Backpropagation for Parametric STL

Karen Leung,[1] Nikos Aréchiga[2] and Marco Pavone[1]

***Abstract*— This paper proposes a method to evaluate Signal Temporal Logic (STL) robustness formulas using computation graphs. This method results in efficient computations and enables the use of backpropagation for optimizing over STL parameters. Inferring STL formulas from behavior traces can provide powerful insights into complex systems, such as long-term behaviors in time-series data. It can also be used to augment existing prediction and planning architectures by ensuring specifications are met. However, learning STL formulas from data is challenging from a theoretical and numerical standpoint. By evaluating and learning STL formulas using computation graphs, we can leverage the computational efficiency and utility of modern machine learning libraries. The proposed approach is particularly effective for solving parameteric STL (pSTL) problems, the problem of parameter fitting for a given signal. We provide a relaxation technique that makes this method tractable when solving general pSTL formulas. Through a traffic-weaving case-study, we show how the proposed approach is effective in learning pSTL parameters, and how it can be applied for scenario-based testing for autonomous driving and other complex robotic systems.**

I. Introduction

The "rules of the road", for the most part, govern how people behave while driving, thus high-level behaviors can often be described using rule-based or logic-based techniques. However, this depends on domain knowledge, and not all the rules are always met as there is a spectrum in how individuals obey these rules. For example, some drivers tend to tail gate, while others maintain a three-second gap. In this paper, we develop a testing strategy for autonomous vehicles that make use of this rule-based domain knowledge. For this purpose, we need a modeling language that offers rigidity with regards to the rule-based nature of driving, but also some flexibility to describe the spectrum in which these rules may be followed.

Signal Temporal Logic (STL) [1], [2] is an expressive formal language that can specify properties of both continuous and hybrid systems, and give a measure of how much those properties are satisfied. It can deal with real-valued dense-time signals, making it a versatile and useful tool in many robotics applications such as autonomous driving. In addition to Boolean semantics, STL is equipped with *quantitative* semantics. It can provide a robustness value—a continuous real-valued scalar that represents the *degree of satisfaction* over the specification. As such, STL is a very attractive tool to model human behaviors, such as how one drives on the road, as it can describe high-level specifications

*This work was supported by Toyota Research Institute

[1]Karen Leung and Marco Pavone are with the Department of Aeronautics and Astronautics, Stanford University, Stanford, CA, 94305, USA. {karenl7, pavone}@stanford.edu

[2]Nikos Aréchiga is with Toyota Research Institute, Los Altos, CA, 94022, USA. nikos.arechiga@tri.global

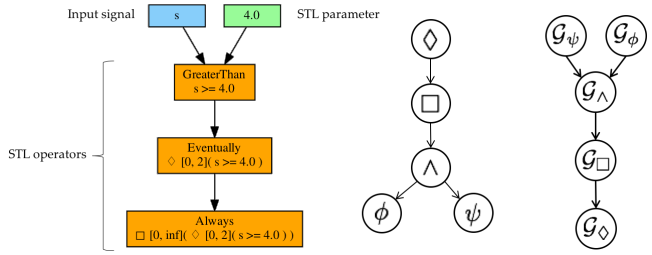


Fig. 1: Left: Graphical representation of the STL formula $\phi = \Diamond\Box_{[0,2]}(s > 4)$ from Example 1. This is generated from the toolbox which implements the proposed method. Right: Representation of $\theta = \Diamond\Box(\phi \wedge \psi)$ from Example 2. $\mathcal{G}$ (far right) of a formula θ is constructed by connecting subgraphs whose node represent each operation. The topological ordering of $\mathcal{G}$ is the post-order traversal of the parse tree $\mathcal{PT}$ (left) generated by θ.

and provide a measure of how much the specification is met. Recently, STL has gained traction in the learning and controls community due to its ability to specify behaviors over a longer time horizon. For example, STL has been used as constraints in model predictive control [3] and stochastic control [4], [5], [6] problems, and as rewards in reinforcement learning [7], [8], [9].

