Reward Generation for Model-Free Reinforcement Learning from Formal Specifications

NIKHIL SINGH, IIT Kanpur, India
INDRANIL SAHA, IIT Kanpur, India

Deep Reinforcement Learning (DRL) has the potential to be used for synthesizing feedback controllers (agents) for various complex systems with unknown dynamics. These systems are expected to satisfy diverse safety and liveness properties best captured using temporal logic. In RL, the reward function plays a crucial role in specifying the desired behaviour of these agents. However, the problem of designing the reward function for an RL agent to satisfy complex temporal logic specifications has received limited attention in the literature. To address this, we provide a systematic way of generating rewards in real-time by using the quantitative semantics of Signal Temporal Logic (STL), a widely used temporal logic to specify the behaviour of cyber-physical systems. We propose a new quantitative semantics for STL having several desirable properties, making it suitable for reward generation. We evaluate our STL-based reinforcement learning mechanism on several complex continuous control benchmarks and compare our STL semantics with those available in the literature in terms of their efficacy in synthesizing the controller agent. Experimental results establish our new semantics to be the most suitable for synthesizing feedback controllers for complex continuous dynamical systems through reinforcement learning.

JAIR Track: Integration of Logical Constraints in Deep Learning

JAIR Associate Editor: Insert JAIR AE Name

JAIR Reference Format:
Nikhil Singh and Indranil Saha. 2025. Reward Generation for Model-Free Reinforcement Learning from Formal Specifications. *Journal of Artificial Intelligence Research* 4, Article 6 (August 2025), 28 pages. DOI: 10.1613/jair.1.xxxxx

1 Introduction

Feedback controllers form the core of any safety-critical cyber-physical system (CPS). The traditional approaches for synthesizing feedback controllers rely on the availability of a mathematical model of the dynamical system whose behaviour the controller is supposed to regulate. However, for complex dynamical systems, the creation of a faithful mathematical model poses a tremendous challenge to the control engineers. Reinforcement learning (RL) provides an alternative for synthesizing feedback controllers in the form of a controller agent for complex systems without precise mathematical models. The agent interacts with the dynamical system in a simulation environment providing a faithful but complex system model that is not suitable for controller synthesis using traditional control-theoretic procedures.

A deep neural network can be used as the agent in RL, and in that case, the learning procedure is called the *Deep Reinforcement Learning* (DRL). In recent times, DRL has been extremely popular in solving highly complex problems such as playing Atari 2600 games [Mnih et al.(2015)] and Go [Silver et al.(2017)] with super-human performance, making BiPedal Robots walk [Lillicrap et al.(2016)], learning visuomotor controllers for robots [Levine et al.(2016)], and many more. The success of DRL in solving those complex problems is attributed

Authors' Contact Information: Nikhil Singh, ORCID: 0000-0002-8057-9934, nksingh@cse.iitk.ac.in, IIT Kanpur, Kanpur, UP, India; Indranil Saha, ORCID: 0000-0002-1329-8286, isaha@cse.iitk.ac.in, IIT Kanpur, Kanpur, UP, India.

DOI: 10.1613/jair.1.xxxxx

to well-defined reward functions. Thus, designing correct reward functions [Sutton and Barto(2018)] is extremely important for synthesizing DRL-based controllers.

Several papers have considered the RL-based methods for synthesizing feedback controllers (e.g. [Lillicrap et al.(2016), Levine and Koltun(2013), Deisenroth et al.(2013), Fulton and Platzer(2018), Hafner and Riedmiller(2011)]). In these approaches, the reward function is designed by a control engineer having complete knowledge of the system dynamics. As the system becomes high-dimensional and highly nonlinear, designing a correct reward function in this manner becomes prohibitively hard. For control engineers, it would have been significantly more convenient if they could write the specification of the closed-loop system in a formal language, and the reward could be generated automatically from this specification.

In the recent past, Signal Temporal Logic (STL) [Donzé and Maler(2010)] has been used widely in capturing real-time specifications for synthesizing controllers for complex CPSs [Singh and Saha(2021), Raman et al.(2014), Raman et al.(2015)]. The robustness semantics of STL makes it a potential candidate for specification-based reward generation in controller synthesis through DRL. An STL-based reward can efficiently perform *temporal aggregation*, which is difficult to achieve in a hand-crafted reward function. However, the classical quantitative semantics [Donzé and Maler(2010), Jakšić et al.(2018)] of STL often leads to improper rewards, which in turn may lead to the synthesis of sub-optimal controllers. This is because the point-based classical semantics suffer from the *shadowing problem* [Várnai and Dimarogonas(2020)], where an increase in the robustness of an individual sub-specification does not influence the robustness of the full specification computed by the AND operator, except for the case when it is minimum.

Of late, researchers have proposed several new semantics of STL that attempt to incorporate aggregate notions of robustness instead of point-based estimates provided by the classical semantics. Some of these semantics such as `AGM` [Mehdipour et al.(2019)] and `SoftMax` [Várnai and Dimarogonas(2020)] were designed to address the shadowing problem. However, as the robustness functions for these semantics are not smooth, they are not well-suited for RL-based controller synthesis. Another semantics `LSE` [Li et al.(2018), Pant et al.(2017)], aims to provide a smooth approximation to the robustness function, but it does not address the shadowing problem. Due to the limitations of the existing aggregation-based semantics, STL has not yet been adopted for DRL-based controller synthesis despite its tremendous potential.

In this work, we propose a new aggregate-based semantics for STL, called `SSS` (Smoothened-Summation-Subtraction). Our Semantics is not sound, but we show that soundness is not an essential requirement of an STL semantics when it is used for reward generation in the DRL process. Rather, our semantics have all the essential properties for being qualified as a reward generation mechanism — it offers smoothness, addresses the shadowing problem, and ensures min-max boundedness.

We implement our semantics in the online monitoring tool RTAMT [Ničković and Yamaguchi(2020)] and evaluate it on several challenging continuous control benchmarks available in the `gym` environment [Brockman et al.(2016)]. For each of those benchmarks, we introduce a suitable STL specification that captures the safety and liveness requirements of the system. While safety requirements ensure that system behavior always remain within safe region of state space, liveness ensures that system makes continuous progress. We compare the performance of the controller synthesized using our semantics with those synthesized with other aggregate-based semantics available in the literature. Experimental results establish that our semantics consistently outperforms all the other semantics by a significant margin.

This paper extends our previous work [Singh and Saha(2023)] published in AAAI 2023 in several dimensions. First, in this version, we include a discussion on the intuition behind the design of the SSS semantics so that it becomes successful in reward generation for RL-based controller synthesis for complex dynamical systems. Second, we provide formal proofs of all the theorems presented in the conference paper. We have also added Corollary 1 that provides the bounds for robustness computed by the SSS semantics. Third, we present additional experiments for analyzing (a) the effect of hyper-parameters, (b) the effect of training for longer duration, and

(c) the effect of tuning the parameters involved in the predicates within the STL specification on the controller synthesis using the SSS semantics.

The rest of the paper is organized as follows. In Section 2, we present the background for this work. In Section 3, we define the controller synthesis problem and describe various metrics for controller evaluation. In Section 4, we describe the controller synthesis method using Deep RL. In Section 5, we present our proposed semantics for Signal Temporal Logic, the intuition behind it and various useful properties satisfied by it along with the proofs. In Section 6, we describe our experimental setup including benchmarks, hyper-parameters, specifications, etc. along with the results of our experiments. In Section 7, we conclude this paper along with directions for future research.

2 Preliminaries

This section presents a brief overview of the key concepts used in our work.

2.1 Closed-loop Control System

Open-loop dynamical systems (a.k.a. plants) often do not satisfy desired specifications. Control engineers design a feedback controller C for an open-loop system $\mathcal{P}$ to ensure that the closed-loop system $\mathcal{M}$ satisfies its specification. We assume that the set of states of the closed-loop system is fully observable and thus is the same as the set of outputs, denoted by $\mathcal{X}$. The output of the system (x_t) is used to generate control input (u_t) that is applied to the system at every time step t to regulate its behavior. A trace $\omega = \langle x_0, u_0\rangle, \langle x_1, u_1\rangle, \ldots$ is defined as the sequence of alternating states and control inputs of the system evolving in discrete time-steps, where x_t and u_t, $t \in \mathbb{N}$, refer to the state (output) and control input at the t-th time-step. We use $\mathcal{L}(\mathcal{M})$ to denote the set of all traces of $\mathcal{M}$.

2.2 Signal Temporal Logic

Signal Temporal Logic (STL) [Donzé and Maler(2010)] enables us to reason about the real-time properties of signals (simulation traces). These specifications consist of real-time predicates over the signal variables.

The syntax of an STL specification ϕ is defined by the grammar

$$\phi = \mathsf{true} \mid \pi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \, U_I \, \phi_2, \tag{1}$$

where $\pi \in \Pi$, Π is a set of atomic predicates defined on the outputs of the closed-loop system, and $I \subseteq \mathbb{R}^+$ is an arbitrary interval of non-negative real numbers. The operators $\neg$ and $\wedge$ denote logical NOT and AND operators. Other logical operators like logical OR ($\vee$) and implication ($\implies$) can be derived using $\neg$ and $\wedge$. The temporal operator U_I is the *until* operator implying that ϕ_2 becomes true sometime in the time interval I and ϕ_1 must remain true until ϕ_2 becomes true. There are two other useful temporal operators, namely *eventually* ($\Diamond_I$) and *always* ($\Box_I$), which can be derived from the temporal and logical operators defined above. The formula $\Diamond_I \, \phi$ means that the formula ϕ will be true sometime in the time interval I. The formula $\Box_I \, \phi$ means that the formula ϕ will always be true in the time interval I.

Robust Semantics of STL. : We follow the robust semantics of STL provided in [Donzé et al.(2013)]. We use Euclidean metric as the norm to measure the distance d between two values $v, v' \in \mathbb{R}$, i.e., $d(v, v') = \| \, v - v' \, \|$. Let $v \in \mathbb{R}$ be a value, $A \subseteq \mathbb{R}$ be a set. Then the *signed distance* from v to A is defined as:

$$\mathtt{Dist}(v, A) = \begin{cases} inf\{d(v, v') \mid v' \notin A\} & \text{if } v \in A, \\ -inf\{d(v, v') \mid v' \in A\} & \text{if } v \notin A. \end{cases} \tag{2}$$

Intuitively, $\mathtt{Dist}(v, A)$ captures how far a value v is from the violation of the inclusion in the set A. In both cases, we search for the minimum distance between v and a point on the boundary of A. Also, the case $v \notin A$ refers to a violation. Thus, the negative sign is used in the definition.

We use $\mathrm{O} : \mathcal{K} \to 2^X$ to denote the mapping of a predicate κ to a set of states (X). Given a trace ω and the mapping O, we define the robust semantics of ω w.r.t. ϕ at time $t \in \mathbb{R}$, denoted by $\rho(\phi, \omega, t)$, by induction as follows:

$$\rho(\mathsf{true}, \omega, t) = +\infty, \tag{3a}$$
$$\rho(\kappa, \omega, t) = \mathtt{Dist}(x_t, \mathrm{O}(\kappa)), \tag{3b}$$
$$\rho(\neg\phi, \omega, t) = -\rho(\phi, \omega, t), \tag{3c}$$
$$\rho(\phi \wedge \psi, \omega, t) = \mathtt{min}(\rho(\phi, \omega, t), \rho(\psi, \omega, t)), \tag{3d}$$
$$\rho(\phi\, U_I\, \psi, \omega, t) = \sup_{t' \in t+I} \mathtt{min}(\rho(\psi, \omega, t'), \inf_{t'' \in [t,t']} \rho(\phi, \omega, t'')). \tag{3e}$$

The robustness of a trace ω w.r.t a specification ϕ is defined as $\rho(\phi, \omega) = \rho(\phi, \omega, 0)$, i.e., the robustness at time 0. If $\rho(\phi, \omega, t) \neq 0$, its sign indicates the satisfaction status. The robustness metric ρ maps each simulation trace ω to a real number ρ. Intuitively, the robustness of a trace $\omega \in \mathcal{L}(\mathcal{M})$ with respect to an STL formula ϕ is the radius of the largest ball centered at trace ω that we can fit within $\mathcal{L}_\phi$, where $\mathcal{L}_\phi$ is the set of all signals satisfying ϕ.

The semantics described above is often referred to as *classical* semantics. Apart from this, other popular semantics of STL are based on Arithmetic-Geometric mean [Mehdipour et al.(2019)], Logarithm-summation-exponential [Li et al.(2018), Pant et al.(2017)] and Softmax [Várnai and Dimarogonas(2020)].

