Programmatic Reinforcement Learning with Formal Verification

Yuning Wang [1] **He Zhu** [1]

Abstract

We present Marvel, a verification-based reinforcement learning framework that synthesizes safe programmatic controllers for environments with continuous state and action space. The key idea is the integration of program reasoning techniques into reinforcement learning training loops. Marvel performs abstraction-based program verification to reason about a programmatic controller and its environment as a closed-loop system. Based on a novel verification-guided synthesis loop for training, Marvel minimizes the amount of safety violation in the system abstraction, which approximates the *worst-case* safety loss, using gradient-descent style optimization. Experimental results demonstrate the substantial benefits of leveraging verification feedback for safe-by-construction of programmatic controllers.

1. Introduction

In safety-critical domains, guaranteeing the safety of reinforcement learning controllers is important. Principally, a controller can be verified or even synthesized using program verification and program synthesis. Indeed, the use of automated program reasoning techniques to aid the design of reliable machine learning systems has risen rapidly over the last few years. A notable example is the application of abstract interpretation (Cousot & Cousot, 1977) to verify robustness of convolutional neural networks (Gehr et al., 2018). For supervised learning, the robustness property of a neural network requires that its outputs be consistent for narrow input spaces surrounding individual data points. A natural extended question is that can we use program verification and synthesis to address safe reinforcement learning where a system includes both an environment and a controller? Moreover, in case verification fails, can we exploit verification counterexamples to synthesize a safe controller?

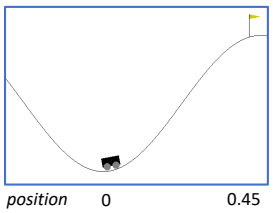
*Equal contribution [1]Rutgers University, USA. Correspondence to: Yuning Wang <yw895@rutgers.edu>, He Zhu <hz375@cs.rutgers.edu>.

1st Workshop on Formal Verification of Machine Learning, Baltimore, Maryland, USA. Colocated with ICML 2022. Copyright 2022 by the author(s).

For reinforcement learning using deep neural networks, the primary barrier to safety verification is that repeatedly verifying a deep neural network at every timestep within a nonlinear, closed-loop control system is computationally infeasible. Consider the continuous MountainCar environment from OpenAI Gym visualized in Fig. 1a. A car is on a one-dimensional track, positioned between two "mountains". The goal is to drive up the mountain on the right. Because the car's engine is not strong enough, a controller has to drive the car back and forth to build up momentum in multiple passes. Even for this simple example, verifying a deep neural controller satisfying the goal requires huge verification budgets. For example, Verisig (Ivanov et al., 2018) (a state-of-the-art verification tool for neural network controlled systems) needs more than 22 minutes to verify a deep neural controller for MountainCar.

Recently, programmatic controllers in more structured representations emerge as a promising solution to address the lack of interpretability problem in deep reinforcement learning (Verma et al., 2018; Inala et al., 2020; Trivedi et al., 2021; Qiu & Zhu, 2022). In this paper, other than interpretability, we show verifiability, another key benefit of programmatic controllers. A learned programmatic controller to control the continuous MountainCar environment learned by (Qiu & Zhu, 2022) is given in Fig. 1b. The logic of the program is interpretable — whenever the car position is greater than 0, the controller accelerates the force to drive the car forward; when ever the car position is less than 0, the controller accelerates the force to drive the car backward. Indeed, this helps drive the car back and forth to build up momentum. The controller solves the task and succeeds in reaching the goal on experienced episodes.

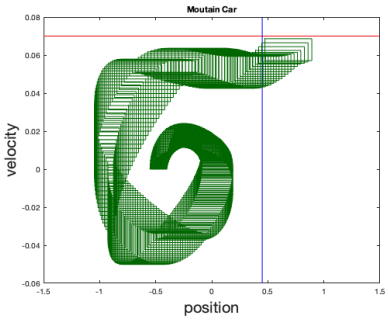
To formally verify that the goal can be reached on all (unseen) episodes, we used a reachability analyzer Flow* (Chen et al., 2013), a verification tool for hybrid or continuous systems, to conduct reachability analysis to compute an over-approximation for a reachable state set between each time interval within the episode horizon (the controller is applied to generate a control action at the start and end of each time interval). The result is depicted in Fig. 1c. The car starts at any position within $[-0.6, -0.4]$ and eventually reaches the goal: position ≥ 0.45 while maintaining a safety constraint on the upperbound of its speed before reaching the goal. Verification takes only 2 minutes. The



(a) MountainCar Gym Environment

if $position > -0.007$
then $0.92 \longleftarrow$ **Branch 1**
else $-0.95 \longleftarrow$ **Branch 2**

(b) MountainCar Programmatic Controller



(c) MountainCar Reachability Analysis. The blue line represents the goal region: position ≥ 0.45. The read line represents a safety constraint: velocity ≤ 0.07.

Figure 1. The continuous MountainCar environment, its programmatic controller, and the reachability analysis of the controller.

example shows that it is more feasible to conduct verification for a programmatic controller and have verification in a programmatic reinforcement learning loop, compared to using a deep neural network. The question remains when verification fails – rather than retraining a new controller, how can we leverage verification feedback to construct a verifiably safe controller?

We present Marvel as a new verification-based controller synthesis framework that consists of two main components, namely *controller verification*, and *controller synthesis*.

- **Controller Verification**. Given an environment model and a programmatic controller, Marvel verifies the safety of the controller by reachability analysis over a closed-loop system that combines the environment model and the programmatic controller.
- **Controller Synthesis**. A safety counterexample detected by Marvel is a *symbolic rollout* of abstract states of the closed-loop system because it is obtained by over-approximation. Marvel quantifies the safety properties violation by the abstract states. The goal of controller synthesis is to effectively minimize the amount of safety violation to refute any safety counterexamples.

The most important feature of our algorithm is that, instead of synthesizing a programmatic controller with *concrete* examples, we use *symbolic rollouts* with abstract states obtained by reachability analysis. Marvel reduces the amount of safety properties violation by the abstraction states, which approximates the *worst-case* safety loss, using a lightweight gradient-descent style optimization. Marvel efficiently leverages verification feedback in a learning loop to enable controller safe-by-construction. Our experiments demonstrate the benefits of integrating formal verification as part of the training objective and using verification feedback for safe controller synthesis.

2. Verification-guided Controller Synthesis

We formulate safe controller synthesis in the context of environment models akin to Markov Decision Process (MDP).

Environment Models. Formally, an environment is a structure $M = (X, S, A, F : \{S \times A \to S\}, S_0, \varphi_{safe}, \varphi_{reach}, R : \{S \times A \to \mathbb{R}\}, \beta)$ where X is a finite set of variables interpreted over the reals $\mathbb{R}$; S is an infinite set of *continuous real-vector* environment states which are valuations of the variables X ($S \subseteq \mathbb{R}^{|X|}$); A is a set of *continuous real-vector* agent actions; F is a state transition function that emits the next environment state given a current state s and an agent action a; We assume that the initial states of M are uniformly sampled from a set of environment states $S_0 \subseteq S$. $R(s, a)$ is the immediate reward after transition from an environment state s with action a and $0 < \beta \leq 1$ is a reward discount factor.

Controllers. An agent of an environment M interacts with the environment by taking actions via a controller conditioned on environment states. Formally, a controller is a (stochastic) map $\pi : \{S \to A\}$ that determines which action the agent ought to take in a given state.

State Transitions. Particularly, we assume the transition function F in an environment model is defined by an ordinary differential equation (ODE) in the form of $\dot{x} = f(x, a)$ such that x represents state variables of dimension n and a represents control inputs of dimension m. We assume that the function $f : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^m$ is Lipschitz continuous in x and continuous in a. Given a sampling period δ, the controller π reads the environment state $s_i = s(i\delta)$ at time $t = i\delta$ ($i = 0, 1, 2, \ldots$), and computes a control input as $a_i = a(i\delta) = \pi(s(i\delta))$. Then the environment evolves as $\dot{x} = f(x, a(i\delta))$ within the time slot $[i\delta, (i + 1)\delta]$ to obtain the next state $s_{i+1} = s((i + 1)\delta)$ at time $(i + 1)\delta$. We assume s_i (*resp.* s_{i+1}) as the solution of the ODE at time $t = i\delta$ (*resp.* at time $t = (i + 1)\delta$) is given by a flow

function[1] $\phi(s_0, t) : S_0 \times \mathbb{R}^+ \to S$ that maps some initial state s_0 to the environment state $\phi(s_0, t)$ at time t where $\phi(s_0, 0) = s_0$.

Example 2.1. Consider a Van der Pol's oscillator system taken from (Wang et al., 2021). The oscillator is a 2-dimensional non-linear system. The system state transition can be expressed by the following ordinary differential equations:

$$\dot{x_1} = x_2 \qquad \dot{x_2} = (1 - x_1^2)x_2 - x_1 + a \tag{1}$$

where $(x1, x2)$ is the system state variables and a represents a control action. A system as such is representative of a number of autonomous systems, like drones, that have thus far proven difficult for safety verification, but for which high assurance is extremely important.

Definition 2.2 (**STS** – State Transition Systems)**.** An environment $M = (X, S, A, F, S_0, \varphi_{safe}, \varphi_{reach}, R, \beta)$ can be viewed as a state transition system $\mathcal{F}^\delta[\cdot] = (F, \cdot, S_0)$ with sampling period δ, parameterized by an unknown controller $\pi : S \to A$, where the state transition function F receives control actions given by the controller every δ time and S_0 is the initial state space. We explicitly model controller deployment as $\mathcal{F}^\delta[\pi] = (F, \pi, S_0)$. Structurally, $\mathcal{F}^\delta[\pi]$ consists of T layers. Each layer at i ($i \geq 0$) is a function F_π^i: $\lambda s_i.F(s_i, \pi(s_i))$, which takes as input a state s_i at time $t = i\delta$ and outputs its next state s_{i+1} at time $(i + 1)\delta$. Formally, layer i is a compositional function of an initial state: $\mathcal{F}_i^\delta[\pi] = F_\pi^i \circ \ldots \circ F_\pi^1 \circ F_\pi^0$. Analytically, $\mathcal{F}_i^\delta[\pi](s_0) = \phi(s_0, (i + 1)\delta)$ where ϕ is the flow function that compute the solutions of the ODE that constitute the state transition function F (formally defined above).

Rollouts. Given a time horizon $T\delta$ with δ as the sampling period, a T-timestep rollout of a controller π is denoted as $s_0, a_0, s_1, \ldots, s_{T'} \sim \pi$ where $T' \leq T$ and $s_i = s(i\delta)$ and $a_i = a(i\delta)$ are the environment state and the action taken at timestep i such that $s_0 \in S_0$, $s_{i+1} \sim F(s_i, a_i)$, and $a_i = \pi(s_i)$. The aggregate reward of π is

$$J^R(\pi) = \mathbb{E}_{s_0, a_0, \ldots, s_{T'} \sim \pi}[\sum_{t=0}^{T'} \beta^t R(s_i, a_i)]$$

Controller search via reinforcement learning aims to produce a controller π that maximizes $J^R(\pi)$.

Safe Rollouts. In this work, an environment M includes two safety properties φ_{safe} and φ_{reach} as logical formulae over environment states. They specify the intended behavior of any agents of M. φ_{safe} enforces that agents should only visit safe environment states evaluated true by φ_{safe}. For example, an agent should remain within a safety boundary or avoid any obstacles. Additionally, M enforces that agents should eventually reach some environment states evaluated true by φ_{reach}. For instance, an agent should meet some goals.

