STL-Based Synthesis of Feedback Controllers Using Reinforcement Learning

Nikhil Kumar Singh and Indranil Saha

Department of Computer Science and Engineering
IIT Kanpur
{nksingh, isaha}@cse.iitk.ac.in

Abstract

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.

1 Introduction

Feedback controllers form the core of any safety-critical cyber-physical systems (CPSs). 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, Deep Reinforcement Learning (DRL) has been extremely popular in solving highly complex problems such as solving Atari (Mnih et al. 2015) and Go (Silver et al. 2017), 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 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, Neumann, and Peters 2013; Fulton and Platzer 2018; Hafner and Riedmiller 2011)). In this approach, 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 being used 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), (Jaksic 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, Vasile, and Belta 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, Ma, and Belta 2018; Pant, Abbas, and Mangharam 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. 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.

2 Preliminaries

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 $\mathcal{C}$ 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, $i \in \mathbb{N}$, 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

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

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, Ferrère, and Maler 2013). We use Euclidean metric as the norm to measure the distance d between two values $v, v' \in \mathbb{R}$, i.e., $d(v, v') = \parallel v - v' \parallel$. 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:

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

Intuitively, $\texttt{Dist}(v, A)$ 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 $\mathsf{O} : \mathcal{K} \to 2^{\mathcal{X}}$ to denote the mapping of a predicate κ to a set of states ($\mathcal{X}$). 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:

$$\rho(\texttt{true}, \omega, t) = +\infty, \tag{3a}$$

$$\rho(\kappa, \omega, t) = \texttt{Dist}(x_t, \mathsf{O}(\kappa)), \tag{3b}$$

$$\rho(\neg\phi, \omega, t) = -\rho(\phi, \omega, t), \tag{3c}$$

$$\rho(\phi \wedge \psi, \omega, t) = \texttt{min}(\rho(\phi, \omega, t), \rho(\psi, \omega, t)), \tag{3d}$$

$$\rho(\phi \, U_I \, \psi, \omega, t) = \sup_{t' \in t+I} \texttt{min}(\rho(\psi, \omega, t'), \inf_{t'' \in [t, t']} \rho(\phi, \omega, t'')). \tag{3e}$$

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, Vasile, and Belta 2019), Logarithm-summation-exponential (Li, Ma, and Belta 2018; Pant, Abbas, and Mangharam 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. 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, we change the definition of U_I operator in the classical robustness semantics of STL (Equation 3).

$$\bar{\rho}(\phi\, U_I\, \psi, \omega, t) = \sup_{t' \in t-I} \mathtt{min}(\bar{\rho}(\psi, \omega, t'), \inf_{t'' \in [t',t]} \bar{\rho}(\phi, \omega, t'')).$$

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:

$$\begin{aligned}
h(\mathsf{true}, \omega, t) &= 0, && \text{(4a)}\\
h(\kappa, \omega, t) &= 0, && \text{(4b)}\\
h(\neg\phi, \omega, t) &= h(\phi, \omega, t), && \text{(4c)}\\
h(\phi \wedge \psi, \omega, t) &= \mathtt{max}(h(\phi, \omega, t), h(\psi, \omega, t)), && \text{(4d)}\\
h(\phi \vee \psi, \omega, t) &= \mathtt{max}(h(\phi, \omega, t), h(\psi, \omega, t)), && \text{(4e)}\\
h(\Diamond_I \phi, \omega, t) &= I + h(\phi, \omega, t), && \text{(4f)}\\
h(\Box_I \phi, \omega, t) &= I + h(\phi, \omega, t), && \text{(4g)}\\
h(\phi\, U_I\, \psi, \omega, t) &= I + \mathtt{max}(h(\phi, \omega, t), h(\psi, \omega, t)). && \text{(4h)}
\end{aligned}$$

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.

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 $\mathcal{C}^*$ 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,*

$$\mathcal{C}^* = \arg\max_{\mathcal{C}} \mathop{\mathbb{E}}_{\omega \sim \mathcal{C}} [\rho(\phi, \omega)]. \tag{5}$$

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

$$CC = \frac{1}{T} \cdot \sum_{t=1}^{T} CC_t^2 \tag{6}$$

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

$$DC = x_T^k \tag{7}$$

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*

$$MoS = \frac{1}{T} \cdot \sum_{t=1}^{T} \rho(\phi, \omega, t) \tag{8}$$

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*

$$SAT = \mathbb{I}[min(\{\rho(\phi_s, \omega, t)\}_{t=1}^{T})] \tag{9}$$

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

Unlike these metrics, the gym provides a reward function for each environment which is applicable to that specific environment only. We use this metric also in our evaluaation and refer 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 Methodology

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, Erez, and Todorov 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 $\mathcal{T} : \mathbb{R}^{p \times q \times p} \rightarrow [0, \infty)$ 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.