2.3 Online Monitoring of STL

The standard computation of STL robustness is offline, i.e., the signal values over the entire time horizon are available. In order to train DRL-based controllers, we require online computation of the robustness of the traces (signals). An online monitor takes as input the signal value at each time instant, and it computes the robustness of the signal at that time instant w.r.t a given specification. Here, the specifications can contain future temporal operators (like eventually, always, etc.), which are impossible to evaluate until the end of the episode. Hence, these specifications are converted into some equi-satisfiable form which postpones the evaluation of the formula from the current time to the end of horizon [Ničković and Yamaguchi(2020)]. Intuitively, the online monitor continuously computes the robustness of specification w.r.t. each new data coming in.

For online robustness $\bar{\rho}$ computation, the definition of $\Box_{[a,b]}$, $\Diamond_{[a,b]}$ and $U_{[a,b]}$ operator in the classical robustness semantics of STL is given as follows:

$$\bar{\rho}(\Box_{[a,b]}\, \phi, \omega, t) = \min_{t' \in [t+a, t+b]} \bar{\rho}(\phi, \omega, t'), \tag{4}$$

$$\bar{\rho}(\Diamond_{[a,b]}\, \phi, \omega, t) = \max_{t' \in [t+a, t+b]} \bar{\rho}(\phi, \omega, t'), \tag{5}$$

$$\bar{\rho}(\phi\, U_{[a,b]}\, \psi, \omega, t) = \max_{t' \in [t+a, t+b]} \left(\mathtt{min}(\bar{\rho}(\psi, \omega, t'), \min_{t'' \in [t', t]} \bar{\rho}(\phi, \omega, t'') \right), \tag{6}$$

.

This involves the computation of robustness backward in time. Note that for the online robustness computation, it is required that $length(\omega) \geq horizon(\phi)$. The horizon h for a specification ϕ is defined as follows:

$$
\begin{aligned}
h(\mathsf{true}, \omega, t) &= 0, && \text{(7a)}\\
h(\kappa, \omega, t) &= 0, && \text{(7b)}\\
h(\neg\phi, \omega, t) &= h(\phi, \omega, t), && \text{(7c)}\\
h(\phi \wedge \psi, \omega, t) &= \mathtt{max}(h(\phi, \omega, t), h(\psi, \omega, t)), && \text{(7d)}\\
h(\phi \vee \psi, \omega, t) &= \mathtt{max}(h(\phi, \omega, t), h(\psi, \omega, t)), && \text{(7e)}\\
h(\Diamond_I \phi, \omega, t) &= I + h(\phi, \omega, t), && \text{(7f)}\\
h(\Box_I \phi, \omega, t) &= I + h(\phi, \omega, t), && \text{(7g)}\\
h(\phi\, U_I\, \psi, \omega, t) &= I + \mathtt{max}(h(\phi, \omega, t), h(\psi, \omega, t)). && \text{(7h)}
\end{aligned}
$$

In this paper, the symbol $\bar{\rho}$ is used to denote robustness function only for online setting whereas ρ to define the general robustness function, i.e., applicable to both online and offline setting.

2.4 Other Semantics of Signal Temporal Logic

In this section, we mention various semantics of STL. The aim of these semantics has been to incorporate aggregate notions of robustness instead of point-based estimates and to provide a smooth approximation to the robustness function. Aggregation is needed to overcome the shadowing problem associated with the classical semantics of STL [Donzé and Maler(2010)].

The full semantics consists of expression for Negation, AND, OR, Always, and Eventually operator. Since negation is always defined as $\mathcal{N}(\rho) = -\rho$, the semantics of AND operator is sufficient to generate the full semantics, i.e., OR, Always and Eventually [Várnai and Dimarogonas(2020)]. We define the semantics of OR from that of AND using De Morgan's Law. Note that the Always operator is simply a conjunction (AND) over a given horizon. Similarly, the Eventually operator is disjunction (OR) over a given horizon.

We now provide the semantics of the AND operator for various state-of-the-art aggregation-based STL semantics, denoted by $\mathcal{A}(\rho_1, \ldots, \rho_n)$, where $\rho_1, \ldots, \rho_n$ are the robustness values passed as the inputs to the AND operator.

2.4.1 AGM Robustness [Mehdipour et al.(2019)]. The AGM robustness semantics replaces the max and min functions in the classical STL robustness definition with arithmetic and geometric means, yielding a score $\rho \in [-1, 1]$ that captures the aggregate satisfaction across all subformulas. The key intuition is that the geometric mean is used when a specification is *satisfied* (all relevant scores are positive), while the arithmetic mean is used when a specification is *violated* (at least one score is non-positive), averaging the degree of violation.

Formally, let $\rho_1, \ldots, \rho_n \in [-1, 1]$ be the normalized AGM robustness scores of n subformulae or time-point evaluations. The combined score $\mathcal{A}(\rho_1, \ldots, \rho_n)$ is defined as:

$$
\mathcal{A}(\rho_1, \ldots, \rho_n) = \begin{cases} \left(\prod_{i=1}^{n} (1 + \rho_i)\right)^{1/n} - 1, & \text{if } \forall\, i \in [1, \ldots, n],\ \rho_i > 0, \\ \dfrac{1}{n} \sum_{i=1}^{n} [\rho_i]^-, & \text{if } \exists\, i \in [1, \ldots, n],\ \rho_i \leq 0, \end{cases} \tag{8}
$$

where $[\rho_i]^- = \min(\rho_i, 0)$ extracts the non-positive part of ρ_i. The per-predicate normalized score is computed as

$$
\rho_i = \tfrac{1}{2}(s_i - p), \tag{9}
$$

where $s_i \in [-1, 1]$ is the normalized signal value and $p \in [-1, 1]$ is the predicate threshold, mapping the raw satisfaction margin into $[-1, 1]$.

2.4.2 Log-sum-exp [Li et al.(2018), Pant et al.(2017)]. This semantics (called LSE) uses logarithmic, summation, and exponential functions to construct a smooth approximation of the max/min function.

$$\mathcal{A}(\rho_1, \ldots, \rho_n) = -\frac{1}{\beta} \log \sum_{i=1}^{n} exp(-\beta \cdot \rho_i).$$

Here β is the smoothness parameter. The function reduces to the traditional max function when $\beta \to \infty$.

2.4.3 Softmax [Várnai and Dimarogonas(2020)]. This semantics uses the exponential function to create an aggregate approximation of the max/min function.

$$\mathcal{A}(\rho_1, \ldots, \rho_n) = 0 \qquad \text{if } \rho_{min} = 0,$$

$$\mathcal{A}(\rho_1, \ldots, \rho_n) = \frac{\sum_{i=1}^{n} \rho_i \cdot e^{-\nu \cdot \hat{\rho}_i}}{\sum_{i=1}^{n} e^{-\nu \cdot \hat{\rho}_i}} \qquad \text{if } \rho_{min} > 0,$$

$$\mathcal{A}(\rho_1, \ldots, \rho_n) = \frac{\sum_{i=1}^{n} \rho_{min} \cdot e^{\hat{\rho}_i} \cdot e^{\nu \cdot \hat{\rho}_i}}{\sum_{i=1}^{n} e^{\nu \cdot \hat{\rho}_i}} \qquad \text{if } \rho_{min} < 0,$$

where $\hat{\rho}_i = (\rho_i - \rho_{min})/\rho_{min}$. Here, ν is a constant. The SoftMax function reduces to the traditional max function when $\nu \to \infty$.

3 Problem

This section presents the controller synthesis problem that we address in this paper. We also introduce a few criteria to evaluate the synthesized controllers.

3.1 Controller Synthesis Problem

The controller synthesis problem addressed in this paper is formally presented below.

Problem 1 (Controller Synthesis)**.** *Let $\mathcal{P}$ be an open-loop dynamical system with unknown dynamics. Let the STL specification for the system be given by ϕ. Assuming that $\mathcal{P}$ does not satisfy ϕ, synthesize a controller C^* so that the expectation of robustness of all the traces generated by the closed-loop system $\mathcal{M}$ with respect to the specification ϕ gets maximized. Mathematically,*

$$C^* = \arg\max_{C} \mathop{\mathbb{E}}_{\omega \sim C} [\rho(\phi, \omega)]. \tag{10}$$

3.2 Controller Specification

It is well established that the specification of a real system should be captured as a conjunction of safety (something bad never happens) and liveness (something good happens eventually) properties [Alpern and Schneider(1987), Kindler(1994)]. Since real systems do not run forever, the liveness requirement is specified with a time-bound and is called bounded liveness [Lamport(2000)].

In this paper, we consider the systems involving locomotion where the liveness specification requires the system to make steady progress towards the goal, and the safety specification requires that the system does not move to a bad state during its movement. For example, in the case of a walker [Schulman et al.(2016)], the liveness condition would mean that the walker always moves forward, and the safety condition would ensure that it never falls down during the movement. During the controller synthesis phase, we perform the synthesis with a restricted notion of liveness, i.e., we want to ensure that the system makes a certain amount of progress

within a given duration. Our goal is to learn the best possible controller in terms of liveness while ensuring the safety constraints.

3.3 Controller Evaluation

Let T denote the duration of an episode for observing the behavior of the closed-loop system. Let the number of outputs of the system be p and the number of control inputs be q. The state output and the control input at time step t are denoted by $x_t = (x_t^1, x_t^2, \ldots, x_t^p)$ and $u_t = (u_t^1, u_t^2, \ldots, u_t^q)$. In this paper, we consider continuous control based locomotion benchmarks that have "distance covered" as a state. To evaluate a controller and compare it with other controllers, we use the metrics defined below.

Definition 1 (Control Cost). *We define the control cost at a given time-step t as*

$$CC_t = \sqrt{\frac{1}{q} \cdot \sum_{i=1}^{q} (u_t^i)^2}.$$

Now, the overall control cost (CC) is given by

$$CC = \frac{1}{T} \cdot \sum_{t=1}^{T} CC_t^2. \tag{11}$$

Definition 2 (Distance Covered (DC)). *We define the distance covered (DC) for a full episode as*

$$DC = x_T^k, \tag{12}$$

where x^k is the output corresponding to the distance.

Definition 3 (Margin of Satisfaction (MoS)). *We define the margin of satisfaction (MoS) for a trace ω of length T as*

$$MoS = \frac{1}{T} \cdot \sum_{t=1}^{T} \rho(\phi, \omega, t). \tag{13}$$

Definition 4 (Safety Satisfaction (SAT)). *Let ϕ_s denote the safety component of the specification ϕ. We define the safety satisfaction (SAT) for a full episode of length T as*

$$SAT = \mathbb{I}[\min(\{\rho(\phi_s, \omega, t)\}_{t=1}^{T})], \tag{14}$$

where $\mathbb{I}$ is an indicator function which is 1 when the inner expression is positive and 0 otherwise.

Unlike these metrics, the `gym` environment provides a reward function for each environment which is applicable to that specific environment only. We use this metric also in our evaluation and refer to it as *Default Reward (DR)*.

Ideally, a controller with low CC, high DC, high MoS, high SAT, and high DR is preferable. Note that both MoS and SAT are evaluated using the classical semantics of STL (by convention). Since the classical semantics uses the min function for conjunction (AND), the margin of satisfaction (MoS) of the whole specification often depends on the margin of safety specification (a.k.a. safety margin). This is because even though the liveness predicate values increase, generally the safety predicate values are bounded within a range, and hence the minimum for the whole specification most of the time depends on safety specification.

4 Deep RL Based Controller Synthesis

In this section, we present the deep reinforcement learning-based controller synthesis mechanism using STL specifications. Our controller synthesis methodology uses STL specifications to guide the policy search. However, unlike the traditional guided policy search method [Levine and Koltun(2013)], we do not employ any trajectory

optimization method [Tassa et al.(2012)]. Rather, the STL specification is considered as the collection of trajectories representing the expected behaviour of the system. The other benefit of using an STL specification is that we can utilize the quantitative semantics to generate the reward online using the robustness function.