Definition 2.3 (Safety Property)**.** Given an environment model, we assume S_0 defines a bounded domain in the form of an interval $[\underline{x}, \overline{x}]$ where $\underline{x}, \overline{x} \in \mathbb{R}^{|X|}$ are lower and upper bounds of the initial states. Both φ_{safe} and φ_{reach} are quantifier-free Boolean combinations of linear inequalities over the environment variables X:

$$\langle \varphi_{safe},\ \varphi_{reach} \rangle ::= \langle P \rangle \mid \varphi \wedge \varphi \mid \varphi \vee \varphi;$$

$$\langle P \rangle ::= \mathcal{A} \cdot x \leq b \text{ where } \mathcal{A} \in \mathbb{R}^{|X|},\ b \in \mathbb{R};$$

A state $s \in S$ satisfies φ_{safe} or φ_{reach}, denoted as $s \models \varphi_{safe}$ or $s \models \varphi_{reach}$, iff $\varphi_{safe}(s)$ or $\varphi_{reach}(s)$ is true.

Given a time horizon $T\delta$, a controller π is safe for a state transition system $\mathcal{F}^\delta[\cdot]$ with respect to φ_{safe} and φ_{reach}, denoted as $\mathcal{F}^\delta[\pi] \models \varphi_{safe}, \varphi_{reach}$, iff, for any rollout of $\mathcal{F}^\delta[\pi]$,

$$\underbrace{s_0, s_1, \ldots s_{T'-2}, s_{T'-1}}_{\forall t \in 1 \ldots T'-1,\ s_i \models \varphi_{safe}} \quad \underbrace{s_{T'}}_{s_{T'} \models \varphi_{reach}}$$

where $T' \leq T$, $s_0 \in S_0$ and $\forall i \geq 0, s_{i+1} = \mathcal{F}_i^\delta[\pi](s_0)$.

Throughout the paper, we consider both safety specifications φ_{safe} and bounded-time reachability specifications φ_{reach} as safety properties to constrain a safe controller.

Example 2.4. Continue the oscillator example. The initial set S_0 of oscillator is $[-0.51, -0.49] \times [0.49, 0.51]$. We specify its initial states as:

$$S_0 \equiv \{(x_1, x_2) \mid -0.51 \leq x_1 \leq -0.49 \wedge 0.49 \leq x_2 \leq 0.51\}$$

The oscillator unsafe set is $[-03, -0.25] \times [0.2, 0.35]$. The safety property φ_{safe} of the system is specified as:

$$\varphi_{safe}((x_1, x_2)) \equiv \neg(-03 \leq x_1 \leq -0.25 \wedge 0.2 \leq x_2 \leq 0.35)$$

For this example, the goal set is $[-0.05, 0.05] \times [-0.05, 0.05]$. We set the *bounded-time* reachability property φ_{reach} as

$$\varphi_{reach}((x_1, x_2)) \equiv -0.05 \leq x_1, x_2 \leq 0.05$$

The goal set should be eventually reached within 120 timesteps.

An episode rollout of the oscillator environment is $\tau = s_0, a_0, s_1, \ldots, a_{T-1}, s_T$ ($T = 120$) that starts from an initial state $s_0 \in S_0$. The sampling period δ is $0.05s$ and time horizon is $6s$. A controller π reads the state $s_i = s(i\delta)$ at time $t = i\delta$ ($i = 0, 1, 2, \ldots$), and computes a control action as $a_i = a(i\delta) = \pi(s(i\delta))$. Then the system evolves as Eq. 1 within the time slot $[i\delta, (i+1)\delta]$ and yields a new state s_{i+1}.

[1] ϕ may be implemented using scipy.integrate.odeint (or scipy.integrate.solve_ivp).

$$
\begin{aligned}
E &::= C \mid \textbf{if } B \textbf{ then } C \textbf{ else } E \\
B &::= \theta_1 + \theta_2^T \cdot \mathcal{X} \geq 0 \\
C &::= \theta_3 + \theta_4 \cdot \mathcal{X} \mid \theta_c
\end{aligned}
$$

Figure 2. A Context-free DSL Grammar for programmatic controllers.

Programmatic Controllers. Previous work (Verma et al., 2018; Qiu & Zhu, 2022) has shown that simple programmatic controllers are more interpretable than and achieve comparable reward performance to deep neural network controllers. In this paper, we focus on programmatic controllers as *differentiable* programs (Qiu & Zhu, 2022).

Our programmatic controllers follow the high-level context-free grammar depicted in Fig. 2 where E is the start symbol, θ represents real-valued parameters of the program. The nonterminals E and B stand for program expressions that evaluate to action values in $\mathbb{R}^m$ and Booleans, respectively, where m is the action dimension size, $\theta_1 \in \mathbb{R}$ and $\theta_2 \in \mathbb{R}^n$. We represent a state input to a programmatic controller as $s = \{x_1 : \nu_1, x_2 : \nu_2, \ldots, x_n\}$ where n is the state dimension size and $\nu_i = s[x_i]$ is the value of x_i in s. As usual, the unbounded variables in $\mathcal{X} = [x_1, x_2, \ldots, x_n]$ are assumed to be input variables (i.e., state variables). C is a low-level affine controller that can be called by a programmatic controller where $\theta_3, \theta_c \in \mathbb{R}^m, \theta_4 \in \mathbb{R}^{m \cdot n}$ are controller parameters. Notice that C can be as simple as some (learned) constants θ_c.

The semantics of a programmatic controller in E is mostly standard and given by a function $[\![E]\!](s)$, defined for each language construct. For example, $[\![x_i]\!](s) = s[x_i]$ reads the value of a variable x_i in a state input s. A controller may use an **if-then-else** branching construct. To avoid discontinuities for differentiability, we interpret its semantics in terms of a smooth approximation where σ is the sigmoid function:

$$
\begin{aligned}
&[\![\textbf{if } B \textbf{ then } C \textbf{ else } E]\!](s) = \\
&\quad \sigma([\![B]\!](s)) \cdot [\![C]\!](s) + (1 - \sigma([\![B]\!](s))) \cdot [\![E]\!](s)
\end{aligned} \tag{2}
$$

Thus, any controller programmed in this grammar becomes a differentiable program. During execution, a programmatic controller can invoke a set of low-level affine controllers under different environment conditions, according to the activation of B conditions in the program.

Programmatic Reinforcement Learning. We conduct the programmatic reinforcement learning algorithm (Qiu & Zhu, 2022) to learn a programmatic controller. Compared to other programmatic reinforcement learning approaches, this algorithm stands out by jointly learning both program structures and program parameters. It relaxes the expansive discrete program structure search problem to efficiently learning the probabilistic distribution of high-reward program structures in the search space induced by a context-free grammar. It is completely automated and does not require user-provided oracles to seed imitation learning (Verma et al., 2018), or any other guidance. We present the details of this learning algorithm in Appendix. A.1.

Ideally, synthesizing an oscillator controller that satisfies the safety properties φ_{safe} and φ_{reach} can be achieved by shaping the reward function consistent with φ_{safe} and φ_{reach}, i.e., rewarding actions leading to states close to that specified in φ_{reach} and penalizing actions leading to states violating φ_{safe}. On the oscillator example, the learned controller $\pi(x_1, x_2)$ is given in Eq. 1 depicted in Fig. 3a. However, reward shaping does not lead to a safe oscillator controller after RL training (see Sec. 2.1).

2.1. Controller Verification

An alternative approach to reward shaping is constrained safe reinforcement learning (Moldovan & Abbeel, 2012; Turchetta et al., 2016). Most safe-RL algorithms specify safety specifications as a cost function in addition to an objective reward function (Achiam et al., 2017; Berkenkamp et al., 2017; Dalal et al., 2018; Le et al., 2019; Wen & Topcu, 2018; Tessler et al., 2019; Yang et al., 2020). For the Oscillator problem, our goal would be to train a controller π that maximizes the cumulative object reward $J^R(\pi)$ and bounds the amount of safety violation to φ_{safe} during an episode and φ_{reach} by the end of the episode under a threshold (e.g. zero) on some sampled rollouts. However, there is no formal safety guarantee on a learned controller. In our experience, applying state-of-the-art safe-RL algorithms (Achiam et al., 2017; Tessler et al., 2019; Yang et al., 2020) in the oscillator environment does not even lead to a controller that satisfies φ_{reach} on sampled episode rollouts.

In this paper, we instead formalize safe controller synthesis as a verification-based controller optimization problem. A synthesized controller π is certified by a program verifier against the safety properties φ_{safe} and φ_{reach}. The verifier returns true if π is verified safe. Otherwise, π is optimized to eliminate any verification counterexamples. We aim to synthesize a controller π that is verified safe.

To verify an STS (Definition. 2.2 state transition system) over an infinite set of initial states, we apply abstract interpretation to approximate the infinite set of system behaviors.

An STS $\mathcal{F}^{\delta}[\cdot]$ is treated as a discretization of a continuous system with sampling period δ. Discretization is needed for learning a controller π. In verification, we consider all states reachable by the original continuous system. Formally, we use S_i $(i > 0)$ to represent the set of reachable states in the time interval of $[(i-1)\delta, i\delta]$:

$$
S_i = \{\phi(s_0, t) \mid \forall s_0 \in S_0, \forall t \in [(i-1)\delta, i\delta]\}
$$

Definition 2.5 (STS Symbolic Rollout)**.** Given a state transition system $\mathcal{F}^\delta[\pi] = (F, \pi, S_0)$ and an abstract interpreter $\mathcal{D}$, a symbolic rollout of $\mathcal{F}^\delta$ over $\mathcal{D}$ is $S_0^\mathcal{D}, S_1^\mathcal{D}, \ldots$ where $S_0^\mathcal{D} = \alpha(S_0)$ is the abstraction of the initial states S_0. And $S_i^\mathcal{D} = \mathcal{F}_i^{\delta\,\mathcal{D}}[\pi]\big(\alpha(S_0)\big)$ over-approximates S_i all possible states reachable from an initial state $s_0 \in S_0$ at timestep i (or in the time interval of $[(i-1)\delta, i\delta]$).

We use Flow* (Chen et al., 2013), an abstract reachability analyzer for safety verification of continuous systems, to implement $\mathcal{F}_i^{\delta\,\mathcal{D}}$ to construct overapproximations of the set of states S_i at timestep i and the abstract domain $\mathcal{D}$ is flowpipes. Flow* works by constructing flowpipe overapproximations of the dynamics using Taylor Models, which scales well in practical applications.

Example 2.6. We conducted reachability analysis to compute an over-approximation for a reachable state set between each time interval within the episode horizon using the reachability analyzer Flow* to verify the continuous system composed by the oscillator ODE in Eq. 1 and the learned controller in Fig. 3a. The result is depicted in Fig. 3b. It can be seen that the controller does not entirely reach the goal set.

2.2. Controller Safety Loss

An unsafe rollout of a controller violates safety properties at some states. We define a safety loss function to quantify the safety violation of a state.