The inverse problem, constructing an STL specification given a signal, is also an interesting yet challenging problem. It is interesting because learning STL formulas from complex time-series data can offer valuable insight and interpretability into the structure and properties of the system, and potentially improve other methods that rely on the signal, such as classification, prediction, and decision-making [6], [10], [11], [12]. However, the inverse problem is challenging because the space of STL formulas is infinitely large and the solution is not unique. A more tractable approach is parametric STL (pSTL) [13]. A pSTL formula is a template STL formula, and given time series data, it is possible to optimize over the parameters to find the values that provide the tightest fit in terms of robustness. Using pSTL formulas in this way is especially powerful when the formula comes from domain knowledge. This parameter identification can be thought as a form of feature extraction from time series data. However, a key drawback of existing techniques [10], [14] is that only monotonic formulas are supported, i.e. formulas whose satisfaction depends monotonically on the parameters, and the analysis is carried out in isolation.

In this paper, we develop a technique that leverages the efficiency of computation graphs and use smooth differentiable approximations to tractably and efficiently learn parameters of complex, possibly non-monotonic, pSTL for-

mulas. Smooth approximations to the robustness formulas is a common approach to ensure tractable computations and has been used in other contexts such as optimal control [15], and reinforcement learning [9]. In this work however, by using computation graphs and smooth approximations, we are able to leverage state-of-the-art machine learning tools to create an efficient framework for evaluating the robustness of STL formulas, and solve pSTL problems. The benefits afforded to us by using state-of-the-art machine learning tools include autodifferentiation, batching, ease of parallelization, and ability to combine pSTL within a larger learning and/or modeling framework.

Learning logical formulas from data can be a powerful analysis tool, as it offers insight into the structure and logical properties of a system in an unsupervised way [6], [16]. However, learning the logical structure directly from data is very challenging as the space of logical formulas is extremely large. There has been work on mining formulas from data, but tractable methods often limit the class of learnable formulas and can be computationally demanding [11], [12]. A proxy for logical structure is to use decision trees, in the sense that logical predicates are tested for truth or falsehood at each node of the tree. [17], [18] provide a differentiable formulation for decision trees that can be trained by backpropagation. However, decision trees do not have an internal state, which means that they do not provide the same dynamic richness as many temporal logic formulas (e.g., STL) which can express and monitor properties that evolve over time.

Instead of attempting to learn complete STL formulas from scratch, a common approach is to instead learn parameters of a parametric STL formula (pSTL) [1]. Existing methods [10], [13], [14] provide efficient algorithms to solve for parameters exactly but are restricted to monotonic formulas, and use a local search method or involve additional computations such as finding validity domains. Although monotonic formulas represent a broad class of STL formulas and the existing methods are powerful in their own right, we aim to relax the pSTL problem to develop a broader framework with minimal overhead and can be easily integrated within a larger modeling framework.

Statement of Contribution: First, we present a method for constructing a computation graph to evaluate exactly the robustness of an STL formula and show that this method is sound and complete. Second, using smooth approximations to the robustness formulas, we provide an algorithm for optimizing pSTL formulas using backpropagation. Third, we demonstrate our method on a traffic-weaving case study. We show that we are able to efficiently learn pSTL parameters and use them to gain interpretability into the system.

Organization: In Section II, we introduce our problem formulation and give a formal definition of STL and pSTL. In Section III, we detail our proposed computation graph method and show that it is sound and complete. We also provide examples of our approach. In Section IV, we describe the smooth relaxation on the robustness formula and present the overall optimization for solving a pSTL problem using computation graphs. We demonstrate our approach with an automotive case study in Section V. Finally, we conclude and outline the future directions of this framework in Section VI.

II. Problem Formulation

The objectives of this paper are to (1) provide a lightweight and efficient method to compute STL robustness values, and (2) leverage readily available machine learning frameworks to efficiently fit pSTL parameters. Next, we define STL and its parametric extension, pSTL.

A. Signal Temporal Logic: Syntax and Semantics

STL is a specification language for *real-valued signals* that can be applied to continuous and hybrid systems. A timed trace, or a *signal* is a data structure that contains a time-series. Formally, a timed trace s is an ordered finite sequence of states and their associated time, $s = (\mathbf{x}_0, t_0), ..., (\mathbf{x}_n, t_n)$ where $t_{i-1} < t_i$ and $\mathbf{x}_i \in \mathbb{R}^n$. Further we use the notation $s(t_i) = \mathbf{x}_i$.