To solve Problem 1 using reinforcement learning, we represent the model-free environment $M = \langle X, U, \mathcal{T}, \bar{\rho} \rangle$, where X denotes the set of continuous states $x \in \mathbb{R}^p$ of the system, U denotes the set of continuous control actions $u \in \mathbb{R}^q$ and $\bar{\rho}(\phi, \omega) \in \mathbb{R}$ represents the robustness of trace ω w.r.t. specification ϕ. The function $\mathcal{T} : \mathbb{R}^{p \times q \times p} \rightarrow [0, 1]$ represents the unknown dynamics of the system, i.e., the probability density of transitioning to the next state given the current state and action. To find the optimal policy $\pi^* : S \rightarrow A$, we define a parameterized policy $\pi_\theta(u|x)$ as the probability of choosing an action u given state x corresponding to parameter θ, i.e.,

$$\pi_\theta(u|x) = \mathbb{P}[u|x; \theta]. \tag{15}$$

We define the cost function (reward) associated with the policy parameter θ as follows:

$$J(\theta) = \mathop{\mathbb{E}}_{\omega \sim \pi_\theta} [\bar{\rho}(\phi, \omega)], \tag{16}$$

where $\mathop{\mathbb{E}}_{\omega \sim \pi_\theta}$ denotes the expectation operation over all traces ω generated by using the policy π_θ.

Our goal is to learn a policy that maximizes the reward. Mathematically,

$$\theta^* = \arg\max_\theta \ J(\theta). \tag{17}$$

Hence, our optimal policy would be the one corresponding to θ^*, i.e., π_{θ^*}.

The direct way to find the optimal policy is via policy gradient (PG) based algorithms. However, the PG algorithms are not sample-efficient and also unstable for continuous control problems. Actor-critic algorithms [Konda and Tsitsiklis(2003)], on the other hand, are highly sample-efficient as they use Q-function approximation (critic) instead of sample reward return.

The gradient for the Actor (policy) can be approximated as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \ [\sum_{t=0}^{T-1} \nabla_\theta \ log \ \pi_\theta(u_t|x_t) \cdot Q(x_t, u_t)]. \tag{18}$$

The function $Q(x_t, u_t)$ is given as:

$$Q(x_t, u_t) = \mathbb{E}_{\pi_\theta}[\sum_{t'=t}^{T} \bar{\rho}(\phi, \omega_{[t'-h(\phi),t']})|x_t, u_t]. \tag{19}$$

Here, $\omega_{[t'-h(\phi),t']}$ denotes the partial trace over which online robustness is computed. The length of the trace is given by $h(\phi)$.

The policy gradient, given by Equation (18), is used to find the optimal policy. The critic, on the other hand, uses an action-value function $Q^w(x_t, u_t)$ to estimate the function $Q(x_t, u_t)$. The term $\bar{\rho}(\phi, \omega_{[t'-h(\phi),t']})$ in Equation (18) refers to the online STL robustness, i.e., the robustness at time t.

The Q value plays an important role in RL. It has a considerable influence on the types of policies that can be expressed and the region of exploration. In fact, smoother Q-values (and hence ρ) are extremely useful in RL [Nachum et al.(2018)] as it leads to smooth policies [Shen et al.(2020)]. Moreover, using the smooth robustness function as the reward has similar semantic guarantees as the classical robustness function and provides significant speedup in learning [Li et al.(2018)].

5 A New Aggregation-Based STL Semantics

In this section, we present a new aggregation-based STL semantics and prove its various desirable properties.

Table 1. Comparing properties of different Semantics (SD=Soundness; MB=MinMax-boundedness; SL=Shadow-Lifting; SM=Smoothness).

STL Semantics	*SD*	*MB*	*SL*	*SM*
Classical	✓	✓	✗	✗
AGM	✓	✗	✓	✗
LSE	✗	✓	✗	✓
SoftMax	✓	✓	✓	✗
SSS	✗	✓	✓	✓

5.1 Desirable Properties of an STL Semantics

Let us denote the temporal operator used in the semantics of STL using f. We consider a specification ϕ and assume that ϕ is composed using subspecifications $\phi_1, \phi_2, \ldots, \phi_n$.

Recursion. This property specifies that the robustness of a specification can be computed using the robustness of its subspecifications, i.e.,

$$\rho(\phi) = f(\rho(\phi_1), \ldots, \rho_(\phi_n)). \tag{20}$$

Boundedness. This property requires that the robustness of a any specification is finite and bounded within a range, i.e.,

$$lb \leq \rho(\phi) \leq ub, \tag{21}$$

where $lb > -\infty$ and $ub < \infty$.

Shadow-lifting. This property ensures that for a specification, the robustness of any subspecification does not overshadow the robustness computed by the other subspecifications.

$$\frac{\partial f(\rho(\phi_1), \ldots, \rho(\phi_i), \ldots, \rho(\phi_n))}{\partial \rho(\phi_i)} \mid_{\rho(\phi_1),\ldots,\rho(\phi_n)=\rho} > 0, \forall i = 1, \ldots, n. \tag{22}$$

Soundness. This property requires that robustness of a specification is positive/negative iff the specification is satisfied/violated.

$$\rho(\phi) \geq 0 \qquad \text{if } \mathcal{M} \models \phi,$$
$$\rho(\phi) < 0 \qquad \text{if } \mathcal{M} \not\models \phi,$$

where $\models$ denote the satisfaction of a specification ϕ by a model $\mathcal{M}$.

Smoothness. The property requires that the robustness function is continous and differentiable, i.e., $\frac{\partial f}{\partial \rho(\phi)}$ exists everywhere ($\forall x \in X$).

In Table 1, we provide a comparison of the different state-of-the-art STL semantics - `Classical`, `AGM`, `LSE`, and `Softmax` along with our semantics (`SSS`) with respect to the four properties - Soundness, Min-max boundedness, Shadow-lifting, and Smoothness. The classical semantics suffers from the shadowing problem and is not smooth due to the usage of max/min functions. The `AGM` and `SoftMax` semantics address the shadowing problem but are not smooth. The `LSE` semantics, although smooth, does not address the shadowing problem, leading to misleading gradient signals during policy optimization. This makes them unsuitable for stable and informative reward generation in RL settings. Although [Várnai and Dimarogonas(2020)] characterizes certain tradeoffs and proposes a variant that relaxes the smoothness condition, our contribution goes beyond selecting a different tradeoff point. We provide both theoretical analysis and extensive empirical evaluation demonstrating superior learning stability and policy performance compared to existing semantics. Our `SSS` semantics (introduced in the next subsection), though not sound, addresses the shadowing problem and is smooth as well.

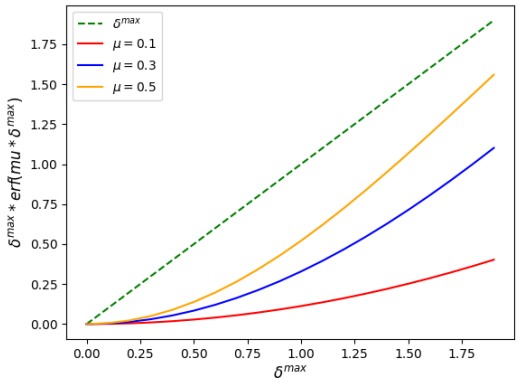


Fig. 1. Plot showing `erf` function for different values of μ

5.2 Smoothened-Summation-Subtraction

In this section, we present our new semantics **Smoothened-Summation-Subtraction (SSS)** that has been designed specifically to be used for reward generation in the RL-based controller synthesis algorithm. Since the semantics of AND operator is sufficient to generate the full semantics, we will define the semantics only for the AND operator.

The default definition of the minimum of two numbers is given by

$$\min(x_1, x_2) = \frac{(x_1 + x_2) - |x_1 - x_2|}{2}. \tag{23}$$

We extend this definition to approximate the robustness for AND as

$$\mathcal{A}(\rho_1, \ldots, \rho_n) = \frac{\sum_{i=1}^{n} \rho_i - (\rho_{max} - \rho_{min})}{n}. \tag{24}$$

where $\rho_{max} = \max(\rho_1, \ldots, \rho_n)$ and $\rho_{min} = \min(\rho_1, \ldots, \rho_n)$.

We formulate the approximation for AND in expression (24) because of two reasons - firstly, we want to use an aggregate measure of all the robustnesses in the AND expression, and secondly, we want to penalize the largest difference between any two robustnesses. Let us denote this largest difference by δ^{max}.

In Equation 24, we incorporate two modifications in order to make it suitable for reward generation. Firstly, the robustness computed by the AND operator is penalized more when δ^{max} is higher and lesser when δ^{max} is close to 0. This is needed as correct reward generation requires the ability to distinguish between different scenarios and reward/penalize them suitably. However, this is not the case with using the default δ^{max} value which can take the same value in both cases. We approximate the δ^{max} term using a non-linear function. We found the most suitable for approximation as $x.\mathtt{erf}(\mu.x)$ using the Gaussian error function `erf` where μ is a smoothness parameter. The `erf` function is defined as

$$\mathtt{erf}(x) = \frac{2}{\pi} \int_0^x e^{-t^2} dt. \tag{25}$$

Hence, our approximation to the robustness for the AND expression is

$$\mathcal{A}(\rho_1, \ldots, \rho_n) = \frac{\sum_{i=1}^{n} \rho_i - (\delta^{\max} \cdot \mathtt{erf}(\mu \cdot \delta^{\max}))}{n}, \tag{26}$$

where $\delta^{max} = \rho_{max} - \rho_{min}$.

Case 1 - $\delta^{max} > 1$: Consider a set of robustness values $\{1, 1, 3\}$. For $\mu = 0.3$, the robustness computed by Equation 26 is $\frac{1+1+3-\{2*erf(0.3*2)\}}{3} = 1.26$. Compare this with the robustness computed by Equation 24 (i.e., without `erf` approximation), given by $\frac{1+1+3-\{2\}}{3} = 1$ which is exactly equal to the min function (no aggregation). Note that the robustness computed in this case is not always equal to min function, but this is improbable in case of `erf` approximation for non-zero μ.
Case 2 - $\delta^{max} < 1$: Consider a set of robustness values $\{1, 1, 1.5\}$. For $\mu = 0.3$, the robustness computed by Equation 26 is $\frac{1+1+1.5-\{0.5*erf(0.3*0.5)\}}{3} = 1.14$. On the other hand, the robustness computed by Equation 24 (i.e., without `erf` approximation), given by $\frac{1+1+1.5-\{0.5\}}{3} = 1$.

The above example demonstrates the issue with the default δ^{max} in Equation 24. Although the δ^{max} values are different (0.5 and 2), the robustness computed is same, i.e., 1 in both cases. This is anomalous as we need to distinguish between the two behaviours - when robustness of all components of specification are increasing or when robustness of only some components are increasing. This is achieved with the erf function approximation. Moreover, although the δ^{max} values in the two cases is 2 and 0.5, the approximated values are 1.20 (60% penalization) and 0.08 (16% penalization) respectively, i.e., penalizing more for larger δ^{max}.

Secondly, the AND operator is non-smooth because of the max/min function. The non-smoothness w.r.t the max/min function in the δ^{max} expression is addressed as follows:

$$\delta^{max} \approx \frac{\log(n + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \exp(\eta \cdot (\rho_i - \rho_j)))}{\eta}. \tag{27}$$

As we know, the LSE [Li et al.(2018)] approximation of min and max function is given as

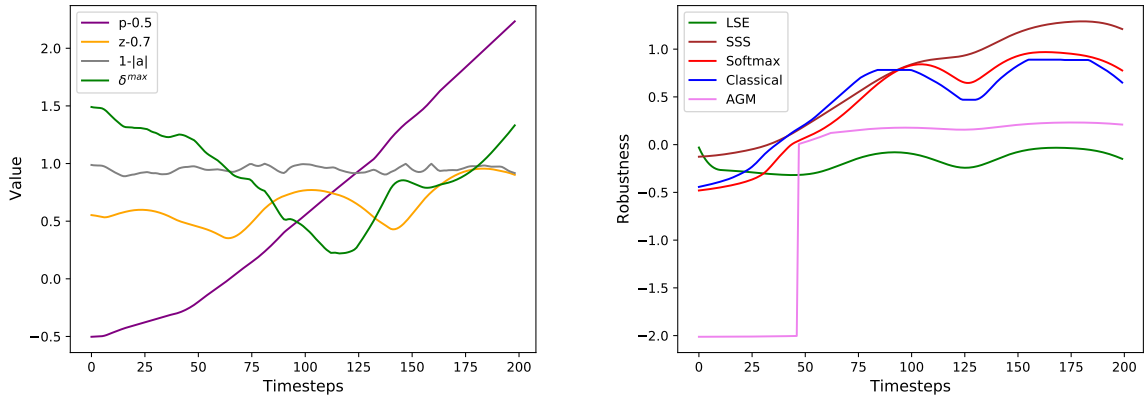
$$\rho_{min} \approx -\frac{1}{\eta} \log \sum_{i=1}^{n} exp(-\eta \cdot \rho_i),$$

$$\rho_{max} \approx \frac{1}{\eta} \log \sum_{i=1}^{n} exp(\eta \cdot \rho_i).$$