Definition 2.7 (Safety Loss Function)**.** For a safety property φ over states $s \in S$, we define a non-negative loss function $\mathcal{L}(s, \varphi)$ such that $\mathcal{L}(s, \varphi) = 0$ iff s satisfies φ, i.e. $s \models \varphi$. We define $\mathcal{L}(s, \varphi)$ recursively, based on the possible shapes of φ (Definition 2.3):

- $\mathcal{L}(s, \mathcal{A} \cdot x \leq b) := \max(\mathcal{A} \cdot s - b, 0)$
- $\mathcal{L}(s, \varphi_1 \wedge \varphi_2) := \max(\mathcal{L}(s, \varphi_1), \mathcal{L}(s, \varphi_2))$
- $\mathcal{L}(s, \varphi_1 \vee \varphi_2) := \min(\mathcal{L}(s, \varphi_1), \mathcal{L}(s, \varphi_2))$

Notice that $\mathcal{L}(s, \varphi_1 \wedge \varphi_2) = 0$ iff $\mathcal{L}(s, \varphi_1) = 0$ and $\mathcal{L}(s, \varphi_2) = 0$, which by construction is true if both φ_1 and φ_2 are satisfied by s, and similarly $\mathcal{L}(\varphi_1 \vee \varphi_2) = 0$ iff $\mathcal{L}(\varphi_1) = 0$ or $L(\varphi_2) = 0$.

Although it is possible to learn STS controller parameters solely via reinforcement learning using the safety loss function as a negative reward function, we do not have any formal safety guarantee of a learned controller. Instead, we aim to use verification feedback to improve controller safety. To this end, we lift the safety loss function over concrete states (Definition 2.7) to an *abstract safety loss* function for over-approximated abstract states.

Definition 2.8 (Abstract Safety Loss Function)**.** Given an abstract state $S^\mathcal{D}$ and a safety property φ, we define an abstract safety loss function as

$$\mathcal{L}_\mathcal{D}(S^\mathcal{D}, \varphi) = \max_{s \in \gamma(S^\mathcal{D})} \mathcal{L}(s, \varphi)$$

The abstract safety loss function applies γ to obtain all concrete states represented by an abstract state $S^\mathcal{D}$. It measures the worst-case safety loss of φ among all concrete states subsumed by $S^\mathcal{D}$. Given an abstract domain $\mathcal{D}$, we can usually approximate the concretization of an abstract state $\gamma(S^\mathcal{D})$ with a tight interval $\gamma_I(S^\mathcal{D})$. Especially, it is straitforward to represent flowpipes as intervals in Flow*. Based on the possible shape of φ, we can more efficiently compute $\mathcal{L}_\mathcal{D}(S^\mathcal{D}, \varphi)$ as:

- $\mathcal{L}_\mathcal{D}(S^\mathcal{D}, \mathcal{A} \cdot x \leq b) := \max_{s \in \gamma_I(S^\mathcal{D})} \big(\max(\mathcal{A} \cdot s - b, 0)\big)$
- $\mathcal{L}_\mathcal{D}(S^\mathcal{D}, \varphi_1 \wedge \varphi_2) := \max(\mathcal{L}_\mathcal{D}(S^\mathcal{D}, \varphi_1), \mathcal{L}_\mathcal{D}(S^\mathcal{D}, \varphi_2))$
- $\mathcal{L}_\mathcal{D}(S^\mathcal{D}, \varphi_1 \vee \varphi_2) := \min(\mathcal{L}_\mathcal{D}(S^\mathcal{D}, \varphi_1), \mathcal{L}_\mathcal{D}(S^\mathcal{D}, \varphi_2))$

Theorem 2.9 (Abstract Safety Loss Function Soundness)**.** *Given an abstract state $S^\mathcal{D}$ and a safety property φ, we have:*

$$\mathcal{L}_\mathcal{D}(S^\mathcal{D}, \varphi) = 0 \implies s \models \varphi \; \forall s \in \gamma_I(S^\mathcal{D}).$$

We further extend the definition for the safety of an abstract state to the safety of a symbolic rollout.

Definition 2.10 (Symbolic Rollout Safety Loss)**.** Given an STS $\mathcal{F}^\delta[\pi]$ with sampling period δ, its T-step symbolic rollout $S_0^\mathcal{D}, S_1^\mathcal{D}, S_2^\mathcal{D}, \ldots, S_{T-1}^\mathcal{D}, S_T^\mathcal{D}$ (Definition 2.5) satisfies the safety properties φ_{safe} and φ_{reach} iff there exists $T' \leq T$ such that:

$$\underbrace{S_0^\mathcal{D}, S_1^\mathcal{D}, S_2^\mathcal{D}, \ldots, S_{T'-1}^\mathcal{D}}_{\forall i \in 1 \ldots T'-1,\ \mathcal{L}_\mathcal{D}(S_i^\mathcal{D}, \varphi_{safe}) = 0} \quad \underbrace{S_{T'}^\mathcal{D}}_{\mathcal{L}_\mathcal{D}(S_{T'}^\mathcal{D}, \varphi_{reach}) = 0}$$

If the symbolic rollout is unsafe, the safety loss of $\mathcal{F}^\delta[\pi]$ is:

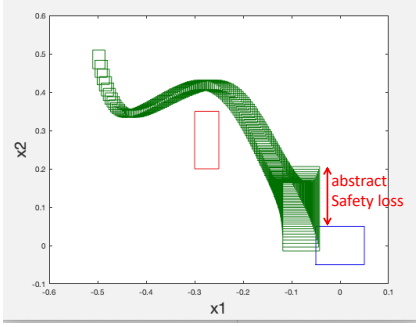
$$\mathcal{L}_\mathcal{D}(\mathcal{F}^\delta[\pi], \varphi_{safe}, \varphi_{reach}) = \mathcal{L}_\mathcal{D}(S_T^\mathcal{D}, \varphi_{reach}) + \sum_{t=1}^{T} \mathcal{L}_\mathcal{D}(S_t^\mathcal{D}, \varphi_{safe}) \quad (3)$$

Example 2.11. In Fig. 3b, there is a safety loss between the state abstraction at the last timestep and φ_{reach}, which we characterize as abstract safety loss, formally defined in Definition. 2.8.

Definition 2.10 specifies a sound verification procedure for STS, formalized below.

$$\textbf{if } 28.33x_1 + 4.23x_2 + 4.16 \geq 0$$
$$\textbf{then } 6.79x_1 + -8.56x_2 + 0.35$$
$$\textbf{else } 11.01x_2 + -13.50x_2 + 8.71$$

(a) Oscillator Programmatic Controller



(b) Oscillator Reachability Analysis

Figure 3. The oscillator programmatic controller and its reachability analysis. In Fig. 3b, the red box represents the oscillator unsafe set $[-03, -0.25] \times [0.2, 0.35]$, and the blue box depicts the goal set is $[-0.05, 0.05] \times [-0.05, 0.05]$. The initial set of oscillator is $[-0.51, -0.49] \times [0.49, 0.51]$.

Theorem 2.12 (Safety Verification Soundness)**.** *For an STS $\mathcal{F}^\delta[\pi]$ deployed with a controller π, $\mathcal{F}^\delta[\pi] \models \varphi_{safe}, \varphi_{reach}$ denotes that the deployed system satisfies safety properties φ_{safe} and φ_{reach}.*

$$\mathcal{L}_\mathcal{D}(\mathcal{F}^\delta[\pi], \varphi_{safe}, \varphi_{reach}) = 0 \implies \mathcal{F}^\delta[\pi] \models \varphi_{safe}, \varphi_{reach}.$$

2.3. Controller Synthesis

At a high level, our safe controller synthesis algorithm takes as input a programmatic controller learned after reinforcement learning converges and when verification fails uses abstract safety losses as verification back to improve controller safety. Intuitively, abstract states and abstract safety losses are parameterized by controller parameters. Thus, we can leverage a gradient-style optimization to update controller parameters by taking steps proportional to the negative of the gradient of the abstract safety losses. As opposed to standard gradient descent, we optimize controllers based on verification feedback in a proof space, favouring the verifier directly to construct a safe controller.

Controller Synthesis in the Proof Space. Synthesizing controllers in its proof space is critical to learning a verified controller because the verification procedure introduces approximation error and considers states in between each time interval, both of which cannot be observed by a reinforcement learning agent during training in the concrete state space. Even a well-trained controller may fail verification because of approximation error. Synthesis in the proof space leverages verification feedback on either true unsafe states or approximation error introduced by the verification procedure to search for a provably safe controller.

In the following, we deem a controller as a function $\pi(\theta)$ of its parameters θ (e.g. the parameters of a programmatic controller in Fig. 2). We abbreviate $\pi(\theta)$ as π_θ for simplicity. Given an STS $\mathcal{F}^\delta[\pi_\theta]$ with sampling period δ, the abstract safety loss function $\mathcal{L}_\mathcal{D}$ of $\mathcal{F}^\delta[\pi_\theta]$ is essentially a function of π_θ, or more specifically, a function of π_θ's parameters θ. To reduce the abstract safety loss of π_θ, we leverage a gradient-descent style optimization to update θ by taking steps proportional to the negative of the gradient of $\mathcal{L}_\mathcal{D}$ at θ. As opposed to standard gradient descent, we optimize π_θ based on symbolic rollouts produced by the controller in the proof space for $\mathcal{F}^\delta[\pi_\theta]$, favouring the abstract interpreter (i.e., Flow*) directly for verification-based controller updates.

Black-box Gradient Estimation. Although a verification procedure such as Flow* may not be differentiable or a third-party implementation does not allow differentiation, we effectively estimate gradients based on random search (Mania et al., 2018). Given an STS $\mathcal{F}^\delta[\pi_\theta]$, at each training iteration, we obtain perturbed STSs $\mathcal{F}^\delta[\pi_{\theta+\nu\omega}]$ and $\mathcal{F}^\delta[\pi_{\theta-\nu\omega}]$ where we add sampled Gaussian noise ω to the current controller π_θ's parameters θ in both directions and ν is a small positive real number. By evaluating the abstract safety losses of the symbolic rollouts of $\mathcal{F}^\delta[\pi_{\theta+\nu\omega}]$ and $\mathcal{F}^\delta[\pi_{\theta-\nu\omega}]$, we update the controller parameters θ with a finite difference approximation along an unbiased estimator of the gradient:

$$\nabla_\theta \mathcal{L}_\mathcal{D} \leftarrow \frac{1}{N}\sum_{k=1}^{N} \frac{\begin{pmatrix}\mathcal{L}_\mathcal{D}(\mathcal{F}^\delta[\pi_{\theta+\nu\omega_k}],\ \varphi_{safe},\ \varphi_{reach}) - \\ \mathcal{L}_\mathcal{D}(\mathcal{F}^\delta[\pi_{\theta-\nu\omega_k}], \varphi_{safe}, \varphi_{reach})\end{pmatrix}}{\nu}\omega_k$$

To achieve verified safety, with a verification-based controller gradient $\nabla_\theta \mathcal{L}_\mathcal{D}$, we update controller parameters θ as below where η is a learning rate:

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}_\mathcal{D}$$

We repeatedly perform such a gradient-based update to optimize controller parameters until a verifiably safe STS controller is synthesized.

Such a verification-based controller update only improves the safety of a controller. It does not incorporate the reward function R that may additionally include performance requirements. We address this shortcoming in Appendix. A.2.