Given a timed trace s that starts at time t_0, a timed trace subsequence (s, t_i) is a timed trace starting at time t_i, $i \in \mathbb{N}_0$.

$$(s, t_i) = (\mathbf{x}_i, t_i),\ (\mathbf{x}_{i+1}, t_{i+1}),\ ...,(\mathbf{x}_n, t_n)$$

Further, we use the following notation for a subsequence $s_i = (s, t_i)$. Additionally, we make the following assumption to ensure the dimensions of the vectors in our approach are consistent. If the timesteps are not uniformly spaced, the signal can be interpolated accordingly.

Assumption 1: The time steps are uniformly spaced.

STL formulas are defined over predicates of the form $f(s) < c$, where s is a timed trace (signal), $f : \mathbb{R}^n \to \mathbb{R}$ is a function and $c \in \mathbb{R}$. STL formulas are written using the following grammar:

$$\begin{aligned} I &:= \ (a, b) \mid (a, b] \mid [a, b) \mid [a, b] \\ \phi &:= \ true \mid f(s) < c \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \\ &\quad \mid \Diamond_I \phi \mid \Box_I \phi \mid \phi \ \mathcal{U}_I \ \psi \mid \phi \ \mathcal{T}_I \ \psi \end{aligned}$$

where $f(s) < c$ is a predicate, $\neg$ (negation/not), $\wedge$ (conjunction/and), and $\vee$ (disjunction/or) are Boolean connectives, and $\Diamond$ (eventually), $\Box$ (always), $\mathcal{U}$ (until) and $\mathcal{T}$ (then) are temporal operators. The temporal operators have an associated time interval $I \in [a, b]$ where $0 \leq a < b$. For ease of notation, I is dropped from the grammar when $a = 0$, $b = \infty$. Further we make the following assumption,

Assumption 2: Without loss of generality, we only consider I to be of the form $[a, b]$ or $[a, \infty)$ where $0 \leq a < b$. By Assumption 1, if I does not include the endpoints, we can change the boundaries to the nearest inclusive time step.

Let (s, t) be a timed trace starting at time t. The *Boolean semantics* of STL can be defined recursively:

$$
\begin{aligned}
(s,t) \models f(s(t)) < c \quad &\Leftrightarrow f(s(t)) < c \\
(s,t) \models \neg\phi \quad &\Leftrightarrow \neg((s,t) \models \phi) \\
(s,t) \models \phi \wedge \psi \quad &\Leftrightarrow ((s,t) \models \phi) \wedge ((s,t) \models \psi) \\
(s,t) \models \phi \vee \psi \quad &\Leftrightarrow ((s,t) \models \phi) \vee ((s,t) \models \psi) \\
(s,t) \models \Diamond_I \phi \quad &\Leftrightarrow \exists t' \in I \oplus t \;\; s.t. \; (s,t') \models \phi \\
(s,t) \models \Box_I \phi \quad &\Leftrightarrow \forall t' \in I \oplus t \;\; s.t. \; (s,t') \models \phi \\
(s,t) \models \phi \mathcal{U}_I \psi \quad &\Leftrightarrow \exists t' \in I \oplus t \;\; s.t. \; ((s,t') \models \psi) \wedge \\
&\qquad ((s,t) \models \Box_{[0,t']} \phi) \\
(s,t) \models \phi \mathcal{T}_I \psi \quad &\Leftrightarrow \exists t' \in I \oplus t \;\; s.t. \; ((s,t') \models \psi) \wedge \\
&\qquad ((s,t) \models \Diamond_{[0,t']} \phi)
\end{aligned}
$$

For a timed trace (s,t) starting at time t, satisfying $\Box\phi$ means ϕ is always true for the entire sequence (since the I is dropped, $I = [0,\infty)$). Since STL specifications are defined recursively, temporal operators can be composed with each other. For example, a timed trace (s,t) satisfying $\Diamond\Box_{[1,3]}\phi$ means that *eventually* ϕ will *always* be satisfied over the next 1-3 time units.