Hence, we can approximate $\delta^{max} = \rho_{max} - \rho_{min}$ as

$$\begin{aligned}
\hat{\delta}^{max} &\approx \frac{\log(\sum_{i=1}^{n} \exp(\eta \cdot \rho_i)) + \log(\sum_{i=1}^{n} \exp(-\eta \cdot \rho_i))}{\eta}, \\
&\approx \frac{\log(\sum_{i=1}^{n} \exp(\eta \cdot \rho_i) \cdot \sum_{i=1}^{n} \exp(-\eta \cdot \rho_i))}{\eta}, \\
&\approx \frac{\log((e^{\eta \cdot \rho_1} + e^{\eta \cdot \rho_2} + ...) \cdot (e^{-\eta \cdot \rho_1} + e^{-\eta \cdot \rho_2} + ..))}{\eta}, \\
&\approx \frac{\log(e^{\eta \cdot (\rho_1 - \rho_1)} + e^{\eta \cdot (\rho_2 - \rho_2)} + ... + e^{\eta \cdot (\rho 1 - \rho_2)} + ...)}{\eta}, \\
&\approx \frac{\log(1 + 1 + .. + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \exp(\eta \cdot (\rho_i - \rho_j)))}{\eta}, \\
&\approx \frac{\log(n + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \exp(\eta \cdot (\rho_i - \rho_j)))}{\eta}.
\end{aligned}$$

In the above expression, δ^{max} gets a smooth approximation of the largest difference between any two robustness values $\rho_i - \rho_j$. In the above expressions, μ and η are tunable parameters. Note that as $\eta \to \infty, \mu \to \infty$ reduces the AND operator in (26) to the traditional min function. We define our semantics with a high η and low μ value. A



(a) Values of different components for Hopper Example

(b) Robustness curves for each semantics

Fig. 2. Intuition behind SSS semantics

high value of η enables us to be close to the true value of δ^{max} and a low μ gives us smoothened overall robustness value.

The corresponding expression representing OR (O), Eventually ($\Diamond$) and Always ($square$) operator is given by:

$$O(\rho_1, \ldots, \rho_n) = \frac{\sum_{i=1}^{n} \rho_i + (\delta^{\max} \cdot \texttt{erf}(\mu \cdot \delta^{\max}))}{n}, \tag{28}$$

,

$$\Diamond_{[1,m]}(\rho) = \frac{\sum_{i=1}^{m} \rho_i + (\delta^{\max} \cdot \texttt{erf}(\mu \cdot \delta^{\max}))}{m}, \tag{29}$$

,

$$\Box_{[1,m]}(\rho) = \frac{\sum_{i=1}^{m} \rho_i - (\delta^{\max} \cdot \texttt{erf}(\mu \cdot \delta^{\max}))}{m}, \tag{30}$$

where $\delta^{max} = \rho_{max} - \rho_{min}$. The term ρ_i in expressions for Eventually and Always operator represents the value of ρ at each time instant $i \in [1, m]$.

5.3 Design Intuition for SSS

In this section, we provide a detailed analysis of the reward generation using SSS semantics. We first compare SSS with the other semantics in terms of the properties they satisfy. Subsequently, we discuss the design principles for the SSS in detail.

Consider the specification for Hopper:

$$\phi = \Diamond_{[0,15]}(p[t] > 0.5) \wedge \Box_{[0,20]}((z[t] > 0.7) \wedge (|a[t]| < 1)). \tag{31}$$

Note that in the above specification, there are two types of aggregation - one by temporal operators (Always/Eventually) and the other by AND/OR operator. The temporal aggregation works similar to aggregation done by the AND/OR operator, but in time domain.

We have generated a synthetic data for Hopper using the default controller (stable-baselines [Raffin et al.(2021)]). This dataset consists of simulation data for 200 time-steps. The values for predicates $(p - 0.5)$, $(z - 0.7)$ and $(1 - |a|)$ as well as δ^{max} are shown in Figure 2a. Consider the case of initial time-step. The values of these three predicates are -0.44, 0.52 and 0.90 respectively. The plots for all the semantics for the specification in (31) is

shown in Figure 2b. The robustness values at this point for the different semantics are `Classical` = −0.44, `AGM` = −2.01, `LSE` = −0.03, `Softmax` = −0.48 and `SSS` = −0.12 (refer Figure 2b).

We now analyze the soundness of different curves using plots in Figure 2b. The soundness property is only satisfied by the `Classical`, `AGM` and `Softmax`. When the minimum robustness among the three predicates (Figure 2a) is negative, traditional soundness requires the semantics to generate negative robustness. For instance, in Figure 2a we notice the purple curve (which is the minimum of the purple, yellow, and grey curves) crosses the 0 value near time-step 40. This effect is reflected in Fig 2b as the robustnesses of different semantics become positive at around time-step 40. This is satisfied by all semantics except `LSE`. The `SSS` semantics sometimes does not satisfy the soundness property, but in this case, it does. This can be explained via cases 1, 2, and 4 of Theorem 2 which is the same as traditional soundness and is satisfied by the SSS semantics.

From the figures, it is hard to show the effect of shadow-lifting property due to multiple aggregation sources (conjunction, eventually, always). The min-max boundedness is not satisfied by the `AGM` since the robustness is even less than the minimum of all the predicates. We observe that only the `LSE` and `SSS` robustness curves are smooth.

We now discuss the main intuitions behind designing the SSS semantics. To obtain the robustness (reward) from a given set of robustness values, the strategies can be divided into two broad categories - independent and dependent. Summation (like mean) is an independent strategy where overall robustness will increase if either of the robustness increase. Min (minimum) is a dependent strategy where overall robustness will increase only if the minimum robustness increases. Summation is problematic because it might happen that overall robustness (reward) might increase even though some components are performing worse (say, the liveness margin outweighs the safety margin). Min is problematic because of the shadowing problem. We try to incorporate the best of both worlds into SSS semantics.

We now try to understand how aggregation happens via `SSS` semantics. Ideally, the aggregate robustness corresponding to AND of the predicate values at time-step $t = 0$ −0.44, +0.52, +0.90 (refer Figure 2a) should not be close to −0.44 but be pushed towards a higher value due to the effect of robustness values +0.52 and +0.90. This is captured by `SSS` semantics. The strategy that sets `SSS` apart from other semantics is that it performs two operations simultaneously - it does more aggregation and penalizes the largest difference between the robustness (δ^{max}) values for the AND operator. This penalization will only increase when the minimum value goes far from other robustness values. In Figure 2a, we observe that the δ^{max} is high in the initial part, low in the middle, and becomes high again towards the end. Moreover, the aggregation (intuitively like mean) increases continuously. Consequently, the curve for `SSS` semantics while increasing, is flatter initially, steeper in the middle and tends to get flatter towards the end. We want to capture the AND of the three predicate curves which generally means the min of the three curves (classical semantics). In `SSS`, by penalizing δ^{max}, what we essentially get is more reward (robustness) when min of the predicates is closer to the *mean* and less reward when they are farther. This way, we are neither totally dependent on the min (and hence get aggregation) nor deviate too much from min (since AND operator should be analogous to min).

5.4 Properties Satisfied by SSS Semantics

We now present various useful properties satisfied by the semantics of AND in Equation (26).

Theorem 1 (Properties of SSS semantics). *The STL semantics generated by operator $\mathcal{A}$ in Equation (26) is*
1) recursive, i.e., the qualitative/quantitative satisfaction of a specification can be derived from the qualitative/quantitative satisfaction of its sub-specifications,
2) satisfies min-max boundedness, i.e., the following condition holds:

$$\rho_{min} \leq \mathcal{A}(\rho_1, \ldots, \rho_n) \leq \rho_{max}$$

3) continuous and differentiable, and
4) satisfies the shadow-lifting property, i.e., for any $\rho \neq 0$,

$$\frac{\partial \mathcal{A}(\rho_1, \ldots, \rho_i, \ldots, \rho_n)}{\partial \rho_i} \mid_{\rho_1, \ldots, \rho_n = \rho} > 0, \forall i = 1, \ldots, n.$$

Proof. 1) As we know, the corresponding max function for the min function defined in Equation (23) is given by

$$\max(x_1 + x_2) = \frac{(x_1 + x_2) + |x_1 - x_2|}{2}. \tag{32}$$

Consequently, the SSS semantics for OR operator would be defined as:

$$\mathcal{O}(\rho_1, \ldots, \rho_n) = \frac{\sum_{i=1}^{n} \rho_i + \{\delta^{max} \cdot \texttt{erf}(\mu \cdot \delta^{max})\}}{n}. \tag{33}$$

De Morgan's law expresses the relation between AND and OR operator as follows

$$\mathcal{O}(\rho_1, \ldots, \rho_n) = \mathcal{N}(\mathcal{A}(\mathcal{N}(\rho_1), \ldots, \mathcal{N}(\rho_n))), \tag{34}$$

where $\mathcal{N}$ refers to negation. In SSS semantics, negation corresponds to taking the negative of the robustness value, i.e., $\mathcal{N}(\rho)$ becomes $-\rho$. We use negation notation explicitly in the proof to reflect the application of the Negation operator with the AND operator.

Therefore,

$$\begin{aligned}
\mathcal{N}(\mathcal{A}(\mathcal{N}(\rho_1), \ldots, \mathcal{N}(\rho_n))) &= \mathcal{N}(\mathcal{A}(-\rho_1, \ldots, -\rho_n)), \\
&= -\{\frac{-\sum_{i=1}^{n} \rho_i - \{\delta^{max} \cdot \texttt{erf}(\mu \cdot \delta^{max})\}}{n}\}, \\
&= \frac{\sum_{i=1}^{n} \rho_i + \{\delta^{max} \cdot \texttt{erf}(\mu \cdot \delta^{max})\}}{n}, \\
&= \mathcal{O}(\rho_1, \ldots, \rho_n).
\end{aligned}$$

From the expressions for Eventually and Always operator (Equation 29-30), it is trivial to show the recursive structure for the Eventually and Always operators follows directly.

2) Since $\texttt{erf}(x) \in [0, 1]$ for $x \in [0, 1]$, we can approximate the expression $x \cdot \texttt{erf}(\mu \cdot x)$ with minimum value 0 and maximum value x. Let us consider the first part of the inequality

$$\begin{aligned}
\rho_{min} &\leq \mathcal{A}(\rho_1, \ldots, \rho_n), \\
&\leq \frac{\sum_{i=1}^{n} \rho_i - \{\delta^{max} \cdot \texttt{erf}(\mu \cdot \delta^{max})\}}{n}.
\end{aligned}$$

The minimum value of $\mathcal{A}(\rho_1, \ldots, \rho_n)$ is achieved when expression $\texttt{erf}(\mu \cdot \delta^{max})$ is at its maximum value, i.e., 1.

$$\rho_{min} \leq \frac{\sum_{i=1}^{n} \rho_i - (\rho_{max} - \rho_{min})}{n},$$

$$(n-1) \cdot \rho_{min} \leq \sum_{i=1}^{n} \rho_i - \rho_{max}.$$

The RHS of the above inequality represents the sum of $n-1$ robustness values, excluding the largest robustness. This is definitely greater than or equal to $n-1$ times the smallest robustness value. This proves the first part of inequality.

For the second part of the inequality

$$\mathcal{A}(\rho_1, \ldots, \rho_n) \leq \rho_{max}.$$

The maximum value of $\mathcal{A}(\rho_1, \ldots, \rho_n)$ is achieved when expression $\texttt{erf}(\mu \cdot \delta^{max})$ is at its minimum value, i.e., 0.

$$\frac{\sum_{i=1}^{n} \rho_i}{n} \leq \rho_{max},$$

$$\sum_{i=1}^{n} \rho_i \leq n \cdot \rho_{max}.$$

The RHS above is n times the largest robustness value which is always greater than the LHS.

3) The summation function Σ and the error function $\texttt{erf}()$ are both well known continuous and differentiable functions. We can easily verify that the expression $x \cdot \texttt{erf}(\mu \cdot x)$ is also differentiable. As we know, the difference between two differentiable expressions is also differentiable. Hence, the operator $\mathcal{A}$ is continuous and differentiable.