3. Experimental Results

We have implemented our verification-based safe controller synthesis algorithm in a tool called Marvel. Marvel takes

	ReachNN*			Marvel	
#	layers	hidden nodes	time	iterations	time
1	3	20	26s	1	62s
2	3	20	5s	1	0.2s
3	3	20	94s	1	20s
4	3	20	8s	21	68s
5	4	100	103s	1	3s
6	4	20	1126s	2	7s

Table 1. Comparison with ReachNN*. The dimensions of states are from 2 to 4 for these benchmarks. Time of ReachNN* shows the runtime of the reachability analysis of the tool. Time of Marvel shows the runtime of both reachability analysis and verification-guided controller synthesis.

a converged programmatic controller π learned by the programmatic reinforcement learning algorithm (Qiu & Zhu, 2022) as input and optimizes the controller if it fails verification with the verification-guided training algorithm.

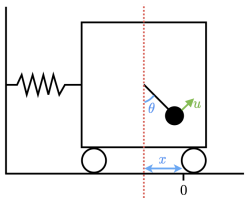
ReachNN* benchmarks. We first provide a full comparison between Marvel and ReachNN* (Fan et al., 2020), a state-of-the-art formal verification tool for neural network controlled systems on all the examples in (Fan et al., 2020). ReachNN* solves the reachability problem of a provided neural network controller by verifying if the controller can drive the system under control to reach a goal region. Marvel verifies if a programmatic controller can be used to reach the same goal region and additionally *learns* to improve the controller with verification feedback if the controller fails verification. Other than the first benchmark, Marvel runs much faster despite having the additional learning procedure. Marvel's results are averaged over 5 random seeds. On benchmark 4, the controller by programmatic reinforcement learning cannot be verified safe and Marvel takes averagely 21 iterations to synthesize a safe one. Our results demonstrate that it is computationally challenging to have deep neural network verification repeatedly in a reinforcement learning training loop as each learning iteration would be very expensive. On the other hand, programmatic controllers make verification-guided training feasible.

We further evaluated Marvel on several *nonlinear* continuous cyber-physical systems with known environment models taken from the ARCH-COMP21 competition on formal verification of Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants. We are unable to use ReachNN* to verify trained neural network controllers on theses systems.

Adaptive Cruise Control. The first benchmark, Adaptive Cruise Control, involves an ego vehicle and a lead vehicle with 6 variables representing the position, velocity and acceleration of the two vehicles. Our training objective is to learn a controller that when the lead vehicle suddenly reduces its speed, the ego car can decelerate to maintain a safe distance. φ_{safe} specifies the minimum relative distance that a controller should maintain at each timestep as well as an upper bound to prevent the ego car from stopping. A rollout lasts 50 timesteps and each timestep is $\delta = 0.1s$. Fig. 4a depicts the training performance of Marvel where we show the abstract safety loss at each training iteration. Marvel learned a safe controller because it uses verification feedback to directly optimize the worst-case safety loss in the proof space. Previous work (Tran et al., 2020) used verification to detect safety issues for a well-trained controller for this system. Our result shows that verification can also be used for controller safe-by-construction.

Oscillator. This is our running example in Sec. 2. The training curve for abstract safety losses is depicted in Fig. 4b.

Tora. The Tora (translational oscillations by a rotational actuator) model (depicted on the right) involves a cart that is attached to a wall with a spring, and is free to move on a friction-less surface. The cart itself has a weight attached to an arm inside it, which is free to rotate about an axis. This serves as the control input for stabilizing the cart at x = 0. The model is a 4 dimensional system, given by the following differential equations:



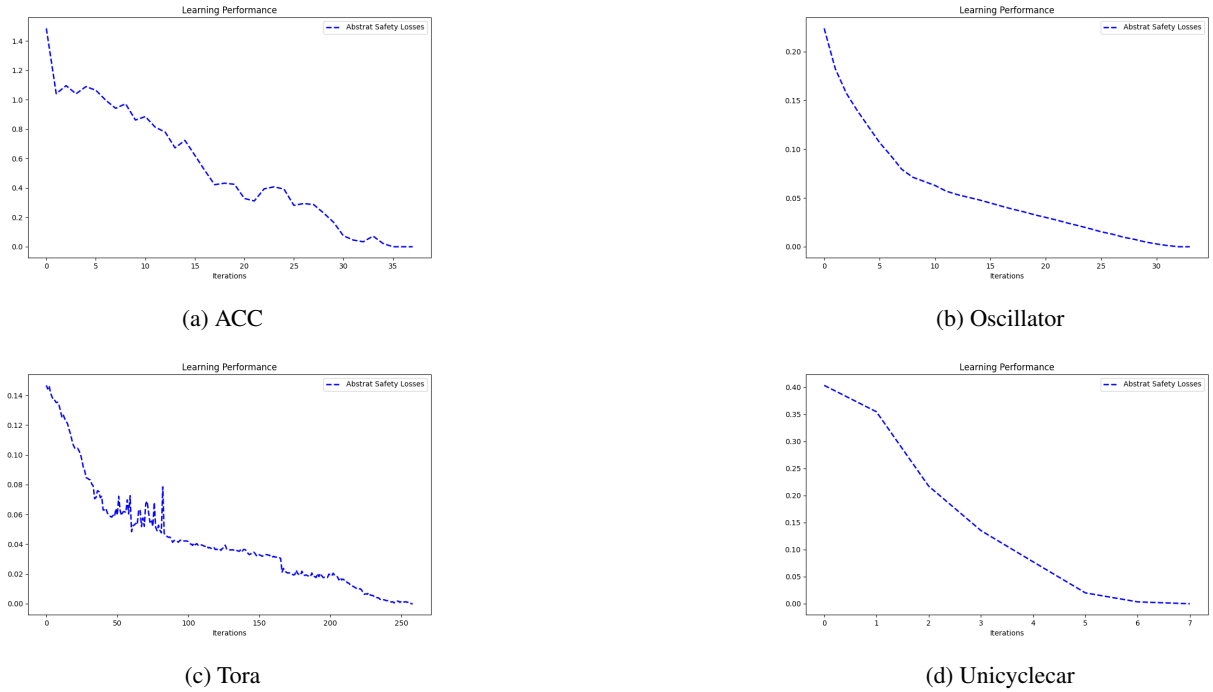
$$\dot{x_1} = x_2 \qquad \dot{x_2} = -x_1 + 0.1 \cdot sin(x_3)$$
$$\dot{x_3} = x_4 \qquad \dot{x_4} = a$$

The verification problem in the competition is that for an initial set of $x_1 \in [0.6, 0.7]$, $x_2 \in [-0.7, -0.6]$, $x_3 \in [-0.4, -0.3]$, and $x4 \in [0.5, 0.6]$, the system states stay within the box $x \in [-2, 2]^4$, for a bounded time window (20s). Marvel can easily verify this property. To make the problem more interesting, we added a new specification for φ_{reach} as an inductive invariant, requiring that any rollout starting from φ_{reach} eventually turns back to it and any rollout states must be safe (including those that temporarily leave φ_{reach}). We set the sampling period $\delta = 0.01s$. The training curve of abstract safety losses is depicted in Fig. 4c.

Unicyclecar. The unicycle car benchmark can be expressed by the following dynamics equations:

$$\dot{x}_1 = x_4 \cos(x_3) \qquad \dot{x}_2 = x_4 \sin(x_3)$$
$$\dot{x}_3 = a_2 \qquad \dot{x}_4 = a_1 + w$$

where w is a random bounded error in the range $[-1e-4, 1e-4]$. In our setting, we set the sampling period $\delta = 0.05s$ and the total time is $5s$ (100 control steps). The initial set is $[9.5, 9.55] \times [-4.5, -4.45] \times [2.1, 2.11] \times [1.5, 1.51]$. The goal set is $[-0.6, 0.6] \times [-0.2, 0.2] \times [-0.06, 0.06] \times$



(a) ACC (b) Oscillator

(c) Tora (d) Unicyclecar

Figure 4. Marvel reduces the abstract safety loss on ACC, Oscillator, Tora, and Unicyclecar to zero.

$[-0.3, 0.3]$. The training curve of abstract safety losses is depicted in Fig. 4d.

On all the four benchmarks, the reinforcement learning algorithm (Qiu & Zhu, 2022) can obtain high reward controllers. However, these controllers cannot be directly verified safe due to approximation error. Marvel optimizes these controllers on top of the proof space and generates verifably safe controllers.

4. Related Work

Robust Machine Learning. Our work on safe controller synthesis is inspired by the recent advances in verifying neural network robustness, e.g. (Gehr et al., 2018; Anderson et al., 2019; Singh et al., 2019; Weng et al., 2018). These approaches apply abstract transformers to relax nonlinearity of activation functions in a neural network into convex representations, based on linear approximation (Wong & Kolter, 2018; Weng et al., 2018; Singh et al., 2018; 2019; Zhang et al., 2020) or interval approximation (Gowal et al., 2018; Mirman et al., 2018). Since the abstractions are differentiable, neural networks can be optimized toward tighter concertized bounds to improve verified robustness (Mirman et al., 2018; Balunovic & Vechev, 2020; Zhang et al., 2020; Wang et al., 2018; Lin et al., 2020).

Recent work (Tran et al., 2020; Ivanov et al., 2018; Sun et al., 2019; Fan et al., 2020; Dutta et al., 2019) achieved initial results about verifying learning-enabled autonomous systems. However, these approaches are expensive and do not attempt to perform verification within a training loop.

Safe Reinforcement Learning. Most safe-RL algorithms form a constraint optimization problem specifying safety specifications as a cost function in addition to an objective reward function (Achiam et al., 2017; Berkenkamp et al., 2017; Dalal et al., 2018; Le et al., 2019; Wen & Topcu, 2018; Tessler et al., 2019; Yang et al., 2020). Their goal is to train a controller that maximizes the accumulated reward and bounds the amount of aggregate safety violation under a threshold. In contrast, Marvel ensures that a learned controller is verified safe with respect to an environment model and can better handle reachability constraints beyond safety.

Revel (Anderson et al., 2020) introduces a mechanism to learn neural-symbolic RL agents to ensure safe exploration during training. It can synthesize adaptive safety shields for complex neural network controllers. This is achieved by projecting a neural network controller onto a search space defined by a domain specific language in which it is possible to construct verifiably safe programmatic controllers. Similar approaches (Verma et al., 2018; Zhu et al., 2019; Bastani et al., 2018) synthesize a symbolic program to approximate an RL controller based on imitation learning (Ross et al., 2011) and show that the symbolic program has better interpretability and safety. Instead, Marvel directly synthesize safe controllers by integrating verification into training.

5. Conclusion

We present Marvel that bridges program synthesis and verification for controller safe-by-construction. Our experiments show that verification-guided controller updates can lead to verifiably safe controllers. We plan to extend Marvel to support controller safety during exploration. When a worst-case environment model is provided, this can be achieved by performing a controller update to maximize the long-term reward and then reconciling safety violation by projecting it back onto the verified safe space (Chow et al., 2019).

Acknowledgements

This work was supported by NSF under award CCF-SHF 2007799.

References

Achiam, J., Held, D., Tamar, A., and Abbeel, P. Constrained policy optimization. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 22–31. PMLR, 2017. URL http://proceedings.mlr.press/v70/achiam17a.html.

Anderson, G., Pailoor, S., Dillig, I., and Chaudhuri, S. Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019.*, pp. 731–744, 2019. doi: 10.1145/3314221.3314614. URL https://doi.org/10.1145/3314221.3314614.

