCC*, 2019.

Arbaaz Khan, Ekaterina Tolstaya, Alejandro Ribeiro, and Vijay Kumar. Graph policy gradients for large scale robot control. In *CORL*, 2019.

Torsten Koller, Felix Berkenkamp, Matteo Turchetta, and Andreas Krause. Learning-based model predictive control for safe exploration. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 6059–6066. IEEE, 2018.

Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous Robots*, 40(3):429–455, 2016.

Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pages 1–9, 2013.

Teodor M. Moldovan and Pieter Abbeel. Safe exploration in markov decision processes. In *ICML*, 2012.

Pablo A Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, California Institute of Technology, 2000.

Romain Pepy, Alain Lambert, and Hugues Mounier. Path planning using a dynamic vehicle model. In *2006 2nd International Conference on Information & Communication Technologies*, volume 1, pages 781–786. IEEE, 2006.

Theodore J. Perkins and Andrew G. Barto. Lyapunov design for safe reinforcement learning. *JMLR*, 2002.

Saied Taheri. An investigation and design of slip control braking systems integrated with four-wheel steering. 1992.

Russ Tedrake. Lqr-trees: Feedback motion planning on sparse randomized trees. In *RSS*, 2009.

Russ Tedrake. *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. 2018. URL `http://underactuated.mit.edu/`.

Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration in finite markov decision processes with gaussian processes. In *NIPS*, 2016.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaud-

huri. Programmatically interpretable reinforcement learning. *arXiv preprint arXiv:1804.02477*, 2018.

Min Wen and Ufuk Topcu. Constrained cross-entropy method for safe reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 7450–7460, 2018.

Yifan Wu, Roshan Shariff, Tor Lattimore, and Csaba Szepesvári. Conservative bandits. In *ICML*, 2016.

He Zhu, Zikang Xiong, Stephen Magill, and Suresh Jagannathan. An inductive synthesis framework for verifiable reinforcement learning. In *PLDI*, 2019.

A Appendix

Proof of Theorem 2.1. We prove by induction that if $x_t \in \mathcal{X}_{\text{rec}}$, then $x_{t+1} = f^{(\pi_{\text{shield}})}(x_t) \in \mathcal{X}_{\text{rec}}$. The base case holds since $x_0 \in \mathcal{X}_0 \subseteq \mathcal{X}_{\text{stable}} \subseteq \mathcal{X}_{\text{rec}}$. For the inductive case, there are two possibilities. (i) If $x' = f^{(\hat{\pi})}(x_t) \in \mathcal{X}_{\text{rec}}$, then $\pi_{\text{shield}}(x_t) = \hat{\pi}(x_t)$, so $x_{t+1} = x' \in \mathcal{X}_{\text{rec}}$. (ii) Otherwise, $\pi_{\text{shield}}(x_t) = \pi_{\text{backup}}(x_t)$; clearly, $x_t \in \mathcal{X}_{\text{rec}}$ implies that $x'' = f^{(\pi_{\text{backup}})}(x_t) \in \mathcal{X}_{\text{rec}}$, so $x_{t+1} = x'' \in \mathcal{X}_{\text{rec}}$. Thus, the inductive case holds. The claim follows. □

Proof of Lemma 4.2. The claim follows by induction on t. □

Proof of Lemma 4.3. Consider any x such that $V(x) \leq \epsilon$. To see (i), note that in the first constraint in (1), the second term is negative since $\lambda(x) \geq 0$, so $V(x) - V(f^{(\pi)}(x)) \geq 0$. Thus, by Lemma 4.2, $\mathcal{G}_\epsilon$ is invariant. Similarly, to see (ii), note that in the second constraint in (1), the second term is negative since $\vec{\mu}(x) \geq 0$, so $b_{\text{safe}} - A_{\text{safe}}x \geq 0$. Thus, $x \in \mathcal{X}_{\text{safe}}$ for all $x \in \mathcal{G}_\epsilon$. Since $\mathcal{G}_\epsilon$ is invariant, π is safe from any $x_0 \in \mathcal{G}_\epsilon$, so the claim follows. □

Proof of Theorem 4.6. We prove by induction on t that $x_t \in \mathcal{X}_{\text{stable}}$ for all $t \geq 0$. The base case follows by assumption. For the inductive case, note that $x_t \in \mathcal{G}_{\rho(x)}$, so we have $f^{(\pi_{\rho(x)})}(x_t) \in \mathcal{G}_{\rho(x)}$ since $\mathcal{G}_{\rho(x)}$ is invariant. Thus, $x_{t+1} = f^{(\pi_{\text{backup}})}(x_t) = f^{(\pi_{\rho(x)})}(x_t) \in \mathcal{G}_{\rho(x)}$, so the inductive case follows. By construction, $\mathcal{X}_{\text{stable}} \subseteq \mathcal{X}_{\text{safe}}$, so the claim follows. □