4) Consider the partial derivative of $\mathcal{A}(\rho_1, \ldots, \rho_n)$ w.r.t ρ_i

$$\begin{aligned}
\frac{\partial \mathcal{A}}{\partial \rho_i} &= \lim_{\epsilon \to 0} \frac{\mathcal{A}(\rho + \epsilon, \rho, \ldots, \rho) - \mathcal{A}(\rho, \ldots, \rho)}{\epsilon}, \\
&= \lim_{\epsilon \to 0} \frac{\frac{(\rho \cdot n + \epsilon - (\rho + \epsilon - \rho) \cdot \texttt{erf}(\mu \cdot (\rho + \epsilon - \rho)))}{n} - \rho}{\epsilon}, \\
&= \lim_{\epsilon \to 0} \frac{\frac{\epsilon}{n}\{1 - \texttt{erf}(\mu \cdot \epsilon)\}}{\epsilon}, \\
&= \lim_{\epsilon \to 0} \frac{\{1 - \texttt{erf}(\mu \cdot \epsilon)\}}{n}, \\
&= \frac{1}{n}.
\end{aligned}$$

Since n is positive, $\frac{\partial \mathcal{A}}{\partial \rho_i} > 0$. □

The following corollary strengthens the bound on $\mathcal{A}$ operator in Theorem 1 by showing that, whenever the robustness values are not all identical, the operator lies strictly between the minimum and the arithmetic mean of the robustness values.

Corollary 1. *Assuming that all robustness values $\rho_1, \ldots, \rho_n$ are not equal, the operator $\mathcal{A}$ in Equation (26) satisfies the following (strict) inequality*

$$\rho_{min} < \mathcal{A}(\rho_1, \ldots, \rho_n) < \rho_{mean},$$

where $\rho_{mean} = \text{mean}(\rho_1, \ldots, \rho_n)$.

If all ρ_i's are equal, then the following equality holds:

$$\rho_{min} = \mathcal{A}(\rho_1, \ldots, \rho_n) = \rho_{mean}.$$

Proof. From part 2 of the proof of Theorem 1, we have

$$(n-1)\cdot\rho_{min} \leq \sum_{i=1}^{n}\rho_i - \rho_{max}.$$

Since the robustness values are not all equal, the right-hand side corresponds to the sum of $(n-1)$ robustness values excluding the largest one, and is therefore strictly greater than $(n-1)\cdot\rho_{min}$. Hence,

$$(n-1)\cdot\rho_{min} < \sum_{i=1}^{n}\rho_i - \rho_{max}.$$

Dividing both sides by n gives

$$\rho_{min} < \frac{\sum_{i=1}^{n}\rho_i - (\rho_{max}-\rho_{min})}{n}.$$

Now, since $\delta^{max} > 0$ and $0 < \texttt{erf}(\mu\cdot\delta^{max}) < 1$, it follows that

$$0 < \delta^{max}\cdot\texttt{erf}(\mu\cdot\delta^{max}) < \delta^{max}.$$

Using the definition of $\mathcal{A}$,

$$\mathcal{A}(\rho_1,\ldots,\rho_n) = \frac{\sum_{i=1}^{n}\rho_i - (\delta^{max}\cdot\texttt{erf}(\mu\cdot\delta^{max}))}{n},$$

we obtain

$$\mathcal{A}(\rho_1,\ldots,\rho_n) > \frac{\sum_{i=1}^{n}\rho_i - \delta^{max}}{n} = \frac{\sum_{i=1}^{n}\rho_i - (\rho_{max}-\rho_{min})}{n}.$$

Combining with the earlier inequality yields

$$\rho_{min} < \mathcal{A}(\rho_1,\ldots,\rho_n).$$

Next, since $\delta^{max}\cdot\texttt{erf}(\mu\cdot\delta^{max}) > 0$ whenever the robustness values are not all equal, we have

$$\mathcal{A}(\rho_1,\ldots,\rho_n) < \frac{\sum_{i=1}^{n}\rho_i}{n} = \rho_{mean}.$$

Therefore,

$$\rho_{min} < \mathcal{A}(\rho_1,\ldots,\rho_n) < \rho_{mean}.$$

Finally, if all ρ_i are equal, then $\delta^{max} = 0$, which implies

$$\mathcal{A}(\rho_1,\ldots,\rho_n) = \frac{\sum_{i=1}^{n}\rho_i}{n} = \rho_{min} = \rho_{mean}.$$

This completes the proof. □

It has been proved in [Várnai and Dimarogonas(2020)] that the operator AND cannot be sound, idempotent, and smooth simultaneously. Consequently, the authors in [Várnai and Dimarogonas(2020)] came up with a metric that was sound and idempotent but not smooth. We instead propose a metric that is idempotent and smooth but not sound. This is because soundness as a property is not relevant when we consider the robustness metric as the reward for enforcing desirable behavior. For, instance, if the specification is too optimistic, it might not be satisfiable. Yet, by using this specification, we may converge to a desirable behavior. We can achieve the same behavior with (say) two different specifications - one having stricter constraints and the other with relaxed constraints. With learning, we will eventually get the desirable behavior in both cases - former with negative robustness (not sound) and latter with positive robustness (sound). Instead, smoothness plays a crucial role in gradients-based controller synthesis methods [Li et al.(2018)]. Nevertheless, our semantics satisfies the aggregate

notion of soundness (Theorem 2), and we provide the bounds on the robustness values generated by the operator in all cases.

Smoothness of the semantics is important because it directly determines the regularity of the induced reward signal used during reinforcement learning. Unlike supervised learning, where gradients are taken with respect to a fixed loss defined on labeled data, reinforcement learning relies on reward gradients (either explicitly in policy gradient methods or implicitly via value-function approximation) to guide exploration and policy improvement. If the semantics are non-smooth, small changes in the trajectory can lead to abrupt changes in the reward signal, producing high-variance or unstable gradient estimates. This in turn can slow convergence or lead to poor local optima. In contrast, a smooth semantics ensures that small perturbations in system trajectories lead to correspondingly small changes in robustness values. This provides a well-behaved optimization landscape, enabling more stable gradient-based updates and more effective exploration.

Depending on the robustness values being positive or negative, we can split it into three cases - all positive, all negative, and lastly, some positive and some negative. In the theorem below, we prove that if all individual robustnesses are positive (negative), then the SSS semantics will certainly give us a positive (negative) robustness (satisfies soundness). Now, consider the third case wherein some robustnesses are positive and some are negative. Obviously, in this case, the largest robustness value is $\rho_{max} > 0$, and the smallest robustness value is $\rho_{min} < 0$. Depending upon the magnitude of the positive and negative robustness values in the set $\rho_1, \ldots, \rho_n$, the sum of the robustness values can be either positive or negative. We provide a bound on the robustness for both these cases as well (here, traditional soundness requires negative robustness for $\mathcal{A}$ in both cases). The traditional soundness is based on the *min* function, i.e., for a given set of values, if the minimum is negative, we require robustness of $\mathcal{A}$ to be negative. On the other hand, here we define the notion of aggregate soundness, which is based on all values in the set instead of point-based estimates (i.e., via min function). In the below theorem, part 1, 2, and 4 are also applicable in the case of traditional soundness as well while part 3 is unique to aggregate soundness. In Part 3, traditional soundness would have required robustness of $\mathcal{A}$ to always remain negative since ρ_{min} is negative. However, aggregate soundness takes into account the fact that the overall sum is positive, and hence robustness of $\mathcal{A}$ can be positive if the magnitude of positive robustness values outweigh the negative ones.

Theorem 2 (Aggregate Soundness). *Consider the set of robustness values $\{\rho_1, \ldots, \rho_n\}$. The operator $\mathcal{A}$ in Equation (26) is sound in aggregate sense i.e., it satisfies the following conditions:*

$$
\begin{aligned}
&1)\ \mathcal{A}(\rho_1, \ldots, \rho_n) > 0, && \text{if } \forall i \in [1, n]\ \rho_i > 0 \\
&2)\ \mathcal{A}(\rho_1, \ldots, \rho_n) < 0, && \text{if } \forall i \in [1, n]\ \rho_i < 0 \\
&3)\ \mathcal{A}(\rho_1, \ldots, \rho_n) > -\frac{1}{n}[|\rho_{max}| + |\rho_{min}|], && \text{if } \sum_{i=1}^{n} \rho_i > 0,\ \rho_{max} > 0,\ \rho_{min} < 0 \\
&4)\ \mathcal{A}(\rho_1, \ldots, \rho_n) < 0, && \text{if } \sum_{i=1}^{n} \rho_i < 0,\ \rho_{max} > 0,\ \rho_{min} < 0.
\end{aligned}
$$

Proof. *For Part 1 and 2:* Note that input to the `erf` function in Equation (26) is always positive (since $\delta^{max} > 0$). This results in the `erf` value always in $[0, 1]$. We consider the two extremes of Equation (26) with `erf` $= 0$ and `erf` $= 1$. The value of $\mathcal{A}$ lies within the range

$$
\mathcal{A}(\rho_1, \rho_2, \ldots, \rho_n) \in [\frac{\sum_{i=1}^{n} \rho_i - (\delta^{max})}{n}, \frac{\sum_{i=1}^{n} \rho_i}{n}]. \tag{35}
$$

The term $\frac{\sum_{i=1}^{n} \rho_i}{n}$ is the mean robustness and the term $\frac{\sum_{i=1}^{n} \rho_i - (\delta^{max})}{n}$ is $\frac{1}{n}$-th of the sum of $n-1$ largest robustness plus the minimum robustness. These two extreme values will always be positive (negative) if all the robustness

values are positive (negative). Hence, if all $\rho_i > 0$ or all $\rho_i < 0$, then the range of robustness of operator $\mathcal{A}$ in Equation (35) will always contain positive or negative robustness values, respectively.

For Part 3: Let us define two variables $a, b > 0$ such that $a = \rho_{max}$ and $b = -\rho_{min}$. Consider the case when $\sum_{i=1}^{n} \rho_i > 0$. This implies,

$$\rho_{max} + \sum_{i=1}^{n-2} \rho_i + \rho_{min} > 0,$$

$$a - b + \sum_{i=1}^{n-2} \rho_i > 0,$$

$$\sum_{i=1}^{n-2} \rho_i - 2 \cdot b > -b - a. \tag{36}$$

As we know, $Range(\texttt{erf}) = [0, 1]$ for a positive domain and the operator $\mathcal{A}$ has $\texttt{erf}$ with input $\mu \cdot \delta^{max}$ which is always positive (since $\mu > 0$ and $\rho_{max} - \rho_{min} > 0$). So, the minimum value of $\mathcal{A}$ is achieved for $\texttt{erf}(.) = 1$. Hence,

$$\begin{aligned}
\mathcal{A}(\rho_1, \ldots, \rho_n) &= \frac{\sum_{i=1}^{n} \rho_i - \{\delta^{max} \cdot \texttt{erf}(\mu \cdot \delta^{max})\}}{n}, \\
&\geq \frac{\sum_{i=1}^{n} \rho_i - \{\rho_{max} - \rho_{min}\}}{n}, \\
&\geq \frac{\sum_{i=1}^{n-2} \rho_i + a - b - \{a + b\}}{n}, \\
&\geq \frac{\sum_{i=1}^{n-2} \rho_i - 2 \cdot b}{n}, \\
&> \frac{-[b + a]}{n} \qquad //Using\ (36), \\
&> -\frac{[|\rho_{max}| + |\rho_{min}|]}{n}.
\end{aligned}$$

For Part 4: As seen in part 3, $Range(\texttt{erf}) = [0, 1]$ for a positive domain. So, maximum value for $\mathcal{A}$ is attained at $\texttt{erf}(.) = 0$. Therefore,

$$\begin{aligned}
\mathcal{A}(\rho_1, \ldots, \rho_n) &= \frac{\sum_{i=1}^{n} \rho_i - \{\delta^{max} \cdot \texttt{erf}(\mu \cdot \delta^{max})\}}{n}, \\
&\leq \frac{\sum_{i=1}^{n} \rho_i}{n}, \\
&< 0.
\end{aligned}$$

Since for this case $\sum_{i=1}^{n} \rho_i < 0$, so the above expression is always negative. □

Table 2. STL specifications

Model	STL Specification	Variables
HC	$\Diamond_{[0,10]}(v[t] > 0.1)$	v : velocity
Hop	$\Diamond_{[0,15]}(v[t] > 0.5) \wedge \Box_{[0,20]}((z[t] > 0.7) \wedge (\lvert a[t]\rvert < 1))$	v: velocity, z: height, a: angle
Ant	$\Diamond_{[0,5]}(vx[t] > 0.2) \wedge \Box_{[0,10]}((z[t] < 1) \wedge (z[t] > 0.2))$	vx: x-velocity, z: height
Wkr	$\Diamond_{[0,15]}(vx[t] > 0.5) \wedge \Box_{[0,20]}((z[t] > 0.8) \wedge (z[t] < 2) \wedge (\lvert a[t]\rvert < 1))$	vx: x-velocity, z : height, a: angle
Swm	$\Diamond_{[0,15]}((vx[t] > 0.1) \wedge \Box_{[0,20]}((\lvert a_1[t]\rvert < 1) \wedge (\lvert a_2[t]\rvert < 1) \wedge (\lvert a_3[t]\rvert < 1))$	vx: x-velocity, $\langle a_1, a_2, a_3\rangle$: angles of first tip, first rotor and second rotor
Hum	$\Diamond_{[0,5]}(vx[t] > 0.5) \wedge \Box_{[0,5]}((z[t] > 1) \wedge (z[t] < 2))$	vx: x-velocity, z: height

6 Experiments

This section presents our experiments to evaluate the Deep-RL based controller synthesis methodology for STL specifications using our proposed aggregation based semantics. All experiments are carried out on an Ubuntu20.04 machine with i7-4770 3.40 GHz×8 CPU and 32 GB RAM. The source code of our implementation and the videos of the simulation results are at https://github.com/iitkcpslab/rlstl.