Since STL formulas are defined recursively, we can represent STL using a parse tree where each node represents an operation, and the leaves are the predicates. We denote the order of operations for a formula ϕ by $\mathcal{O}_\phi$. More formally,

Definition 1 (STL ordering): For an STL formula ϕ, let $\mathcal{PT}$ be the parse tree of ϕ. Define $\mathcal{O}_\phi = \{\varphi_1, \varphi_2, ...\varphi_k\}$ as the post-order traversal of $\mathcal{PT}$.

A special property of STL is the notion of *robustness*, a real value that describes how much a timed trace satisfies (or fails to satisfy) an STL specification. For example, if a signal does not satisfy an STL specification, the robustness value will indicate the level of violation, and vice versa.

The robustness degree can be calculated recursively according to the *quantitative semantics*:

$$
\begin{aligned}
\rho(s,t,true) \quad &= \rho_{\max} \\
\rho(s,t,f(s) < c) \quad &= c - f(s(t)) \\
\rho(s,t,\neg\phi) \quad &= -\rho(s,t,\phi) \\
\rho(s,t,\phi \wedge \psi) \quad &= \min(\rho(s,t,\phi), \rho(s,t,\psi) \\
\rho(s,t,\phi \vee \psi) \quad &= \max(\rho(s,t,\phi), \rho(s,t,\psi) \\
\rho(s,t,\Diamond_I \phi) \quad &= \max_{t' \in I \oplus t} \rho(s,t',\phi) \\
\rho(s,t,\Box_I \phi) \quad &= \min_{t' \in I \oplus t} \rho(s,t',\phi) \\
\rho(s,t,\phi \mathcal{U}_I \psi) \quad &= \max_{t' \in I \oplus t} (\min(\rho(s,t',\psi), \\
&\qquad \min_{t'' \in [t,t']} \rho(s,t'',\phi))) \\
\rho(s,t,\phi \mathcal{T}_I \psi) \quad &= \max_{t' \in I \oplus t} (\min(\rho(s,t',\psi), \\
&\qquad \max_{t'' \in [t,t']} \rho(s,t'',\phi)))
\end{aligned}
$$

Note that there is no difference in using a strict, or non-strict inequality, the robustness value does not change.

We define a *robustness trace* to describe the robustness value of each timed trace subsequence.

Definition 2 (Robustness Trace): Given a timed trace s starting at time t_0 and an STL formula ϕ, the robustness trace $\tau(s,t_0,\phi)$ is a sequence of robustness values for each subsequence s_i of s. That is:

$$
\begin{aligned}
\tau(s,t_0,\phi) &= \tau_0, \tau_1, ...\tau_n \\
&= \rho(s,t_0,\phi), \rho(s,t_1,\phi), ..., \rho(s,t_n,\phi) \\
&= \rho(s_0,\phi), \rho(s_1,\phi), ..., \rho(s_n,\phi)
\end{aligned}
$$

The robustness trace is particularly useful for computing temporal operations.

Example 1: Let $s = (1,0),\ (2,1),\ (3,2),\ (4,3),\ (3,4),\ (3,5)$ and $q = (1,0),\ (2,1),\ (3,2),\ (4,3),\ (5,4),\ (6,5)$. Consider the specification $\phi = \Diamond\Box_{[0,2]}(s > 4)$. For the timed trace s, ϕ is not satisfied since the signal never exceeds four, while q does satisfy ϕ. The robustness trace for ϕ's subformula $\Box_{[0,2]}(s > 4)$ is

$$
\begin{aligned}
\tau(s,t,\Box_{[0,2]}(s > 4)) &= -3, -2, -1, -1, -\rho_{\max}, -\rho_{\max} \\
\tau(q,t,\Box_{[0,2]}(s > 4)) &= -3, -2, -1, 0, -\rho_{\max}, -\rho_{\max}.
\end{aligned}
$$

When $t \oplus I$ is beyond the length of the timed trace, the robustness is equal to $-\rho_{\max}$ because ϕ is not satisfied[1]. Then the robustness is computed by taking the $\max$ over the robustness trace of the subformula, $\rho(s,\phi) = -1$, $\rho(q,\phi) = 0$.