$$\pi_\theta(u|x) = \mathbb{P}[u|x; \theta]. \tag{10}$$

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

$$J(\theta) = \mathop{\mathbb{E}}_{\omega \sim \pi_\theta} [\bar{\rho}(\phi, \omega)], \tag{11}$$

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,

$$\theta^* = \arg\max_\theta \ J(\theta) \tag{12}$$

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:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \ [\sum_{t=0}^{T-1} \ \nabla_\theta \ log \ \pi_\theta(u_t|x_t) \cdot Q(x_t, u_t)] \tag{13}$$

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

$$Q(x_t, u_t) = \mathbb{E}_{\pi_\theta} [\sum_{t=t'}^{T} \bar{\rho}(\phi, \omega_{[t'-h(\phi), t']}) | x_t, u_t] \tag{14}$$

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 (13), 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 (13) 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, Ma, and Belta 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.

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. Our `SSS` semantics (introduced in the next subsection), though not sound, addresses the shadowing problem and is smooth as well.

5.1 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

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

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

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

$$\min(x_1, x_2) = \frac{(x_1 + x_2) - |\, x_1 - x_2 \,|}{2} \tag{15}$$

We extend this definition to approximate the robustness for AND as

$$\mathcal{A}(\rho_1, \ldots, \rho_n) = \frac{\sum_{i=1}^{n} \rho_i - |\, \max_i(\rho_i) - \min_i(\rho_i) \,|}{n} \tag{16}$$

We formulate the approximation for AND in expression (16) because of two reasons - firstly, we want to use an aggregate measure of all the robustness in the AND expression, and secondly, we want to penalize the largest difference between any two robustness. Let us denote this largest difference as δ^{max}.

In Equation 16, the non-smoothness of the AND operator is because of two reasons - the modulus function ($|.|$) and the max/min function. The function $|\, x \,|$ is not smooth (differentiable) at 0 and hence can be approximated by a smooth function (Bagul and Chesneau 2020). We found the most suitable for approximation of $|\, x \,|$ as $x.\texttt{erf}(\mu.x)$ using the Gaussian error function `erf` where μ is a smoothness parameter. The `erf` function is defined as

$$\texttt{erf}(x) = \frac{2}{\pi} \int_0^x e^{-t^2} dt. \tag{17}$$

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

$$\mathcal{A}(\rho_1, \ldots, \rho_n) = \frac{\sum_{i=1}^{n} \rho_i - \{\delta^{\max} \cdot \texttt{erf}(\mu \cdot \delta^{\max})\}}{n} \tag{18}$$

where $\delta^{max} = \max_i(\rho_i) - \min_i(\rho_i)$.

The non-smoothness w.r.t the max/min function in the δ^{max} expression is addressed as follows:

$$\delta^{max} \approx \frac{\log(n + \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} \exp(\eta \cdot (\rho_i - \rho_j)))}{\eta}. \tag{19}$$

Please see the full version (Singh and Saha 2022) for derivation.

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 (18) to the traditional $\min$ function. We define our semantics with a high η and low μ value. A 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.

5.2 Properties Satisfied by SSS Semantics

We now present various useful properties satisfied by the semantics of AND in Equation (18).

Theorem 1 (Properties of SSS semantics)**.** *The STL semantics generated by operator $\mathcal{A}$ in Equation* (18) *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:

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

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

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

Proof. Please see the full version (Singh and Saha 2022). □

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, Ma, and Belta 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.

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

Environment	#Observations	#Actions
HalfCheetah (HC)	17	6
Hopper (Hop)	11	3
Ant	27	8
Walker (Wkr)	17	6
Swimmer (Swm)	8	2
Humanoid (Hum)	376	17

Table 2: Benchmarks

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* (18) *is sound in aggregate sense i.e., it satisfies the following conditions:*

$$1)\ \mathcal{A}(\rho_1, \ldots, \rho_n) > 0, \qquad \textit{if}\ \forall i \in [1, n]\ \ \rho_i > 0$$

$$2)\ \mathcal{A}(\rho_1, \ldots, \rho_n) < 0, \qquad \textit{if}\ \forall i \in [1, n]\ \ \rho_i < 0$$

$$3)\ \mathcal{A}(\rho_1, \ldots, \rho_n) > -\frac{1}{n}[|\rho^{max}| + |\rho^{min}|], \quad \textit{if} \sum_{i=1}^{n} \rho_i > 0, \quad \rho^{max} > 0,\ \ \rho^{min} < 0$$

$$4)\ \mathcal{A}(\rho_1, \ldots, \rho_n) < 0, \qquad \textit{if} \sum_{i=1}^{n} \rho_i < 0, \rho^{max} > 0, \quad \rho^{min} < 0.$$

where $\rho^{max} = \max_i \rho_i$ *and* $\rho^{min} = \min_i \rho_i\ \forall i \in [1, n]$.