Anderson, G., Verma, A., Dillig, I., and Chaudhuri, S. Neurosymbolic reinforcement learning with formally verified exploration. *CoRR*, abs/2009.12612, 2020. URL https://arxiv.org/abs/2009.12612.

Åström, K. J. and Hägglund, T. Automatic tuning of simple regulators with specifications on phase and amplitude margins. *Autom.*, 20(5):645–651, 1984. doi: 10.1016/0005-1098(84)90014-1. URL https://doi.org/10.1016/0005-1098(84)90014-1.

Balunovic, M. and Vechev, M. T. Adversarial training and provable defenses: Bridging the gap. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Bastani, O., Pu, Y., and Solar-Lezama, A. Verifiable reinforcement learning via policy extraction. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 2499–2509, 2018.

Berkenkamp, F., Turchetta, M., Schoellig, A. P., and Krause, A. Safe model-based reinforcement learning with stability guarantees. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 908–918, 2017.

Chen, X., Ábrahám, E., and Sankaranarayanan, S. Flow*: An analyzer for non-linear hybrid systems. In Sharygina, N. and Veith, H. (eds.), *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pp. 258–263. Springer, 2013. doi: 10.1007/978-3-642-39799-8_18. URL https://doi.org/10.1007/978-3-642-39799-8_18.

Chow, Y., Nachum, O., Faust, A., Ghavamzadeh, M., and Duéñez-Guzmán, E. A. Lyapunov-based safe policy optimization for continuous control. *CoRR*, abs/1901.10031, 2019. URL http://arxiv.org/abs/1901.10031.

Cousot, P. and Cousot, R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, pp. 238–252, 1977. doi: 10.1145/512950.512973. URL https://doi.org/10.1145/512950.512973.

Dalal, G., Dvijotham, K., Vecerík, M., Hester, T., Paduraru, C., and Tassa, Y. Safe exploration in continuous action spaces. *CoRR*, abs/1801.08757, 2018. URL http://arxiv.org/abs/1801.08757.

Dutta, S., Chen, X., and Sankaranarayanan, S. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In Ozay, N. and Prabhakar, P. (eds.), *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019*, pp. 157–168. ACM, 2019. doi: 10.1145/3302504.3311807. URL https://doi.org/10.1145/3302504.3311807.

Fan, J., Huang, C., Chen, X., Li, W., and Zhu, Q. Reachnn*: A tool for reachability analysis of neural-network controlled systems. In Hung, D. V. and Sokolsky, O. (eds.), *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, volume 12302 of *Lecture Notes in Computer Science*, pp. 537–542. Springer, 2020. doi: 10.1007/978-3-030-59152-6_30. URL https://doi.org/10.1007/978-3-030-59152-6_30.

Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., and Vechev, M. T. AI2: safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pp. 3–18, 2018. doi: 10.1109/SP.2018.00058. URL https://doi.org/10.1109/SP.2018.00058.

Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Arandjelovic, R., Mann, T. A., and Kohli, P. On the effectiveness of interval bound propagation for training verifiably robust models. *CoRR*, abs/1810.12715, 2018. URL http://arxiv.org/abs/1810.12715.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007. ISBN 978-0-321-47617-3.

Inala, J. P., Bastani, O., Tavares, Z., and Solar-Lezama, A. Synthesizing programmatic policies that inductively generalize. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=S1l8oANFDH.

Ivanov, R., Weimer, J., Alur, R., Pappas, G. J., and Lee, I. Verisig: verifying safety properties of hybrid systems with neural network controllers. *CoRR*, abs/1811.01828, 2018. URL http://arxiv.org/abs/1811.01828.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

Kakade, S. A natural policy gradient. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pp. 1531–1538, Cambridge, MA, USA, 2001. MIT Press.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Le, H. M., Voloshin, C., and Yue, Y. Batch policy learning under constraints. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3703–3712. PMLR, 2019. URL http://proceedings.mlr.press/v97/le19a.html.

Lin, X., Zhu, H., Samanta, R., and Jagannathan, S. Art: Abstraction refinement-guided training for provably correct neural networks. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*, pp. 148–157. IEEE, 2020. doi: 10.34727/2020/isbn.978-3-85448-042-6_22. URL https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_22.

Mania, H., Guy, A., and Recht, B. Simple random search of static linear policies is competitive for reinforcement learning. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 1805–1814, 2018.

Mirman, M., Gehr, T., and Vechev, M. T. Differentiable abstract interpretation for provably robust neural networks. In Dy, J. G. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3575–3583. PMLR, 2018. URL http://proceedings.mlr.press/v80/mirman18b.html.

Moldovan, T. M. and Abbeel, P. Safe exploration in markov decision processes. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012. URL http://icml.cc/2012/papers/838.pdf.

Platt, J. C. and Barr, A. H. Constrained differential optimization. In *Neural Information Processing Systems*, NIPS'88, pp. 612–621, 1988.

Qiu, W. and Zhu, H. Programmatic reinforcement learning without oracles. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=6Tk2noBdvxt.

Ross, S., Gordon, G. J., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In Gordon, G. J., Dunson, D. B., and Dudík, M. (eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pp. 627–635. JMLR.org, 2011. URL http://proceedings.mlr.press/v15/ross11a/ross11a.pdf.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. Trust region policy optimization. In Bach, F. R. and Blei, D. M. (eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 1889–1897. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/schulman15.html.

Singh, G., Gehr, T., Mirman, M., Püschel, M., and Vechev, M. T. Fast and effective robustness certification. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 10825–10836, 2018.

Singh, G., Gehr, T., Püschel, M., and Vechev, M. T. An abstract domain for certifying neural networks. *Proc. ACM Program. Lang.*, 3(POPL):41:1–41:30, 2019. doi: 10.1145/3290354. URL https://doi.org/10.1145/3290354.

Sun, X., Khedr, H., and Shoukry, Y. Formal verification of neural network controlled autonomous systems. In Ozay, N. and Prabhakar, P. (eds.), *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019*, pp. 147–156. ACM, 2019. doi: 10.1145/3302504.3311802. URL https://doi.org/10.1145/3302504.3311802.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In Solla, S. A., Leen, T. K., and Müller, K. (eds.), *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pp. 1057–1063. The MIT Press, 1999.

Tessler, C., Mankowitz, D. J., and Mannor, S. Reward constrained policy optimization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=SkfrvsA9FX.

Tran, H., Yang, X., Lopez, D. M., Musau, P., Nguyen, L. V., Xiang, W., Bak, S., and Johnson, T. T. NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In Lahiri, S. K. and Wang, C. (eds.), *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*, volume 12224 of *Lecture Notes in Computer Science*, pp. 3–17. Springer, 2020. doi: 10.1007/978-3-030-53288-8_1. URL https://doi.org/10.1007/978-3-030-53288-8_1.

Trivedi, D., Zhang, J., Sun, S.-H., and Lim, J. J. Learning to synthesize programs as interpretable and generalizable policies. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=wP9twkexC3V.

Turchetta, M., Berkenkamp, F., and Krause, A. Safe exploration in finite markov decision processes with gaussian processes. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 4305–4313, 2016.

Verma, A., Murali, V., Singh, R., Kohli, P., and Chaudhuri, S. Programmatically interpretable reinforcement learning. In Dy, J. G. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5052–5061. PMLR, 2018. URL http://proceedings.mlr.press/v80/verma18a.html.

Wang, S., Chen, Y., Abdou, A., and Jana, S. Mixtrain: Scalable training of formally robust neural networks. *CoRR*, abs/1811.02625, 2018. URL http://arxiv.org/abs/1811.02625.

Wang, Y., Huang, C., Wang, Z., Wang, Z., and Zhu, Q. Verification in the loop: Correct-by-construction control learning with reach-avoid guarantees. *CoRR*, abs/2106.03245, 2021. URL https://arxiv.org/abs/2106.03245.

Wen, M. and Topcu, U. Constrained cross-entropy method for safe reinforcement learning. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 7461–7471, 2018.

Weng, T., Zhang, H., Chen, H., Song, Z., Hsieh, C., Daniel, L., Boning, D. S., and Dhillon, I. S. Towards fast computation of certified robustness for relu networks. In Dy, J. G. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5273–5282. PMLR, 2018. URL http://proceedings.mlr.press/v80/weng18a.html.

Winskel, G. *The formal semantics of programming languages - an introduction*. Foundation of computing series. MIT Press, 1993. ISBN 978-0-262-23169-5.

Wong, E. and Kolter, J. Z. Provable defenses against adversarial examples via the convex outer adversarial polytope. In Dy, J. G. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5283–5292. PMLR, 2018. URL http://proceedings.mlr.press/v80/wong18a.html.

Yang, T., Rosca, J., Narasimhan, K., and Ramadge, P. J. Projection-based constrained policy optimization. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=rke3TJrtPS.

Zhang, H., Chen, H., Xiao, C., Gowal, S., Stanforth, R., Li, B., Boning, D. S., and Hsieh, C. Towards stable and efficient training of verifiably robust neural networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=Skxuk1rFwB.

Zheng, F., Wang, Q., and Lee, T. H. On the design of multivariable PID controllers via LMI approach. *Autom.*, 38(3):517–526, 2002. doi: 10.1016/S0005-1098(01)00237-0. URL https://doi.org/10.1016/S0005-1098(01)00237-0.

Zhu, H., Xiong, Z., Magill, S., and Jagannathan, S. An inductive synthesis framework for verifiable reinforcement learning. In McKinley, K. S. and Fisher, K. (eds.), *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*, pp. 686–701. ACM, 2019. doi: 10.1145/3314221.3314638. URL https://doi.org/10.1145/3314221.3314638.

A. Appendix

A.1. Programmatic Reinforcement Learning

Qiu & Zhu (2022) expresses RL controllers as *differentiable* programs, which use symbolic language constructs to compose a set of parameterized primitive modules. To control an agent, a programmatic controller takes an environment state as input and computes an action as return for the agent to execute.

A programmatic controller can be viewed as a pair (E, θ), where E is a discrete program structure and θ is a vector of real-valued parameters of the program. A program structure E is structured based on the context-free grammar (Hopcroft et al., 2007) of a controller DSL. The context-free grammar is depicted in the standard Backus-Naur form (Winskel, 1993) in Fig. 5. A vertical bar "|" indicates choice. Such a grammar consists of a set of production rules $X ::= \sigma_1\ \sigma_2\ \cdots\ \sigma_j$ where X is a nonterminal and $\sigma_1, \cdots, \sigma_j$ are either terminals or nonterminals. For example, one may expand the nonteriminal E_1 in a partial program **if** B_1 **then** C_1 **else** E_1 to **if** B_1 **then** C_1 **else** (**if** B_2 **then** C_2 **else** E_2). The nonterminals E and B stand for program expressions that evaluate to action values in $\mathbb{R}^m$ and Booleans, respectively, where m is the action dimension size. A state input to a programmatic controller is represented as $s = \{x_1 : \nu_1, x_2 : \nu_2, \ldots, x_n\}$ where n is the state dimension size and $\nu_i = s[x_i]$ is the value of x_i in s. As usual, the unbounded variables in $\mathcal{X} = [x_1, x_2, \ldots, x_n]$ are assumed to be input variables (state variables in our context). A terminal in this grammar is a symbol that can appear in a program's code, e.g. the *if* symbol and x_i.

$$E\ ::= C \mid \textbf{if}\ B\ \textbf{then}\ C\ \textbf{else}\ E$$
$$B\ ::= \theta_c + \theta^T \cdot \mathcal{X} \geq 0$$

Figure 5. A Context-free DSL Grammar for programmatic controllers.