B. Parametric Signal Temporal Logic

A natural extension to STL is parametric STL (pSTL). In pSTL, the specifications are introduced as parametric templates [13]. Typically the templates are proposed by the user and the goal is to fit parameter values given a timed trace. The mapping from a timed trace to parameter values is a form of feature extraction on varying-length time series data. Feature extraction from time-series data, particularly ones of varying length, is challenging due to the different time scales, and difficulty in defining a similarity metric. Traditional methods such as dynamic time warping [19] and the closely related Skorokhod metric [20] are useful, but are inadequate in learning logical structure which can be crucial for defining similarity. After the feature extraction step, further analysis, such as clustering or regression, can be applied on the extracted feature space [10].

Let $\phi_\mathcal{P}$ be a specification template with parameters $\mathcal{P}$ (e.g., $\phi_\alpha = \Box\, f(s) < \alpha$). Given a timed trace s, we would like to find a valuation $\nu(\mathcal{P})$ such that $\phi_{\nu(\mathcal{P})}$ is the best description of s. This is equivalent to solving the optimization problem[2],

$$
\nu^*(\mathcal{P}) = \arg\min_{\mathcal{P}} \|\rho(s,t,\phi_\mathcal{P})\|_2. \tag{1}
$$

We want to find parameter values $\nu(\mathcal{P})$ such that $\rho(s,t,\phi_{v(\mathcal{P})}) = 0$. However, since calculating the robustness of an STL specification involves recursively applying max and min operations, solving (1) becomes a very non-smooth, non-linear and non-convex problem. Standard optimization techniques such as gradient descent and direct methods become ineffective because gradients are non-smooth, and the robustness is expensive to evaluate. There are alternate

[1]If the signal is not long enough, the values can be set to the previous value.

[2]Other objective functions may be used, such as the logarithmic barrier function, but the initialization must be a feasible point.

solutions but they rely on the formula being monotonic which is not always the case [10], or recursive [14] techniques which can be very expensive to compute and scale poorly.

III. Computation Graph Representation

A. Code Toolbox

Code for creating the STL computation graphs can be found here `https://github.com/StanfordASL/stlcg`. It uses PyTorch [21] to create the graphs. Further, this toolbox includes a graph visualizer to show the graph representation of the STL formula and how it depends on the inputs and parameters.

B. Constructing the Computation Graph $\mathcal{G}$

We construct a computation graph (directed acyclic graph) representation $\mathcal{G}$ of the robustness and robustness trace calculation for STL formulas. This is summarized in Algorithm 1. For temporal operators, by propagating backwards in time via recurrent computation graphs, this method can compute the robustness and robustness trace simultaneously. This is particularly effective for formulas involving nested temporal operators. We present a computation graph representation for the predicate $(f(s) < c)$, all the Boolean connectives (negation/not, conjunction/and, disjunction/or), and all the temporal operators (And, Eventually, Until and Then). We prove soundness of this transformation. First, we make the following definitions.

Definition 3 (Valid Robustness Trace): Let $\mathcal{C}$ denote a method for computing the robustness of an STL formula ϕ, and $\tau^{\mathcal{C}}(s,t,\phi)$ is the robustness trace of ϕ using method $\mathcal{C}$. Then we say $\tau^{\mathcal{C}}(s,t,\phi)$ is a valid robustness trace for ϕ if $\tau^{\mathcal{C}}(s,t,\phi) = \tau(s,t,\phi)$.

Definition 4 (Soundness): Let $\mathcal{C}$ denote a method for computing the robustness of an STL formula ϕ. We say $\mathcal{C}$ is sound if it produces a valid robustness trace for all timed traces s and for all ϕ.