6.1 Experimental Setup

To simulate the environment, we use the gym simulator [Brockman et al.(2016)]. The learning is performed using the state-of-the-art Actor-Critic algorithm SAC [Haarnoja et al.(2018)]. We modify the SAC algorithm to use the STL robustness computed online as the reward. We implement different STL semantics on top of the online monitoring tool RTAMT [Ničković and Yamaguchi(2020)]. We have also developed a Python program for controller evaluation in the gym environment.

We have performed our experiments on six continuous control benchmarks, namely HalfCheetah [Wawrzyński(2009)] (HC), Hopper [Erez et al.(2011)] (Hop), Ant [Schulman et al.(2016)], Walker [Erez et al.(2011)] (Wkr), Swimmer [Coulom(2002)] (Swm), and Humanoid [Erez et al.(2011)] (Hum). The number of observable states and actions for the benchmarks are shown in Table 4. In Table 2, we list the specifications used for training the controllers. All the units are in SI. Each specification captures the goal defined for the system by the gym environment. Each specification has a liveness component and a safety component (other than HC). For instance, for the Hopper example, the goal is to "Make a two-dimensional one-legged robot hop forward as fast as possible without falling". This is captured by the specification

$$\phi = \underbrace{\Diamond_{[0,15]}(v[t] > 0.5)}_{\text{liveliness}} \wedge \underbrace{\Box_{[0,20]}((z[t] > 0.7) \wedge (abs(a) < 1))}_{\text{safety}}.$$

The meaning of this specification should be understood in terms of the robustness that would be generated w.r.t. this specification, which gives us the reward for moving towards the goal. Intuitively, the liveness component of the specification implies that from any time step, the hopper should try to achieve velocity of at least 0.5 units within the next 15 time steps. For achieving velocity less than, equal to, and greater than 0.5 units, the reward (robustness) it will get will be negative, zero, and positive, respectively. The safety component ensures that the hopper never falls during the forward movement. All the safety requirements used in our experiments are exactly the same as that in the corresponding gym environment.

Table 3. Comparison of different semantics under different environments

Environment	Semantics	Control Cost	Distance Covered	*MoS*	*SAT*	*DefaultReward*
HalfCheetah	Classical	16127.17 ± 195.61	307.66 ± 4.17	7.30 ± 0.07	-	5755.00 ± 82.5
	AGM	16599.73 ± 181.25	406.29 ± 5.44	10.03 ± 0.10	-	7716.32 ± 108.02
	LSE	15311.38 ± 100.39	111.86 ± 1.26	4.25 ± 0.02	-	1850.36 ± 25.09
	Softmax	14154.17 ± 198.75	471.29 ± 7.10	10.76 ± 0.13	-	9047.92 ± 140.68
	SSS	14647.78 ± 984.72	541.90 ± 38.50	12.04 ± 0.78	-	10453.96 ± 779.36
Hopper	Classical	1698.20 ± 53.72	12.08 ± 0.25	0.68 ± 0.00	99/100	2505.08 ± 40.64
	AGM	928.52 ± 122.87	12.72 ± 1.76	0.58 ± 0.01	0/100	2196.76 ± 311.55
	LSE	42.81 ± 5.72	0.02 ± 0.02	−0.26 ± 0.03	0/100	31.91 ± 4.27
	SoftMax	1571.84 ± 46.85	13.99 ± 0.21	0.69 ± 0.00	100/100	2745.39 ± 27.05
	SSS	1231.04 ± 19.29	17.25 ± 0.07	0.69 ± 0.00	100/100	3153.21 ± 8.84
Ant	Classical	12055.73 ± 355.69	34.97 ± 1.11	0.32 ± 0.00	99/100	13.95 ± 37.32
	AGM	9089.84 ± 182.02	40.04 ± 0.78	0.31 ± 0.00	100/100	366.71 ± 14.32
	LSE	220.42 ± 74.78	0.32 ± 0.32	0.21 ± 0.06	100/100	−11.70 ± 7.54
	SoftMax	10938.52 ± 130.68	38.61 ± 0.88	0.32 ± 0.00	100/100	150.44 ± 14.73
	SSS	12678.64 ± 225.12	85.03 ± 1.57	0.30 ± 0.00	100/100	967.26 ± 36.99
Walker	Classical	5614.93 ± 456.56	8.56 ± 0.81	0.71 ± 0.09	99/100	2056.56 ± 188.54
	AGM	76.63 ± 1.25	−0.07 ± 0.00	−0.37 ± 0.02	0/100	−1.54 ± 0.07
	LSE	368.15 ± 59.34	−0.08 ± 0.01	−0.14 ± 0.07	0/100	44.25 ± 7.59
	SoftMax	10359.56 ± 700.12	−4.53 ± 0.87	0.33 ± 0.03	94/100	413.97 ± 114.76
	SSS	10743.50 ± 189.36	18.71 ± 0.11	0.59 ± 0.00	100/100	3333.06 ± 14.39
Swimmer	Classical	626.38 ± 38.65	−1.59 ± 0.21	0.37 ± 0.02	100/100	−40.05 ± 5.24
	AGM	87.48 ± 51.06	0.78 ± 0.36	0.00 ± 0.02	100/100	19.46 ± 9.06
	LSE	56.31 ± 76.83	0.25 ± 0.67	−0.06 ± 0.02	100/100	6.35 ± 16.89
	SoftMax	573.94 ± 78.30	0.05 ± 0.19	0.07 ± 0.10	3/100	0.98 ± 4.90
	SSS	1256.99 ± 16.22	7.18 ± 0.12	0.37 ± 0.00	99/100	179.09 ± 2.89
Humanoid	Classical	2072.98 ± 37.73	6.66 ± 0.39	0.28 ± 0.00	99/100	5396.01 ± 119.59
	AGM	3143.56 ± 35.54	18.43 ± 0.15	0.24 ± 0.00	100/100	6352.30 ± 12.85
	LSE	1531.31 ± 114.41	2.25 ± 1.03	0.09 ± 0.07	100/100	5060.10 ± 81.72
	SoftMax	2313.87 ± 37.79	7.94 ± 0.09	0.28 ± 0.00	100/100	5505.03 ± 7.43
	SSS	2901.91 ± 31.20	26.79 ± 0.55	0.27 ± 0.00	100/100	7054.92 ± 45.94

6.1.1 Benchmarks. In this subsection, we introduce the benchmark systems and their specifications that we use in our experiments. These benchmarks remain among the most widely used and cited continuous-control testbeds in reinforcement learning, and they continue to serve as standard evaluation platforms in recent literature. Importantly, they provide highly nonlinear, high-dimensional control tasks - for example, the Humanoid environment involves 376 state dimensions. The number of observations and the number of actions available for each benchmark are provided in Table 4.

Half-Cheetah [Wawrzyński(2009)]: It is a 2-D robot consisting of 9 links and 8 joints. Here the goal is to make the half-cheetah run forward as fast as possible by applying torque on the joints. The cheetah gets a

Table 4. Benchmarks

Environment	#Observations	#Actions
HalfCheetah	17	6
Hopper	11	3
Ant	27	8
Walker	17	6
Swimmer	8	2
Humanoid	376	17

positive reward based on the forward distance moved. As the torso and head of the cheetah are fixed, there is no safety requirement here (such as the cheetah falling down).

Hopper [Erez et al.(2011)]: It is a one-legged robot and has four body parts - the torso at the top, followed by the thigh, leg, and foot at the bottom. The goal is to make it hop forward by applying torque on the three hinges, which connect the four body parts. Here the safety requirement is that the hopper should not fall while hopping, i.e., maintaining minimum height z and without bending the foot too much (bounding the angle of foot a).

Ant [Schulman et al.(2016)]: This benchmark is among the most complex environments provided by gym. It consists of a 3-dimensional 4-legged robot, and the goal is to make it walk forward as fast as possible without falling, i.e., maintaining a minimum height of z. This is tricky because Ant has to be moved by coordinating the movement of four legs.

Walker [Erez et al.(2011)]: This is an advancement over the hopper. By having an extra leg, the robot can walk instead of just hopping. Here the goal is to make a 2D bipedal robot walk forward as fast as possible without falling, i.e., maintaining minimum height z, without jumping (bound on maximum height z), and without bending the foot too much (bounding the angle of foot a). What makes this benchmark complex is that for movement, it needs to coordinate both sets of feet, legs, and thighs to move forward.

Swimmer [Coulom(2002)]: It consists of three segments and two joints suspended in a 2-D pool. The goal is to move forward as fast as possible by applying torque on the joints and using the fluid's friction.

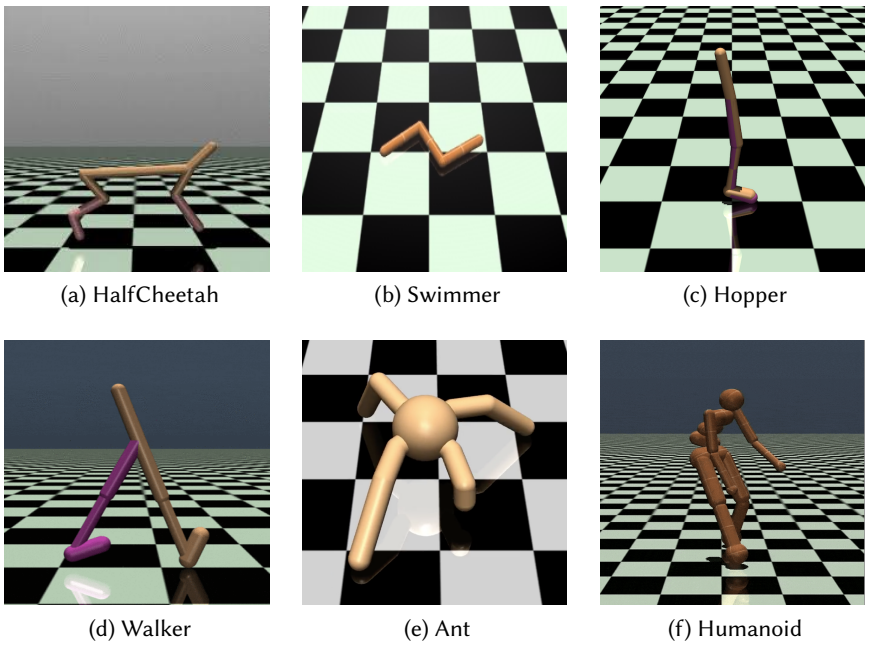
Humanoid [Tassa et al.(2012)]: This system is like a 3D bipedal robot simulating a human. It has the following parts: a torso with a pair of legs and arms. The goal is to walk forward as fast as possible without falling over.

Furthermore, our evaluation does not rely solely on these environments in isolation. We also compare our proposed semantics directly against state-of-the-art STL semantics (Section 2.4), thereby ensuring that the empirical assessment reflects both task complexity and methodological relevance.

6.1.2 Hyper-parameters. There are different hyper-parameters used in various semantics of STL. We have used the hyperparameter values ($\beta = 1$, $\nu = 3$) used in the work [Li et al.(2018), Várnai and Dimarogonas(2020)]. For our SSS semantics, we carry out initial experiments on the Hopper benchmark to understand the effect of the hyperparameters μ and η on the performance of the controller. Based on the insights obtained from experiments, we choose $\mu = 0.3$ and $\eta = 300$ for all our experiments. The benchmark configuration hyper-parameters, such as learning rate, total time-steps for training, etc., are taken from Stable-baselines3 [Raffin et al.(2021)].