The semantics of a program in E is mostly standard and given by a function $[\![E]\!](s)$, defined for each DSL construct. For example, $[\![x_i]\!](s) = s[x_i]$ reads the value of a variable x_i in a state input s. A controller may use an **if-then-else** branching construct. To avoid discontinuities for differentiability, as discussed in Sec. 2, its semantics is encoded in terms of a smooth approximation where σ is the sigmoid function:

$$[\![\textbf{if}\ B\ \textbf{then}\ C\ \textbf{else}\ E]\!](s) = \sigma([\![B]\!](s)) \cdot [\![C]\!](s) + (1 - \sigma([\![B]\!](s))) \cdot [\![E]\!](s)$$

Thus, any controller programmed in this grammar becomes a differentiable program. C is a controller used by a programmatic controller. During execution, the controller can invoke a set of controllers under different environment conditions, according to the activation of B conditions in the program. Qiu & Zhu (2022) consider three DSLs depending on how C is structured for *affine*, *ensemble*, and *PID* controllers.

Affine controllers. As shown in Sec. 2, The DSL for affine controllers allows C to be expanded as an affine transformation:

$$C_{\textit{Affine}}\ ::=\ \theta_c + \theta \cdot \mathcal{X} \mid \theta_c$$

where $\theta \in \mathbb{R}^{m \cdot |\mathcal{X}|}, \theta_c \in \mathbb{R}^m$ are controller parameters. Particularly, $C_{\textit{Affine}}$ can be as simple as some (learned) constants θ_c.

Ensemble controllers. The DSL supports compositionality — composing and reusing task-agnostic primitives in new programs to solve novel problems. The DSL for ensemble controllers includes pre-acquired primitives $\pi_1, \cdots, \pi_N$ as callable library functions:

$$C_\pi\ ::=\ \theta_1 \cdot \pi_1(s) + \theta_2 \cdot \pi_2(s) + \cdots + \theta_N \cdot \pi_N(s)$$

C_π explicitly compose primitive functions (e.g. running forward or jumping) hierarchically into a complex program (e.g. jumping over multiple hurdles to reach a target) where $\theta_1, \cdots, \theta_N \in \mathbb{R}^1$ parameterize a primitive combination. The input space of a primitive function can be different from that of a program (formally defined below). The semantics of C_π is defined as follows:

$$[\![C_\pi]\!](s) = \sum_{i=0}^{N} q_i \cdot \pi_i(s) \text{ where } q_i = \frac{\exp(\theta_i/T)}{\sum_{j=0}^{N} \exp(\theta_j/T)}$$

Here the composition weights $\{q_i\}_{i=0}^N$ for primitive ensemble are computed using gumbel-softmax, where T is the temperature term (Jang et al., 2017).

PID controllers. Suppose that PID control is known a priori suitable for stabilising of an RL system. One can express this knowledge using the DSL for PID functions that allows C to be expanded as discretized, multivariable PID controllers

(Zheng et al., 2002):

$$C_{PID} ::= \textbf{PID}_{\theta_P,\theta_I,\theta_D}(\epsilon, h, s) \mid \theta_c$$

where θ_P, θ_I, $\theta_D \in \mathbb{R}^{m\cdot n}$ are parameters representing the proportional gain, integral gain, and derivative gain matrices of PID control. Notice that a PID controller additionally takes a known constant ϵ that represents a fixed target for the controller to stabilize the system under control around, and a history h of a sequence of states before the current control step. The semantics of the controller is as follows:

$$\begin{aligned} [\![\textbf{PID}_{\theta_P,\theta_I,\theta_D}]\!](\epsilon, h, s) =& \theta_P \cdot P + \theta_I \cdot I + \theta_D \cdot D \text{ where} \\ P = (\epsilon - s) \qquad I =& \ \textbf{fold}(+, \epsilon - h) \qquad D = \textbf{peek}(h, -1) - s \end{aligned} \tag{4}$$

In the semantics definition, P is the proportional term, I is the discrete approximation of the integral term (calculated via a **fold**), and D is the finite-difference approximation of the derivative term. In line with the standard integral error reset strategy (Åström & Hägglund, 1984), the **fold** function acts over a fixed-sized window on the history (e.g. the five latest states of the history). $\textbf{peek}(h, -1)$ returns the most resent state in a history h.

Problem Formulation. Qiu & Zhu (2022) frame programmatic RL as a Markov Decision Process (MDP) defined by a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}\}$ where $\mathcal{S}$ and $\mathcal{A}$ represent the environment state space and action space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ captures the set of transition probabilities, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the reward function. Assume $\mathcal{S} \supseteq \mathbb{R}^{|\mathcal{X}\cup\mathcal{V}|}$ where $\mathcal{X}$ is the set of input variables of a composite program (defined by a DSL) and $\mathcal{V}$ is the set of input variables of primitive functions. For an affine controller, $\mathcal{V} = \emptyset$. At time $t \geq 0$, an RL agent receives an environment state $s_t \in \mathcal{S}$ and performs an action $a_t \in \mathcal{A}$ selected by its controller $\pi(a_t|s_t) : \mathcal{S} \to \mathcal{A}$. Based on s_t and a_t, the agent transits to receive the next state according to the transition model $\mathcal{T}(s_{t+1}|s_t, a_t)$, and receives the reward $R(s_t, a_t)$. Qiu & Zhu (2022) learn a programmatic controller π in the DSL in Fig. 5 by jointly synthesizing the program's structure E and optimizing the program's parameters θ to maximize the cumulative discounted reward $\mathbb{E}_{s_0,a_0,s_1\cdots\sim\pi}\big[\sum_0^\infty \gamma^t \cdot R(s_t, a_t)\big]$ where $\gamma \in (0, 1]$.

Learning Algorithm. The main idea is to relax the controller structure search space to be continuous. This amounts to collectively optimizing the probability distribution of all program structures in the search space and assigning the the highest probability to the structure that maximizes cumulative MDP reward.

The algorithm is not specific to a DSL. It takes as input any controller DSL with differentiable semantics and conducts controller structure search on a *program derivation tree* of the DSL. Formally, a program derivation tree is $\mathcal{T} = \{V, \mathcal{E}\}$ where a node $u \in V$ contains partial structures with missing expressions or a complete structure permissible by the DSL. An edge $(u, u_E) \in \mathcal{E}$ exists if one can obtain the structures in u_E by expanding a nonterminal E within a partial structure in u following some DSL production rules. If more than one rule can be applied to expand the nonterminal E, u_E contains more than one structure. Fig. 6 depicts a program derivation tree for the DSL in Fig. 5 where a controller C is an ensemble controller. On the root node 0, one has two choices to expand the initial nonterminal E_1 to either an ensemble controller C_1 or a partial structure if B_1 then C_2 else E_2. Node 1 thus contains two partial structures. Formally, $\mathcal{F}(u_E)$ represents the set of structures on a node u_E.

To expand a nonterminal or a missing expression of a partial program structure, Qiu & Zhu (2022) relaxes the categorical choice of DSL production rules into a softmax over all possible production rules for the missing expression with trainable weights. For example, on node 1, the choices to expand E_1 between the ensemble controller C_1 and the conditional branching expression are weighted by the weight matrix w_1 (obtained after softmax) drawn in Fig. 6. Based on w_1, one chooses to expand E_1 to the conditional branching expression on node 1. Assume E_2 is further expanded on node 1 to a conditional branching expression as well on node 5. Then again one has two choices to expand the nonterminal E_3 on node 5 weighted by w_2. This time E_3 is expanded to an ensemble controller C_5. Formally, the weight matrix w_{u_E} of the incoming edge to a node u_E is of the shape $\mathbb{R}^{|\mathcal{F}(u_E)|}$, and $w_{u_E}[E']$ weighs the likelihood of choosing a particular structure $E' \in \mathcal{F}(u_E)$ for expanding E.

A program derivation tree $\mathcal{T}$ essentially expresses all possible program derivations up to a certain bound on the depth of program abstract syntax trees. To train structure weights, Qiu & Zhu (2022) encode a program derivation tree itself as a differentiable program $\pi^{\mathcal{T}}_{\theta,w}$ that takes a state s as input. Its action output is weighted by the outputs of all programs included in $\pi^{\mathcal{T}}_{\theta,w}$, where w represents program structure weights and θ includes unknown program parameters of all the mixed programs in the tree. The semantics computation of an expression $[\![E]\!](s)$ in a program derivation tree $\pi^{\mathcal{T}}_{\theta,w}$ is delegated to its tree node u_E where the nonterminal E is expanded and the categorical choice of expanding E on u_E is

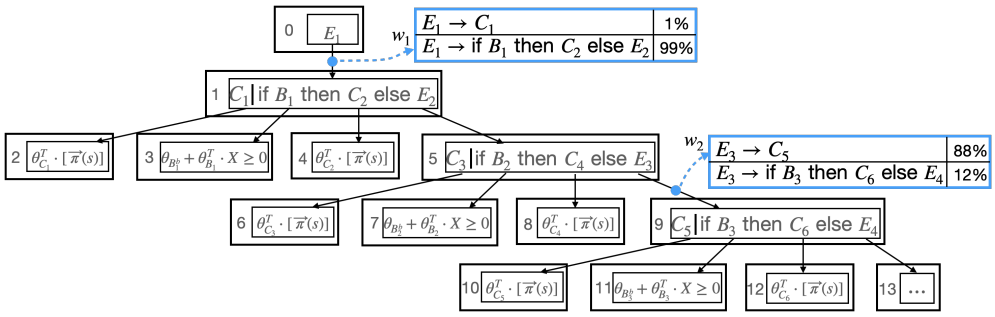


Figure 6. Ant Cross Maze Program Derivation Tree with program input $\mathcal{X}$. $\vec{\pi}$ refers to the primitives of Ant moving up π_{UP}, down π_{DOWN}, left π_{LEFT} and right π_{RIGHT} that take the Ant's own observations.

relaxed to a softmax over all possible choices:

$$[\![E]\!](s) = [\![u_E]\!](s) \qquad [\![u_E]\!](s) = \sum_{E' \in \mathcal{F}(u_E)} \frac{\exp(w_{u_E}[E'])}{\sum_{E'' \in F(u_E)} \exp(w_{u_E}[E''])} \cdot [\![E']\!](s)$$

Complexity. Assume that the root of $\mathcal{T}$ hosts the initial DSL nonterminal $E_{\mathcal{T}}$, d is the depth of $\mathcal{T}$, k is the number of DSL production rules, and m is the maximum number of nonterminals in the body of any rules. The semantics of $\pi^{\mathcal{T}}_{\theta,w}$ is defined as $[\![\pi^{\mathcal{T}}_{\theta,w}]\!](s) = [\![E_{\mathcal{T}}]\!](s)$. The number of DSL operations (e.g. evaluations of ensemble controllers and Boolean conditions) invoked by $[\![E_{\mathcal{T}}]\!](\cdot)$ is bounded by $O((km)^d)$.