Data: Timed trace s, STL formula ϕ
Result: $\rho^{\mathcal{G}}(s,t,\phi), \tau^{\mathcal{G}}(s,t,\phi)$
1 Reverse s;
2 Construct $\mathcal{G}$ by combining $\mathcal{G}_{\varphi_i}$ where $\varphi_i \in \mathcal{O}_\phi$ (Refer to Section III-B). The inputs of $\mathcal{G}_{\varphi_i}$ should only depend on outputs from $\mathcal{G}_{\varphi_j}$, $j < i$;
3 Run the graph;
4 Reverse the output to get $\tau^{\mathcal{G}}(s,t,\phi)$;
5 $\rho^{\mathcal{G}}(s,t,\phi)$ is the first element of $\tau^{\mathcal{G}}(s,t,\phi)$;

Algorithm 1: Computing robustness and robustness trace using the proposed computation graph method $\mathcal{G}$.

We denote the computation graph used to compute the robustness trace of an STL formula ϕ by $\mathcal{G}$. The graph $\mathcal{G}$ is made up of smaller computation graphs $\mathcal{G}_{\varphi_k}$ that take a robustness trace of the subformula as inputs, and outputs the robustness trace after applying φ_k.

Since $\mathcal{G}$ is a directed acyclic graph where each subgraph is a "node", it has a topological ordering. This topological ordering of $\mathcal{G}$ given ϕ is precisely governed by $\mathcal{O}_\phi$, the post-order traversal of the parse tree generated by ϕ.

Example 2: Define a STL formula $\theta = \Diamond\Box(\phi \wedge \psi)$, then $\mathcal{O}_\theta = \{\phi, \psi, \wedge, \Box, \Diamond\}$. $\mathcal{PT}_\theta$ and $\mathcal{G}_\theta$ are represented in Figure 1 and it can be seen that the topological ordering of $\mathcal{G}_\theta$ is $\{\mathcal{G}_\phi, \mathcal{G}_\psi, \mathcal{G}_\wedge, \mathcal{G}_\Box, \mathcal{G}_\Diamond\}$.

The graph representing the robustness at a particular time t_i, denoted by $\mathcal{G}_\varphi^{(i)}$, of predicates and Boolean connectives are illustrated in Figure 2a. Computing the robustness of these operators rely only on elementary operations, so constructing $\mathcal{G}_\varphi^{(i)}$ is straight-forward. To compute the robustness trace, i.e., construct the graph $\mathcal{G}_\varphi$, $\mathcal{G}_\varphi^{(i)}$ is repeated over the timed trace. Inspired by recurrent neural networks [22] and their ability to effectively process sequential data, we propose using a *recurrent* computation graph model to compute the robustness, and robustness trace of the $\Diamond$ (eventually) and $\Box$ (always) operators. This structure can be leveraged and extended to compute the $\mathcal{U}$ (Until) and $\mathcal{T}$ (Then) operator.

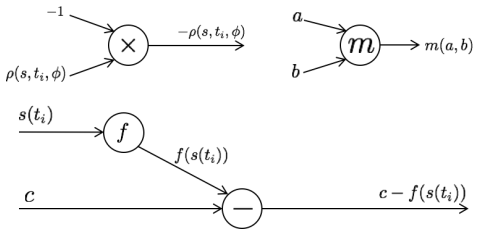
Suppose $\psi = \bigcirc_I \phi$ where $\bigcirc$ represents either $\Diamond$ (eventually) or $\Box$ (always). The graphical model for $\mathcal{G}_\bigcirc$ is depicted by Figure 2b. $\mathcal{G}_\bigcirc$ takes a robustness trace of length $N+1$ as an input, and outputs a robustness trace after applying the $\bigcirc$ operator.

With some abuse of notation, ρ_i represents $\rho(s,t_i,\phi)$, h_i is the hidden state, and o_i is the output at each cell i. Since I describes future times (i.e., $0 \le a < b$) from the given current time t, we need to propagate *backwards* in time in order for the computation graph to have knowledge about the values at $t \oplus I$. For ease of notation, we define the backwards sequence with $\tilde{\rho}$, i.e., $\rho_N = \tilde{\rho}_0$, ..., $\rho_i = \tilde{\rho}_{N-i}$, ...$\rho_0 = \tilde{\rho}_N$.