Proof. Please see the full version (Singh and Saha 2022). □

6 Experiments

This section present 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

Implementation. The controllers used in our experiments are synthesized in a Deep-RL setting with continuous state space and continuous action space. 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, Tassa, and Todorov 2011) (Hop), Ant (Schulman et al. 2016), Walker (Erez, Tassa, and Todorov 2011) (Wkr), Swimmer (Coulom 2002) (Swm), and Humanoid (Erez, Tassa, and Todorov 2011) (Hum). The number of observable states and actions for the benchmarks are shown in Table 2. In Table 3, 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.

Hyper-parameters. There are different hyper-parameters used in various semantics of STL. We have used the hyper-parameter values ($\beta = 1$, $\nu = 3$) used in the work (Li, Ma, and Belta 2018; Várnai and Dimarogonas 2020). For our `SSS` semantics, we carry out initial experiments on the Hopper benchmark to understand the effect of the hyper-parameters 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 4 presents the performance of the controllers synthesized using different semantics for various benchmarks. We

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

Table 3: STL specifications

Env	Sem	CC	DC	MoS	SAT	DR
HC	Classical	16127.17	307.66	7.30	-	5755.00
	`AGM`	16599.73	406.29	10.03	-	7716.32
	`LSE`	15311.38	111.86	4.25	-	1850.36
	`Softmax`	14154.17	471.29	10.76	-	9047.92
	`SSS`	14647.78	541.90	12.04	-	10453.96
Hop	Classical	1698.20	12.08	0.68	99/100	2505.08
	`AGM`	928.52	12.72	0.58	0/100	2196.76
	`LSE`	42.81	0.02	-0.26	0/100	31.91
	`SoftMax`	1571.84	13.99	0.69	100/100	2745.39
	`SSS`	1231.04	17.25	0.69	100/100	3153.21
Ant	Cls	12055.73	34.97	0.32	99/100	13.95
	`AGM`	9089.84	40.04	0.31	100/100	366.71
	`LSE`	220.42	0.32	0.21	100/100	-11.70
	`SMax`	10938.52	38.61	0.32	100/100	150.44
	`SSS`	12678.64	85.03	0.30	100/100	967.26
Wkr	Classical	5614.93	8.56	0.71	99/100	2056.56
	`AGM`	76.63	-0.07	-0.37	0/100	-1.54
	`LSE`	368.15	-0.08	-0.14	0/100	44.25
	`SoftMax`	10359.56	-4.53	0.33	94/100	413.97
	`SSS`	10743.50	18.71	0.59	100/100	3333.06
Swm	Cls	626.38	-1.59	0.37	100/100	-40.05
	`AGM`	87.48	0.78	0.00	100/100	19.46
	`LSE`	56.31	0.25	-0.06	100/100	6.35
	`SMax`	573.94	0.05	0.07	3/100	0.98
	`SSS`	1256.99	7.18	0.37	99/100	179.09
Hum	Classical	2072.98	6.66	0.28	99/100	5396.01
	`AGM`	3143.56	18.43	0.24	100/100	6352.30
	`LSE`	1531.31	2.25	0.09	100/100	5060.10
	`SoftMax`	2313.87	7.94	0.28	100/100	5505.03
	`SSS`	2901.91	26.79	0.27	100/100	7054.92

Table 4: Comparison of different semantics under different environments

tested the synthesized controller on 100 distinct seed inputs for better coverage of the behavior of these controllers. In Table 4, we show the mean values (100 seeds) for all the metrics. The complete results (along with standard deviation) can be found in (Singh and Saha 2022). 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 4 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`, 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 a movement 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 $100/100$ satisfaction 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 99/100 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 100/100 satisfaction of the safety property. The `Softmax` semantics performs the worse, failing to satisfy 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.

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 the aggregate-based STL semantics proposed in the literature. Going forward, we plan to explore the application of our semantics in other complex domains like self-driving cars.

Acknowledgements

The authors would like to thank anonymous reviewers for their valuable feedback. The authors also thank Dejan Nickovic for his support in working with the RTAMT Toolbox.

References

Alpern, B.; and Schneider, F. B. 1987. Recognizing Safety and Liveness. *Distrib. Comput.*, 2(3): 117–126.

Bagul, Y. J.; and Chesneau, C. 2020. Sigmoid functions for the smooth approximation to the absolute value function. *Moroccan Journal of Pure and Applied Analysis*, 7: 12 – 19.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. *CoRR*, abs/1606.01540.

Coulom, R. 2002. *Reinforcement Learning Using Neural Networks, with Applications to Motor Control*. Ph.D. thesis, Institut National Polytechnique de Grenoble.