Controller Optimization Objective. The parameters w and θ of a program derivation tree $\pi^{\mathcal{T}}_{\theta,w}$ can be jointly optimized using any controller gradient methods. To obtain stochastic controller gradients, $\pi^{\mathcal{T}}_{\theta,w}(\cdot|s)$ is encoded as a Gaussian controller where the tree program outputs the action distribution mean. A separate set of parameters specify the (diagonal) distribution covariance. Qiu & Zhu (2022) use trust region methods e.g. (Schulman et al., 2015) to maximize the "surrogate" objective function, subject to a constraint on the size of the controller update by δ, where $\rho_{\pi^{\mathcal{T}}_{\theta_{old},w_{old}}}$ is the discounted state visitation frequency of $\pi^{\mathcal{T}}_{\theta_{old},w_{old}}$, $A_{\pi^{\mathcal{T}}_{\theta_{old},w_{old}}}$ is an estimator of the advantage function over a finite batch of samples from $\pi^{\mathcal{T}}_{\theta_{old},w_{old}}$ and θ_{old}, w_{old} are controller parameters and structure weights before the update:

$$\begin{aligned} &\text{maximize}_{\theta,w}\ J_{\theta_{old},w_{old}}(\theta,w) = \mathbb{E}_{s\sim\rho_{\pi^{\mathcal{T}}_{\theta_{old},w_{old}}},a\sim\pi^{\mathcal{T}}_{\theta_{old},w_{old}}}\left[\frac{\pi^{\mathcal{T}}_{\theta,w}(s,a)}{\pi^{\mathcal{T}}_{\theta_{old},w_{old}}(s,a)} A_{\pi^{\mathcal{T}}_{\theta_{old},w_{old}}}(s,a)\right] \\ &\text{subject to } \mathbb{E}_{s\sim\rho_{\pi^{\mathcal{T}}_{\theta_{old},w_{old}}}}\left[D_{KL}(\pi^{\mathcal{T}}_{\theta_{old},w_{old}}(\cdot|s)\ ||\ \pi^{\mathcal{T}}_{\theta,w}(\cdot|s))\right] \leq \delta \end{aligned} \tag{5}$$

Controller Parameter Optimization. The training algorithm is an iterative bilevel optimization procedure. At training iteration k, Qiu & Zhu (2022) perform two steps. At the first step, the lower-level program parameters θ are optimized with respect to (5), freezing the upper-level structure weights w:

$$\theta_{k+1} = \arg\max_{\theta} J_{\theta_k,w_k}(\theta,w_k) \text{ s.t. } \mathbb{E}_{s\sim\rho_{\pi^{\mathcal{T}}_{\theta_k,w_k}}}\left[D_{KL}(\pi^{\mathcal{T}}_{\theta_k,w_k}(\cdot|s)\ ||\ \pi^{\mathcal{T}}_{\theta_{k+1},w_k}(\cdot|s))\right] \leq \delta \tag{6}$$

Controller structure Optimization. At the second step, the upper-level structure weights w are optimized with respect to (5), freezing the lower-level program parameters θ:

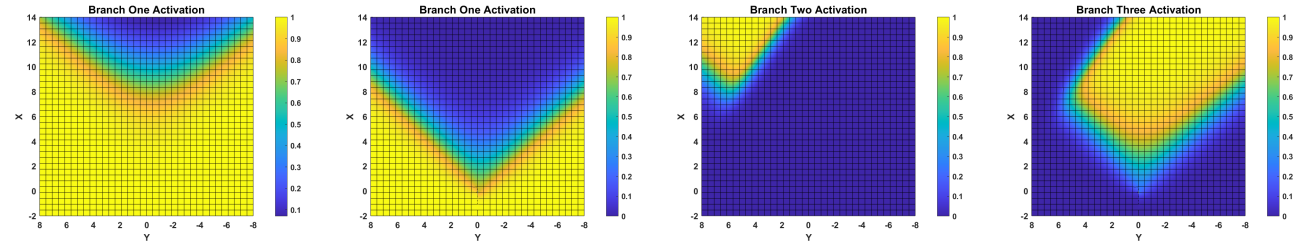
$$w_{k+1} = \arg\max_{w} J_{\theta_{k+1},w_k}(\theta_{k+1},w) \text{ s.t. } \mathbb{E}_{s\sim\rho_{\pi^{\mathcal{T}}_{\theta_{k+1},w_k}}}\left[D_{KL}(\pi^{\mathcal{T}}_{\theta_{k+1},w_k}(\cdot|s)\ ||\ \pi^{\mathcal{T}}_{\theta_{k+1},w_{k+1}}(\cdot|s))\right] \leq \delta \tag{7}$$

Training steps (6) and (7) are alternated across training iterations until reward convergence. They can be approximately solved using the efficient conjugate gradient algorithm, after making a linear approximation to the objective and a quadratic approximation to the constraint (Schulman et al., 2015). Upon convergence, based on structure weights, the algorithm obtains a discrete program structure from $\pi^{\mathcal{T}}_{\theta,w}$ replacing each tree node containing multiple structures with the most likely structure in a top-down manner. Finally, the parameters in the chosen structure are trained using RL (Schulman et al., 2015) until convergence from the parameter values learned by the structure search process.

$$
\begin{aligned}
&\textbf{if } \theta_{1_c} + \theta_1^T \cdot \mathcal{X} > 0 \\
&\quad \textbf{then } (\, 95\% \cdot \pi_{\text{UP}}(s) + 5\% \cdot \pi_{\text{LEFT}}(s)) \longleftarrow \textbf{Branch 1} \\
&\quad \textbf{else if } \theta_{2_c} + \theta_2^T \cdot \mathcal{X} > 0 \\
&\qquad\quad \textbf{then } (\, 95\% \cdot \pi_{\text{LEFT}}(s) + 5\% \cdot \pi_{\text{RIGHT}}(s)) \longleftarrow \textbf{Branch 2} \\
&\qquad\quad \textbf{else } (\, 13\% \cdot \pi_{\text{DOWN}}(s) + 87\% \cdot \pi_{\text{RIGHT}}(s)) \longleftarrow \textbf{Branch 3}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{X} &= [\, x,\ y,\ \mathcal{G}_x,\ \mathcal{G}_y, arctan\tfrac{y}{x}, \|x, y\|_2 \,] \\
\theta_1 &= [\, -2.052,\ 0.049,\ 0.440,\ 0.181,\ 0.241,\ 1.443 \,],\ \theta_{1_c} = -0.202 \\
\theta_2 &= [\, 1.333,\ 2.204, -2.2171,\ 2.132,\ 1.878,\ 0.331 \,],\ \theta_{2_c} = -0.416
\end{aligned}
$$

Figure 8. An Ant Cross Maze program $\mathcal{P}_{cross}$ with three branches. A program input $\mathcal{X}$ includes current Ant position x, y along with the target location $\mathcal{G}_x$, $\mathcal{G}_y$ (sampled from one of the three goals in Fig. 7). $arctan\frac{y}{x}$ and $\|x, y\|_2$ are functions of x and y. Each branch composes primitive functions: π_{UP}, π_{DOWN}, π_{LEFT}, and π_{RIGHT}. Composition weights are shown in percentage.



(a) Branch 1 activation, $(\mathcal{G}_x, \mathcal{G}_y)$ = (12, 0)

(b) Branch 1 activation, $(\mathcal{G}_x, \mathcal{G}_y)$ = (6, -6)

(c) Branch 2 activation, $(\mathcal{G}_x, \mathcal{G}_y)$ = (6, -6)

(d) Branch 3 activation, $(\mathcal{G}_x, \mathcal{G}_y)$ = (6, -6)

Figure 9. Branch activation as functions of Ant position (x, y) for program $\mathcal{P}_{cross}$.

Program Interpretability. Thanks to the structured and symbolic representation of a learned programmatic controller, it is highly interpretable. For example, consider an Ant Cross Maze environment depicted in Fig. 7. The maze contains three possible goal positions and one would be randomly selected at each time. In this environment, the task for a quadruped MuJoCo Ant is to reach the selected location by navigating through the maze staring from an initial position on the bottom and without collision or crash. Consider the DSL with ensemble controllers C_π for this task. Assume it includes four basic primitive functions (which are pretrained) for moving the Ant up π_{UP}, down π_{DOWN}, left π_{LEFT}, and right π_{RIGHT}.

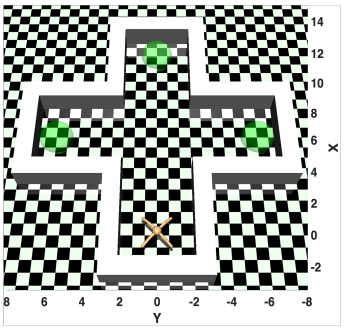


Figure 7. Ant Cross Maze

Fig. 8 depicts a synthesized program $\mathcal{P}_{cross}$ with three branches for solving the Ant Cross Maze environment. As specified in Equation 2, the semantics of a branching construct is approximated by the sigmoid function σ. The value of the predicate in a Boolean condition determines the activation of the controller guarded by the Boolean condition. At each state, branch activation determines the strength of each of the controllers in the program. For example, the activation of branch 1 is $\sigma(\theta_{1_c} + \theta_1^T \cdot \mathcal{X})$, and the activation of branch 2 is $(1 - \sigma(\theta_{1_c} + \theta_1^T \cdot \mathcal{X})) \cdot \sigma(\theta_{2_c} + \theta_2^T \cdot \mathcal{X})$.

Fig. 9a depicts the activation of branch 1 as a function of (x, y) when the goal to reach is sampled at $\mathcal{G}_x = 12, \mathcal{G}_y = 0$. The degree of activation (yellow) is close to 1 on all states under (12, 0) indicating that the ensemble controller at branch 1 is used to drive the Ant up to the goal. Indeed, according to the distribution of each primitive function at branch 1, the effect of π_{UP} dominates. Fig. 9b, Fig. 9c, and Fig. 9d depict the activation of all three branches when the goal is at $\mathcal{G}_x = 6, \mathcal{G}_y = -6$. The program can be interpreted as branch 1 (where π_{UP} dominates) and branch 3 (where π_{RIGHT} dominates) are activated in the yellow areas of Fig. 9b and Fig. 9d respectively. This allows the Ant to make a curved up and right move to the goal (branch 2 is not activated during execution for this goal).

A.2. Integrating Both Safety and Rewards

We integrate reward signals R from environments to achieve both controller reward performance and controller safety:

$$\max_{\theta} J^R(\pi_\theta)$$
$$\textit{subject to } \mathcal{L}_{\mathcal{D}}(\mathcal{G}[\pi_\theta], \varphi_{safe}, \varphi_{reach}) = 0 \tag{8}$$

where an environment model $\mathcal{G}$ is fitted with respect to a current controller π_θ. A constrained optimization problem as such can be solved by the Lagrangian method (Platt & Barr, 1988) that firstly reduces the constrained problem to an equivalent unconstrained optimization problem with an adaptive penalty coefficient λ:

$$\max_{\theta} \min_{\lambda \geq 0} J^R(\pi_\theta) - \lambda \mathcal{L}_{\mathcal{D}}(\mathcal{G}[\pi_\theta], \varphi_{safe}, \varphi_{reach}) \tag{9}$$

The Lagrangian method then solves the unconstrained optimization problem by performing gradient descent on λ with a small fixed learning rate η (e.g. $\eta = 5e^{-2}$):

$$\lambda \leftarrow \lambda - \eta \cdot \big(- \mathcal{L}_{\mathcal{D}}(\mathcal{G}(\pi_\theta), \varphi_{safe}, \varphi_{reach})\big)$$

and performing gradient ascent on θ to maximize the cumulative reward $J^R(\pi_\theta)$ and the negative abstract safety loss $\mathcal{L}(\mathcal{G}(\pi_\theta), \varphi_{safe}, \varphi_{reach})$.