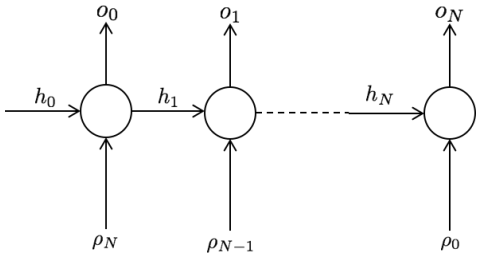
If $\bigcirc = \Diamond$, then $h_0 = -\rho_{\max}$ and $m = \max$ otherwise if $\bigcirc = \Box$, $h_0 = \rho_{\max}$ and $m = \min$. The computation graph for the m operator is given in Figure 2a, but adjusted to accept more inputs. For each computation node (cell), we apply the following update equations:

- Case 1: $I = [0, \infty)$
$$o_i = m(\tilde{\rho}_i, h_i), \qquad h_{i+1} = o_i$$
- Case 2: $I = [a, \infty)$, $a > 0$. We can truncate the start of s such that we obtain Case 1.
- Case 3: $I = [0, b]$, $b < \infty$. Let m be the number of samples from s that lie in $(0, b]$. Then $h_i \in \mathbb{R}^m$, $h_i = [h_{i_1}, h_{i_2}, ..., h_{i_m}]$, h_0 is a vector of $\pm\rho_{\max}$ instead of a scalar, and
$$o_i = m(\tilde{\rho}_i, h_{i_1}, ..., h_{i_m}), \qquad h_{i+1} = [h_{i_2}, ..h_{i_m}, \tilde{\rho}_i]$$
- Case 4: $I = [a, b]$, $b < \infty$. Let m be the number of samples from s that lie in $(0, b]$, and k be the number of samples from s that lie in I. Then $h_i \in \mathbb{R}^m$, $h_i = [h_{i_1}, h_{i_2}, ..., h_{i_m}]$, h_0 is a vector of $\pm\rho_{\max}$ instead of a scalar, and
$$o_i = m(h_{i_1}, ..., h_{i_k}), \quad h_{i+1} = [h_{i_2}, ..h_{i_m}, \tilde{\rho}_i]$$
Equivalently, we can truncate the start of s such that we obtain Case 3.

For both Case 3 and 4, h_i is a vector of all the robustness values at each time step $t' \in t \oplus I$ excluding the value at the current time for that cell i. Then the h_{i+1} update is simply shifting the vector in time by one step, removing the oldest



(a) Top left: $\mathcal{G}^{(i)}_{(\neg)}$, Top right: $\mathcal{G}^{(i)}_{(\vee)}$, or $\mathcal{G}^{(i)}_{(\wedge)}$, Bottom: $\mathcal{G}^{(i)}_{(f(s)<c)}$



(b) $\mathcal{G}_{\square}$, or $\mathcal{G}_{\Diamond}$

Fig. 2: Computation graph representation of the robustness function for the predicates, Boolean connective, and the Always and Eventually temporal operators.

value, and updating the newest values with the current value $\tilde{\rho}_i$.

For the Until and Then operators, notice that the second argument of the inner $\min$ operation is actually $\square_{[t,t']}\phi$ and $\Diamond_{[t,t']}\phi$ respectively. Thus we easily compute the robustness trace for each term in the robustness formula and take the appropriate $\max$ and $\min$ functions to obtain the robustness trace for the Until and Then operations.

The outputs of the temporal graph are precisely the elements of the robustness trace but in reverse. This reversal is accounted for in Algorithm 1.

Remark: Although Algorithm 1 outputs $\rho(s,t,\phi)$ and $\tau(s,t,\phi)$, since we are using a computation graph, the robustness and robustness trace of any subformula of ϕ is easily obtained as we propagate through the graph.

C. Soundness of $\mathcal{G}$

Given the details of $\mathcal{G}$ described in Section III-B, we make the following theorem.

Theorem 1 (Soundness): The sequence $\tau^{\mathcal{G}}(s,t,\phi)$ is equivalent to the robustness trace $\tau(s,t,\phi)$ for all s, t and ϕ.

Proof: We prove by structural induction on ϕ.

Base case: We need to show that $\tau^{\mathcal{G}}(s,t,f(s(t))<c)$ is the same as $\tau(s,t,f(s(t))<c)$. By construction (see Figure 2a), $\rho^{\mathcal{G}}(s,t,f(s(t))<c) = c - f(s(t)) = \rho(s,t,f(s(t))<c)$. Thus $\tau^{\mathcal{G}}(s,t,f(s(t))<c) = \tau(s,t,f(s(t))<c)$.