Deisenroth, M. P.; Neumann, G.; and Peters, J. 2013. A Survey on Policy Search for Robotics. *Found. Trends Robotics*, 2: 1–142.

Donzé, A.; Ferrère, T.; and Maler, O. 2013. Efficient Robust Monitoring for STL. In *CAV*, 264–279.

Donzé, A.; and Maler, O. 2010. Robust Satisfaction of Temporal Logic over Real-Valued Signals. In *Formal Modeling and Analysis of Timed Systems*, 92–106.

Erez, T.; Tassa, Y.; and Todorov, E. 2011. Infinite-Horizon Model Predictive Control for Periodic Tasks with Contacts. In *Robotics: Science and Systems*.

Fulton, N.; and Platzer, A. 2018. Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning. *AAAI*, 32(1).

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *ICML*, 1861–1870.

Hafner, R.; and Riedmiller, M. A. 2011. Reinforcement learning in feedback control. *Machine Learning*, 84: 137–169.

Jaksic, S.; Bartocci, E.; Grosu, R.; Nguyen, T.; and Ničković, D. 2018. Quantitative Monitoring of STL with Edit Distance. *Form. Methods Syst. Des.*, 53(1): 83–112.

Kindler, E. 1994. Safety and Liveness Properties: A Survey. *EATCS-Bulletin*, 53.

Konda, V. R.; and Tsitsiklis, J. N. 2003. OnActor-Critic Algorithms. *SIAM Journal on Control and Optimization*, 42(4): 1143–1166.

Lamport, L. 2000. Fairness and Hyperfairness. *Distrib. Comput.*, 13(4): 239–245.

Levine, S.; Finn, C.; Darrell, T.; and Abbeel, P. 2016. End-to-End Training of Deep Visuomotor Policies. *J. Mach. Learn. Res.*, 17(1): 1334–1373.

Levine, S.; and Koltun, V. 2013. Guided Policy Search. In *ICML*, volume 28 of *Proceedings of Machine Learning Research*, 1–9. Atlanta, Georgia, USA: PMLR.

Li, X.; Ma, Y.; and Belta, C. 2018. A Policy Search Method For Temporal Logic Specified Reinforcement Learning Tasks. In *ACC*, 240–245.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. In *ICLR*.

Mehdipour, N.; Vasile, C. I.; and Belta, C. 2019. Arithmetic-Geometric Mean Robustness for Control from Signal Temporal Logic Specifications. In *ACC*, 1690–1695. IEEE.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518: 529–533.

Nachum, O.; Norouzi, M.; Tucker, G.; and Schuurmans, D. 2018. Smoothed Action Value Functions for Learning Gaussian Policies. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, 3692–3700. PMLR.

Ničković, D.; and Yamaguchi, T. 2020. RTAMT: Online Robustness Monitors from STL. In *Automated Technology for Verification and Analysis*, 564–571.

Pant, Y. V.; Abbas, H.; and Mangharam, R. 2017. Smooth operator: Control using the smooth robustness of temporal logic. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, 1235–1240.

Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; and Dormann, N. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268): 1–8.

Raman, V.; Donzé, A.; Sadigh, D.; Murray, R. M.; and Seshia, S. A. 2015. Reactive Synthesis from Signal Temporal Logic Specifications. In *HSCC*, 239–248.

Raman, V.; Donzé, A.; Maasoumy, M.; Murray, R. M.; Sangiovanni-Vincentelli, A.; and Seshia, S. A. 2014. Model predictive control with signal temporal logic specifications. In *CDC*, 81–87.

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; and Abbeel, P. 2016. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *ICLR*.

Shen, Q.; Li, Y.; Jiang, H.; Wang, Z.; and Zhao, T. 2020. Deep Reinforcement Learning with Robust and Smooth Policy. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*. JMLR.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T. P.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the game of Go without human knowledge. *Nature*, 550: 354–359.

Singh, N. K.; and Saha, I. 2021. Specification Guided Automated Synthesis of Feedback Controllers. *ACM Trans. Embed. Comput. Syst.*, 20(5).

Singh, N. K.; and Saha, I. 2022. STL-Based Synthesis of Feedback Controllers Using Reinforcement Learning. *CoRR*.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book. ISBN 0262039249.

Tassa, Y.; Erez, T.; and Todorov, E. 2012. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *IROS*.

Várnai, P.; and Dimarogonas, D. V. 2020. On Robustness Metrics for Learning STL Tasks. In *ACC*, 5394–5399.

Wawrzyński, P. 2009. A Cat-Like Robot Real-Time Learning to Run. In Kolehmainen, M.; Toivanen, P.; and Beliczynski, B., eds., *Adaptive and Natural Computing Algorithms*, 380–390. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-04921-7.