To simplify the presentation, in the following we fix $\lambda = 1$ as a constant and only show how to optimize controller parameters θ. Specifically, to optimize the cumulative reward $J^R(\pi_\theta)$, we consider the standard model-free vanilla controller gradient $\nabla_\theta J^R(\pi_\theta)$ (Sutton et al., 1999). This method directly optimizes controller parameters θ to maximize the cumulative reward using local search via the samples of the controller π_θ. To optimize the abstract safety loss $\mathcal{L}_{\mathcal{D}}(\mathcal{G}(\pi_\theta), \varphi_{safe}, \varphi_{reach})$, we use the verification-based controller gradient $\nabla_\theta \mathcal{L}_{\mathcal{D}}$ defined in Sec. 2.3.

With the controller gradients, we could simply update controller parameters θ using the fixed learning rate η:

$$\theta \leftarrow \theta + \eta \cdot \big(\nabla_\theta J^R(\pi_\theta) - \nabla_\theta \mathcal{L}_{\mathcal{D}}(\mathcal{G}(\pi_\theta), \varphi_{safe}, \varphi_{reach})\big)$$

However, a controller update as such can change the behavior of π_θ arbitrarily different from the old one at each training iteration. Because the fitted TVLG dynamics $\mathcal{G}[\cdot]$ is a local model and only valid in a local region of the state space around the sampled rollouts of the old controller, an unconstrained parameter update can cause the new controller to visit part of the state space where $\mathcal{G}[\cdot]$ is arbitrarily incorrect and unstable, leading to divergence. Therefore, our approach restricts controller update at each iteration to maintain the validity of a fitted time-varying linear model $\mathcal{G}[\cdot]$. Inspired by a few recently developed RL algorithms (Kakade, 2001; Schulman et al., 2015), we limit the controller change by imposing a constraint on the KL-divergence between the old controller $\pi_{\theta_{old}}$ and the new controller π_θ by a threshold ξ (e.g. $\xi = 1e^{-2}$):

$$\max_{\theta} J^R(\pi_\theta) - \mathcal{L}_{\mathcal{D}}(\mathcal{G}[\pi_\theta], \varphi_{safe}, \varphi_{reach})$$
$$\textit{subject to } \mathbb{E}_{s \sim d^{\pi_{\theta_{old}}}}[D_{KL}(\pi_{\theta_{old}}(\cdot|s) \,||\, \pi_\theta(\cdot|s))] \leq \xi \tag{10}$$

where $d^{\pi_{\theta_{old}}}$ is the state distribution under the controller $\pi_{\theta_{old}}$ estimated in the actual (true) environment by sampling[2] and the KL divergence D_{KL} is the expectation of the logarithmic difference between the probabilities of $\pi_{\theta_{old}}$ and π_θ:

$$D_{KL}(\pi_{\theta_{old}} \,||\, \pi_\theta) = \mathbb{E}_{x \sim \pi_{\theta_{old}}}\Big(\log \frac{\pi_{\theta_{old}}(x)}{\pi_\theta(x)}\Big)$$

here the expectation is taken using the probabilities of $\pi_{\theta_{old}}$. Intuitively, D_{KL} measures how the probability distribution $\pi_{\theta_{old}}$ is different from a new controller π_θ.

The constrained optimization problem (10) can be analytically solved (similar to (Kakade, 2001; Schulman et al., 2015)). We update θ as follows. The derivation of this solution is explained below.

$$\theta = \theta_{old} + \sqrt{\frac{2\xi}{g^T H(\theta_{old})^{-1} g}} H(\theta_{old})^{-1} g \tag{11}$$
$$\textit{where } g = \big(\nabla_\theta J^R(\pi_\theta) - \nabla_\theta \mathcal{L}_{\mathcal{D}}(\mathcal{G}[\pi_\theta], \varphi_{safe}, \varphi_{reach})\big)|_{\theta=\theta_{old}}$$

[2] $d^\pi(s) = P(s_0 = s) + \beta P(s_1 = s) + \beta^2 P(s_2 = s) + \cdots$ where $s_0 \in S_0$, and $a_t \sim \pi(\cdot|s_t)$, $s_{t+1} \sim P(s_{t+1}|s_t, a_t)\ \forall t \geq 0$. If $\beta = 1$, $d_{\pi_{\theta_{old}}}$ is just the state visit frequency under $\pi_{\theta_{old}}$. P is the true transition probability distribution.

$H(\theta_{old})$ is a Hessian matrix that measures the second-order derivative of the log probability of $\pi_{\theta_{old}}$ (after extending the KL divergence definition). Importantly, as opposed to improving controller parameters simply using the abstract safety loss gradient $\nabla_\theta \mathcal{L}_\mathcal{D}$ and the controller gradient $\nabla_\theta J^R$, the update rule in Equation (11) uses an additional KL divergence threshold ξ to restrain controller updates that can turn destructive to the validity of the fitted TVLG model $\mathcal{G}[\cdot]$. For sample efficiency, since $H(\theta_{old})$ is estimated based on sampled states from the old controller $\pi_{\theta_{old}}$, we can reuse the samples used to fit the TVLG model $\mathcal{G}[\cdot]$ for estimating $H(\theta_{old})$.

In the following, we abbreviate $\mathcal{L}_\mathcal{D}(\mathcal{G}[\pi_\theta], \varphi_{safe}, \varphi_{reach})$ as a function $\mathcal{L}^\mathcal{D}_\mathcal{G}(\theta)$ parameterized by controller parameters θ to ease the presentation. Essentially the abstract safety loss of a controller with an abstract interpreter $\mathcal{D}$ depends upon its parameters. Similarly, we abbreviate $J^R(\pi_\theta)$ as a function $J^R(\theta)$ parameterized by controller parameters θ. To solve the constrained optimization problem (10), similar to (Schulman et al., 2015), we use a linear approximation to $\mathcal{L}^\mathcal{D}_\mathcal{G}(\theta)$ and $J^R(\theta)$ and a quadratic approximation to the D_{KL} constraint, based on first-order Taylor expansion:

$$\mathcal{L}^\mathcal{D}_\mathcal{G}(\theta) \approx \mathcal{L}^\mathcal{D}_\mathcal{G}(\theta_{old}) + \nabla_\theta \mathcal{L}^\mathcal{D}_\mathcal{G}(\theta)|_{\theta=\theta_{old}} \cdot (\theta - \theta_{old}) \tag{12}$$

$$J^R(\theta) \approx J^R(\theta_{old}) + \nabla_\theta J^R(\theta)|_{\theta=\theta_{old}} \cdot (\theta - \theta_{old}) \tag{13}$$

$$\begin{aligned} D_{KL}(\pi_{\theta_{old}} \,||\, \pi_\theta) \approx\ & D_{KL}(\pi_{\theta_{old}} \,||\, \pi_{\theta_{old}}) + \\ & \nabla_\theta D_{KL}(\pi_{\theta_{old}} \,||\, \pi_\theta)|_{\theta=\theta_{old}} \cdot (\theta - \theta_{old}) + \\ & \frac{1}{2}(\theta - \theta_{old})^T \cdot \nabla^2_\theta D_{KL}(\pi_{\theta_{old}} \,||\, \pi_\theta)|_{\theta=\theta_{old}} \cdot (\theta - \theta_{old}) \end{aligned} \tag{14}$$

By canceling the constants in (12), (13), and (14), the optimization problem (10) can be approximated as:

$$\max_\theta \nabla_\theta J^R(\theta)|_{\theta=\theta_{old}} \cdot (\theta - \theta_{old}) - \nabla_\theta \mathcal{L}^\mathcal{D}_\mathcal{G}(\theta)|_{\theta=\theta_{old}} \cdot (\theta - \theta_{old}) \tag{15}$$

$$\textit{s.t. } \frac{1}{2}(\theta - \theta_{old})^T \cdot H(\theta_{old}) \cdot (\theta - \theta_{old}) \le \xi$$

$$\textit{where } H(\theta_{old}) = E_{s\sim d_{\pi_{\theta_{old}}}} \nabla^2_\theta D_{KL}(\pi_{\theta_{old}(\cdot|\ s)} \,||\, \pi_\theta(\cdot|\ s))|_{\theta=\theta_{old}}$$

$H(\theta_{old})$ is a Hessian matrix that measures the second-order derivative of the log probability of $\pi_{\theta_{old}}$ (after extending the KL divergence definition).

Since H is the Fisher Information matrix, which automatically guarantees it is positive semi-definite. Therefore, it is a convex program with quadratic inequality constraints. Hence if the primal problem has a feasible point, then Slaters condition is satisfied and strong duality holds. Let θ^* and $\lambda*$ denote the solutions to the primal and dual problems, respectively. In addition, the primal objective function is continuously differentiable. Hence the Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient for the optimality of θ^* and $\lambda*$. In the following, let

$$g = \nabla_\theta J^R(\theta)|_{\theta=\theta_{old}} - \nabla_\theta \mathcal{L}^\mathcal{D}_\mathcal{G}(\theta)|_{\theta=\theta_{old}}$$

We now form the Lagrangian to solve (15):

$$\mathcal{L}(\theta, \lambda) = -g^T(\theta - \theta_{old}) + \lambda(\frac{1}{2}(\theta - \theta_{old})^T \cdot H(\theta_{old}) \cdot (\theta - \theta_{old}) - \xi)$$

And we have the following KKT conditions:

$$-g + \lambda^* H\theta^* - \lambda^* H\theta_{old} = 0 \qquad \nabla_\theta \mathcal{L}(\theta^*, \lambda^*) = 0 \tag{16}$$

$$\frac{1}{2}(\theta^* - \theta_{old})^T H(\theta^* - \theta_{old}) - \xi = 0 \qquad \nabla_\lambda \mathcal{L}(\theta^*, \lambda^*) = 0 \tag{17}$$

$$\frac{1}{2}(\theta^* - \theta_{old})^T H(\theta^* - \theta_{old}) - \xi \le 0 \qquad \text{primal constraints} \tag{18}$$

$$\lambda^* \ge 0 \qquad \text{dual constraints} \tag{19}$$

$$\lambda^*(\frac{1}{2}(\theta^* - \theta_{old})^T H(\theta^* - \theta_{old}) - \xi) = 0 \qquad \text{complementary slackness} \tag{20}$$

By (16), we have $\theta* = \theta_{old} + \frac{1}{\lambda^*} H^{-1} g$. And by plugging (16) into (17), we have $\lambda^* = \sqrt{\frac{g^T H^{-1} g}{2\xi}}$. Hence we have our

optimal solution:

$$\theta = \theta^* = \theta_{old} + \sqrt{\frac{2\xi}{g^T H^{-1} g}} H^{-1} g, \tag{21}$$

which also satisfied (18), (19), and (20). Putting everything together, we derive

$$\theta = \theta_{old} + \sqrt{\frac{2\xi}{g^T H(\theta_{old})^{-1} g}} H(\theta_{old})^{-1} g$$
$$\textit{where } g = \nabla_\theta J^R(\theta)|_{\theta=\theta_{old}} - \nabla_\theta \mathcal{L}^{\mathcal{D}}_{\mathcal{G}}(\theta)|_{\theta=\theta_{old}}$$