6.2 Experimental Results

Table 3 presents the performance of the controllers synthesized using different semantics for various benchmarks. We test the synthesized controller on 100 distinct seed inputs for better coverage of the behavior of these



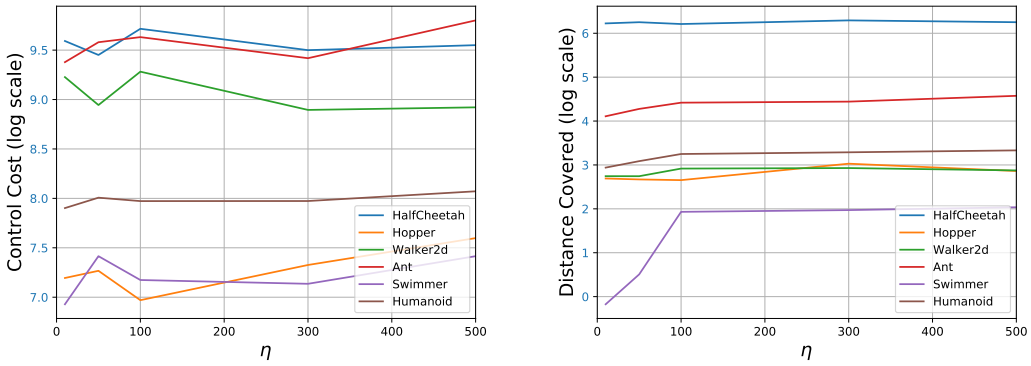
(a) HalfCheetah (b) Swimmer (c) Hopper

(d) Walker (e) Ant (f) Humanoid

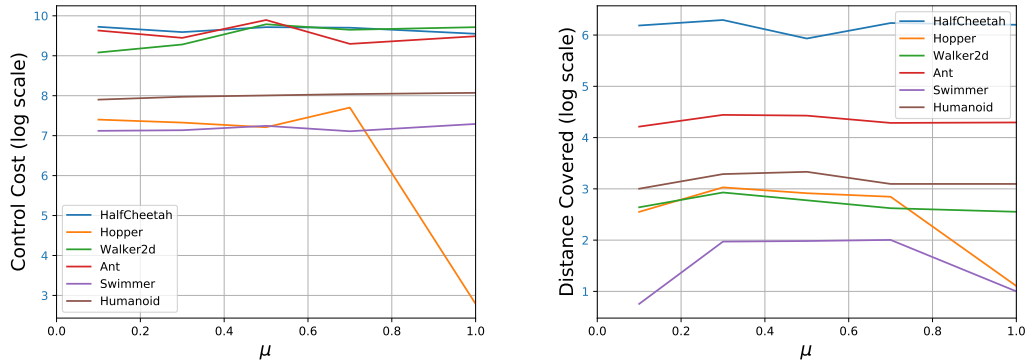
Fig. 3. Illustration of benchmarks (source: [Brockman et al.(2016)])

controllers. In Table 3, we show the mean and standard deviation (100 seeds) for all the metrics. The results show that our proposed STL semantics SSS outperforms all the other semantics on the metrics introduced in Section 3.3.

Some key observations about the experimental results shown in Table 3 are in order. (i) SSS is the only semantics that could synthesize a useful controller for all the benchmarks. In the case of Swimmer, a useful controller (enables the robot to move gracefully in the forward direction without violating the safety property) could be synthesized only via SSS robustness semantics. (ii) The controller synthesized by SSS achieves the maximum distance covered (DC) among all the controllers. It is also the best for acquiring gym's default reward (DR). (iii) In many cases, the controller synthesized using SSS robustness was not the one with minimum control cost (CC). This is reasonable because often low control cost (CC) means that the controller is not generating controls for the movement. For instance, in the case of Swimmer, for LSE and AGM, we have low control cost, but that is because the Swimmer is not moving at all. On the other hand, despite the high control cost for `classical` and `Softmax` (`SMax`), the Swimmer is not able to move. This shows that the controllers do not produce the correct control inputs. (iv) In almost all cases, the controller corresponding to SSS semantics has the highest MoS except for Ant and Humanoid. In both these cases, slightly less than the highest MoS is achieved. This is because it has to perform actions with a lower safety margin for faster movement. This is reasonable as ideally the safety requirement has to be satisfied by any satisfaction margin. (v) In almost all cases, we get the best controller in terms of SAT with 100 satisfaction (out of 100 samples) of the safety specification. In case of Swimmer, due to stricter safety requirements, a useful controller was synthesized only via SSS semantics. However, this was achieved at the cost of 99 safety satisfaction. It is worth noting that the safety constraints are easy to satisfy if the Swimmer does not move forward significantly. This is the reason why the `Classical`, AGM, and LSE semantics can achieve 100 satisfactions of the safety property. The `Softmax` semantics performs the worse, failing to satisfy



(a) For different values of η and fixed $\mu = 0.3$



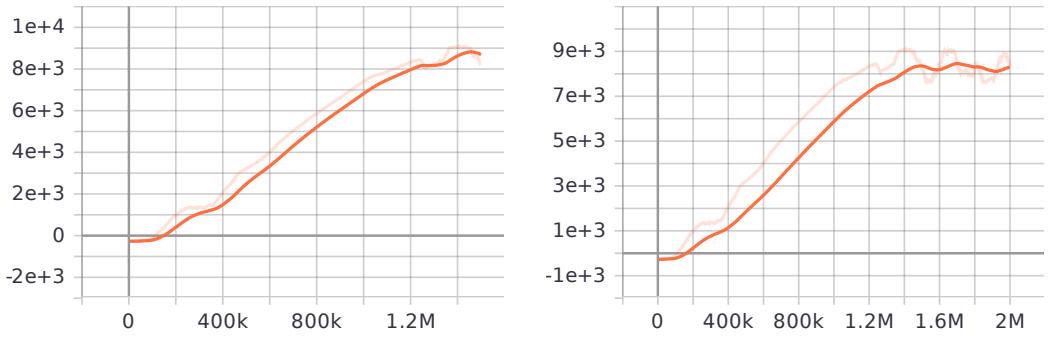
(b) For different values of μ and fixed $\eta = 300$

Fig. 4. Plots for Control Cost and Distance Covered for different environments

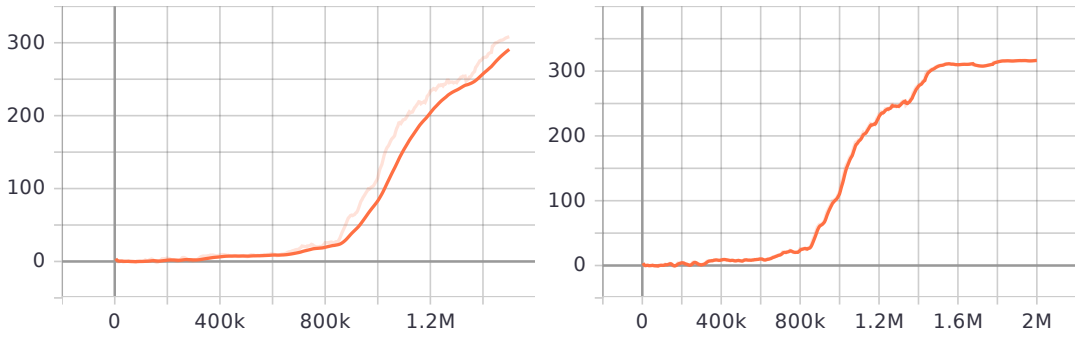
both the liveness and the safety requirements, owing to the U-shape motion of the Swimmer with the resulting controller, leading to poor DC and many safety violations.

6.2.1 Effect of Hyper-parameters of the SSS Semantics on Synthesis of Controllers. We analyze the effect of the hyperparameters μ and η on the performance of the synthesized controllers. We use the benchmarks and specifications listed in Table 2. Figures 4a and 4b illustrate the control cost and the distance covered (DC) for varying values of η and μ, respectively. Note that the y-axis is presented on a logarithmic scale. From Figure 4a, we observe that when $\eta = 300$, the distance covered across different environments is generally maximized while maintaining a moderate control cost. Similarly, Figure 4b shows that the distance covered is maximized at $\mu = 0.3$, and tends to decrease as μ increases. Overall, the results indicate that larger values of η combined with smaller values of μ yield improved controller performance.n particular, the best performance in terms of distance covered is achieved at $\mu = 0.3$ and $\eta = 300$. Accordingly, we adopt these values for all subsequent experiments.

6.2.2 Comparison of Different STL Semantics on Training. In Figure 6, we show the plots for the reward generation via different robustness semantics for each benchmark. It is important to note that in case of HalfCheetah and



(a) HalfCheetah



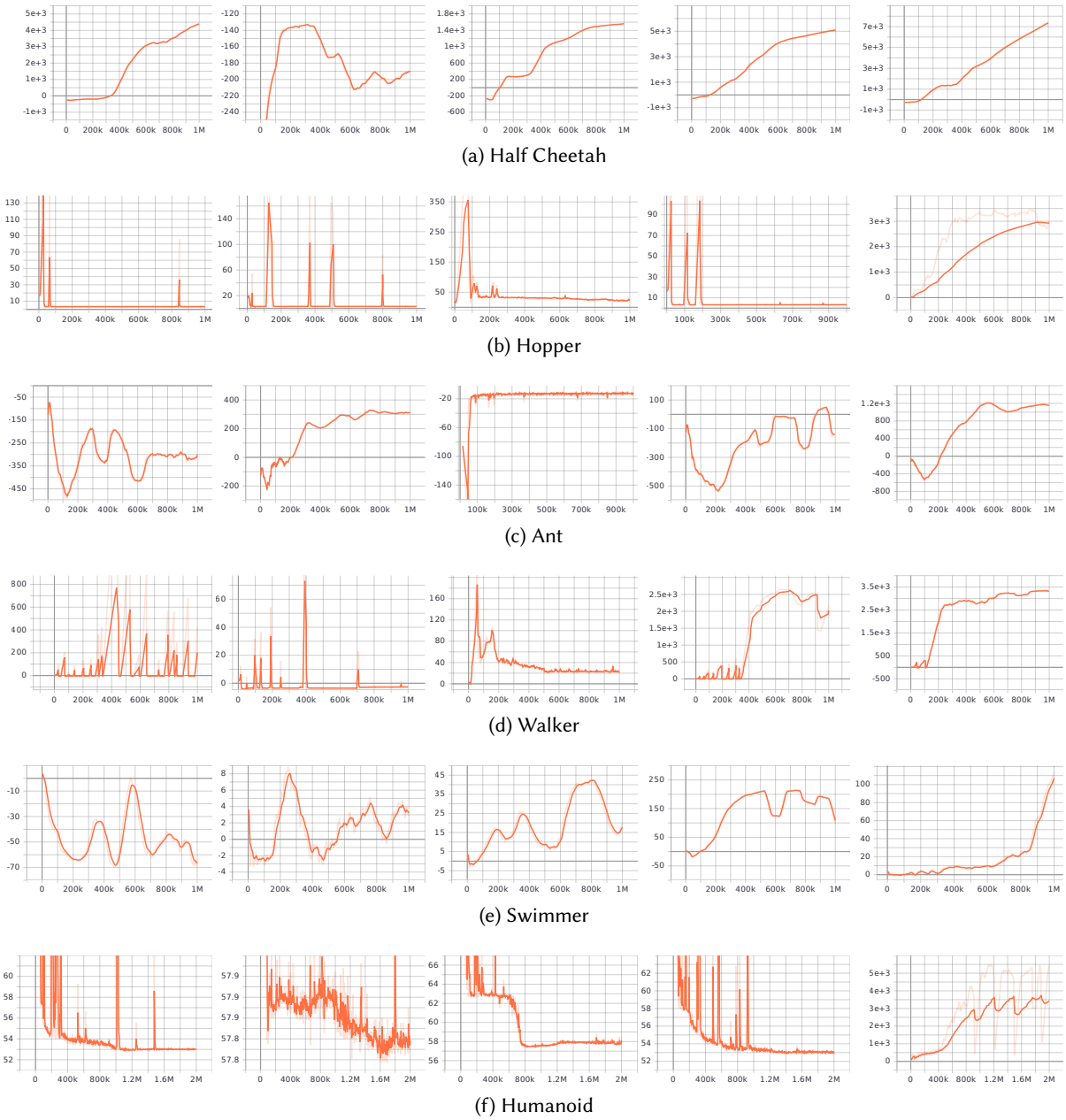
(b) Swimmer

Fig. 5. Plots showing convergence while training for 1.5M and 2M timesteps

Swimmer, it appears that convergence is not achieved for SSS semantics. For these cases, convergence would have been achieved had it been run for longer time (refer Figure 5a-5b). Nevertheless, we still get the best controller in those cases. The reason for running all the experiments for fixed timesteps is to provide a fair comparison of all the controller synthesis techniques (same as in stable-baselines [Raffin et al.(2021)]). In case of Hopper, Swimmer, and Humanoid benchmark, except for our SSS semantics, no other semantics could synthesize a useful controller (Figure 6a– 6f).

6.2.3 Effect of Training for Longer Duration on Controller Synthesis. From Figure 6a and 6e, we observe that convergence is not achieved for SSS semantics within 1M time steps, even though we get the best controller using the SSS semantics. Subsequently, we run the experiments for HalfCheetah and Swimmer for 1.5M and 2M time steps. In Table 5, we show the results for the synthesized controllers. Figures 5a and 5b show the convergence of the reward generated during the training for HalfCheetah and Swimmer, respectively. It is observed that the controllers synthesized by running for a longer duration were better than the standard one (1M time steps). However, we also observed that running for a longer duration comes with a bit of a caveat. In case of Half-Cheetah (2M), a high standard deviation is observed for all the metrics. This is due to the reason that the Cheetah runs at such a high speed that sometimes its forward leg tends to collide with the backward leg, degrading its performance. But due to this high speed only, it is able to achieve a record high reward (>13K).

6.2.4 Effect of Predicates of STL Specification on Controller Synthesis. Consider the predicate $p[t] > \epsilon$ in STL specification $\Diamond_{[0,10]}(p[t] > \epsilon) \wedge \Box_{[0,15]}((z[t] > 0.7) \wedge (|a[t]| < 1))$ for the Hopper benchmark. Table 6 shows



(a) Half Cheetah

(b) Hopper

(c) Ant

(d) Walker

(e) Swimmer

(f) Humanoid

Fig. 6. Plots showing the average rewards produced by different robustness semantics for each benchmark. Order: Classical, `AGM`, `LSE`, `SoftMax`, `SSS`

Table 5. Effect of training for longer duration (2M timesteps) for HalfCheetah and Swimmer w.r.t. SSS semantics

Environment	$\#Steps$	Control Cost	Distance Covered	MoS_{mean}	SAT	$DefaultReward$
HalfCheetah	1M	16529.15 ± 180.84	416.11 ± 8.10	200.42 ± 4.36	–	7912.99 ± 160.96
	1.5M	14515.39 ± 1894.06	432.18 ± 146.42	213.05 ± 57.57	–	8262.84 ± 2899.87
	2M	14449.75 ± 2053.90	464.33 ± 206.13	226.58 ± 94.67	–	8916.12 ± 4093.65
Swimmer	1M	1256.99 ± 16.22	7.18 ± 0.12	0.37 ± 0.00	99/100	179.09 ± 2.89
	1.5M	2075.94 ± 19.16	12.63 ± 0.08	0.39 ± 0.00	99/100	315.09 ± 1.60
	2M	1862.78 ± 19.15	12.69 ± 0.07	0.33 ± 0.00	99/100	317.15 ± 1.41

the results for different values of ϵ. We observe that a controller could not be synthesized for high values of ϵ. This is reasonable because, with a high value of ϵ, we demand that the hopper covers a larger distance within the given time. This forces the hopper to hop (jump) forward at high speed, which in turn leads to the hopper falling down (SAT = 0/100). Comparing the case of $\epsilon = 0.1$ and $\epsilon = 0.5$, we give more reward with the predicate $\epsilon = 0.1$ than $\epsilon = 0.5$ for the same distance covered. For instance, if the Hopper cover 1 unit of distance, the reward (i.e., the robustness) with predicate $p[t] > 0.1$ is 0.9 while that with $p[t] > 0.5$ is 0.5. Since for the same work, the former predicate leads to higher reward, the agent does not need to work as hard as with the latter to get more reward. This is reflected in the control cost as well. Hence, among the given predicates, we get the best controller for $\epsilon = 0.5$. Since for each different value of ϵ, we get a new specification, comparing the controllers synthesized in these cases on the metric MoS is not reasonable and hence omitted from the table. As the safety part of the specifications remains the same, it is included in the table.

Table 6. Effect of predicate $p > \epsilon$ of STL specification $\Diamond_{[0,10]}(p[t] > \epsilon) \wedge \Box_{[0,15]}((z[t] > 0.7) \wedge (|a[t]| < 1))$ for Hopper

Environment	Values	Control Cost	Distance Covered	SAT	$DefaultReward$
Hopper	$\epsilon = 0.1$	1140.78 ± 42.66	15.91 ± 0.14	100/100	2985.48 ± 17.87
	$\epsilon = 0.5$	1519.40 ± 31.73	20.67 ± 0.06	100/100	3579.27 ± 8.56
	$\epsilon = 1$	11.90 ± 0.15	−0.02 ± 0.00	0/100	3.07 ± 0.01
	$\epsilon = 2$	17.24 ± 0.08	−0.02 ± 0.00	0/100	3.07 ± 0.01

7 Conclusion

We propose a new aggregation-based smooth STL semantics for synthesizing feedback controllers via Reinforcement Learning. Our STL semantics is not sound, but has other desirable properties required for efficient RL based controller synthesis. We provide a comparative analysis of the proposed semantics with all the state-of-the-art STL semantics. Experimental results on several complex benchmarks establish that our proposed semantics outperforms all aggregate-based STL semantics proposed in the literature.

A direct extension of our work would be its application in other critical domains like self-driving cars, drones, medical sciences (e.g., Simglucose [Xie(2026)]). Going forward, we plan to work towards explainability and interpretability of the decision making by the control policies by using the reward decomposition in terms of liveness and safety. Another direction would be to combine the specifications for hierarchical RL to decompose complex tasks into multiple subtasks.

8 Acknowledgements

The first author acknowledges the Prime Minister's Research Fellowship from the Government of India and the second author acknowledges the research grant MTR/2019/000257 from Science and Engineering Research Board of Government of India for supporting this research. The authors also thank Dejan Nickovic for his support in working with the RTAMT Toolbox.

References

[Alpern and Schneider(1987)] Bowen Alpern and Fred B. Schneider. 1987. Recognizing Safety and Liveness. *Distributed Computing* 2, 3 (sep 1987), 117–126.

[Brockman et al.(2016)] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *CoRR* abs/1606.01540 (2016). arXiv:1606.01540 http://arxiv.org/abs/1606.01540

[Coulom(2002)] Rémi Coulom. 2002. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. Ph. D. Dissertation. Institut National Polytechnique de Grenoble.

[Deisenroth et al.(2013)] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. 2013. A Survey on Policy Search for Robotics. *Foundation and Trends in Robotics* 2 (2013), 1–142.

[Donzé et al.(2013)] A. Donzé, T. Ferrère, and O. Maler. 2013. Efficient Robust Monitoring for STL. 264–279.

[Donzé and Maler(2010)] Alexandre Donzé and Oded Maler. 2010. Robust Satisfaction of Temporal Logic over Real-Valued Signals. In *Formal Modeling and Analysis of Timed Systems*. 92–106.

[Erez et al.(2011)] Tom Erez, Yuval Tassa, and Emanuel Todorov. 2011. Infinite-Horizon Model Predictive Control for Periodic Tasks with Contacts. In *Robotics: Science and Systems*.

[Fulton and Platzer(2018)] Nathan Fulton and André Platzer. 2018. Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* 32, 1 (Apr. 2018).

[Haarnoja et al.(2018)] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International Conference on Machine Learning (ICML), 2018*. 1861–1870.

[Hafner and Riedmiller(2011)] Roland Hafner and Martin A. Riedmiller. 2011. Reinforcement learning in feedback control. *Machine Learning* 84 (2011), 137–169.

[Jakšić et al.(2018)] Stefan Jakšić, Ezio Bartocci, Radu Grosu, Thang Nguyen, and Dejan Niă?Ković. 2018. Quantitative monitoring of STL with edit distance. *Formal Methods in System Design* 53, 1 (Aug. 2018), 83–112.

[Kindler(1994)] Ekkart Kindler. 1994. Safety and Liveness Properties: A Survey. *EATCS-Bulletin* 53 (1994).

[Konda and Tsitsiklis(2003)] Vijay R. Konda and John N. Tsitsiklis. 2003. OnActor-Critic Algorithms. *SIAM Journal on Control and Optimization* 42, 4 (2003), 1143–1166.

[Lamport(2000)] Leslie Lamport. 2000. Fairness and Hyperfairness. *Distributed Computing* 13, 4 (Nov 2000), 239–245.

[Levine et al.(2016)] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research* 17, 1 (Jan. 2016), 1334–1373.

[Levine and Koltun(2013)] Sergey Levine and Vladlen Koltun. 2013. Guided Policy Search. In *International Conference on Machine Learning (ICML) (Proceedings of Machine Learning Research, Vol. 28)*. Atlanta, Georgia, USA, 1–9.

[Li et al.(2018)] Xiao Li, Yao Ma, and Calin Belta. 2018. A Policy Search Method For Temporal Logic Specified Reinforcement Learning Tasks. In *American Control Conference (ACC), 2018*. 240–245.

[Lillicrap et al.(2016)] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016*.

[Mehdipour et al.(2019)] Noushin Mehdipour, Cristian Ioan Vasile, and Calin Belta. 2019. Arithmetic-Geometric Mean Robustness for Control from Signal Temporal Logic Specifications. In *American Control Conference (ACC)*. IEEE, 1690–1695.

[Mnih et al.(2015)] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518 (2015), 529–533.

[Nachum et al.(2018)] Ofir Nachum, Mohammad Norouzi, George Tucker, and Dale Schuurmans. 2018. Smoothed Action Value Functions for Learning Gaussian Policies. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, Vol. 80. PMLR, 3692–3700.

[Ničković and Yamaguchi(2020)] Dejan Ničković and Tomoya Yamaguchi. 2020. RTAMT: Online Robustness Monitors from STL. In *Automated Technology for Verification and Analysis (ATVA)*. 564–571.

[Pant et al.(2017)] Yash Vardhan Pant, Houssam Abbas, and Rahul Mangharam. 2017. Smooth operator: Control using the smooth robustness of temporal logic. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*. 1235–1240.

[Raffin et al.(2021)] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research (JMLR)* 22, 268 (2021), 1–8.

[Raman et al.(2014)] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M. Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A. Seshia. 2014. Model Predictive Control with Signal Temporal Logic Specifications. In *Proceedings of the 53rd IEEE Conference on Decision and Control (CDC)*. 81–87.

[Raman et al.(2015)] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M. Murray, and Sanjit A. Seshia. 2015. Reactive Synthesis from Signal Temporal Logic Specifications. In *Proceedings of the 8th International Conference on Hybrid Systems: Computation and Control (HSCC 2015)*. 239–248.

[Schulman et al.(2016)] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2016. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *Proceedings of the International Conference on Learning Representations (ICLR), 2026.*

[Shen et al.(2020)] Qianli Shen, Yan Li, Haoming Jiang, Zhaoran Wang, and Tuo Zhao. 2020. Deep Reinforcement Learning with Robust and Smooth Policy. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*. JMLR, Article 808, 12 pages.

[Silver et al.(2017)] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, L. Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550 (2017), 354–359.

[Singh and Saha(2021)] Nikhil Kumar Singh and Indranil Saha. 2021. Specification Guided Automated Synthesis of Feedback Controllers. *ACM Trans. Embed. Comput. Syst.* 20, 5 (2021).

[Singh and Saha(2023)] Nikhil Kumar Singh and Indranil Saha. 2023. STL-based synthesis of feedback controllers using reinforcement learning. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI'23)*. AAAI Press, Article 1695, 9 pages. https://doi.org/10.1609/aaai.v37i12.26764

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

[Tassa et al.(2012)] Yuval Tassa, Tom Erez, and Emanuel Todorov. 2012. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 4906–4913.

[Várnai and Dimarogonas(2020)] Péter Várnai and Dimos V. Dimarogonas. 2020. On Robustness Metrics for Learning STL Tasks. In *2020 American Control Conference, ACC 2020, Denver, CO, USA, July 1-3, 2020*. IEEE, 5394–5399.

[Wawrzyński(2009)] Paweł Wawrzyński. 2009. A Cat-Like Robot Real-Time Learning to Run. In *Adaptive and Natural Computing Algorithms*, Mikko Kolehmainen, Pekka Toivanen, and Bartlomiej Beliczynski (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 380–390.

[Xie(2026)] Jinyu Xie. 2026. Simglucose v0.2.1 (2018) [Online]. https://github.com/jxx123/simglucose

Received Received 29 May 2025; accepted 02 October 2025