Induction step: Assume that $\tau^{\mathcal{G}}(s,t,\phi)$ and $\tau^{\mathcal{G}}(s,t,\psi)$ are valid robustness traces for ϕ and ψ respectively. This means we are assuming $\rho^{\mathcal{G}}(s,t_i,\phi) = \rho(s,t_i,\phi)$ and $\rho^{\mathcal{G}}(s,t_i,\psi) = \rho(s,t_i,\psi)$. We need to show that $\tau^{\mathcal{G}}(s,t,\phi\wedge\psi)$, $\tau^{\mathcal{G}}(s,t,\phi\vee\psi)$, $\tau^{\mathcal{G}}(s,t,\neg\phi)$, $\tau_{\mathcal{G}}(s,t,\Diamond\phi)$, $\tau^{\mathcal{G}}(s,t,\square\phi)$, $\tau^{\mathcal{G}}(s,t,\phi\mathcal{U}\psi)$, and $\tau^{\mathcal{G}}(s,t,\phi\mathcal{T}\psi)$ are all valid robustness traces. Showing the cases for $\wedge$, $\vee$ and $\neg$ follows naturally from the construction of their computation graphs. For clarity, only the proof for $\wedge$ is given:

$$\begin{aligned}\rho^{\mathcal{G}}(s,t_i,\phi\wedge\psi) &= \min(\rho^{\mathcal{G}}(s,t_i,\phi),\rho^{\mathcal{G}}(s,t_i,\psi))\\ &= \min(\rho(s,t_i,\phi),\rho(s,t_i,\psi))\\ &= \rho(s,t_i,\phi\wedge\psi)\end{aligned}$$

For $\bigcirc = \Diamond$ or $\square$, by construction, $\rho^{\mathcal{G}}(s,t_i,\bigcirc\phi) = o_{N-i}$ since the timed trace is fed backwards in time through the network, o_i depends on information in the future only, i.e., it is computed using the subsequence at t_{N-i}. Again, by construction on o_i and h_i, for all the different cases for I described in Section III-B,

$$\begin{aligned}o_{N-i} &= \min_{k\in t_i\oplus I}\rho(s,t_k,\phi)\\ &= \rho(s,t_i,\bigcirc_I\phi)\\ \therefore \rho^{\mathcal{G}}(s,t_i,\bigcirc_I\phi) &= \rho(s,t_i,\bigcirc_I\phi).\end{aligned}$$

To prove for $\mathcal{U}$ (and it follows easily to $\mathcal{T}$), first define (s,t_{-i}) as a sequence from t_0 to t_i. By construction and since we have proven that the other operators all have valid robustness traces:

$$\begin{aligned}\rho^{\mathcal{G}}(s,t,\phi\mathcal{U}_I\psi) &= \max_{i\in t\oplus I}(\min(\rho^{\mathcal{G}}(s,t_i,\psi),\min_{i'\in[0,i]}\rho^{\mathcal{G}}(s,t_{i'},\phi)))\\ &= \max_{i\in t\oplus I}(\min(\rho^{\mathcal{G}}(s,t_i,\psi),\rho^{\mathcal{G}}(s,t_{-i},\square\phi)))\\ &= \max_{i\in t\oplus I}(\rho^{\mathcal{G}}(s,t,(\rho^{\mathcal{G}}(s,t_i,\psi)\wedge\rho^{\mathcal{G}}(s,t_{-i},\square\phi))))\\ &= \max_{i\in t\oplus I}(\min(\rho(s,t_i,\psi),\min_{i'\in[0,i]}\rho(s,t_{i'},\phi)))\\ &= \rho(s,t,\phi\mathcal{U}_I\psi)\end{aligned}$$

Note that by using $\tau^{\mathcal{G}}(s,t_{-i},\square\phi)$ instead of $\rho^{\mathcal{G}}(s,t_{-i},\square\phi)$, we get the robustness trace (up to time t_i) for $\rho^{\mathcal{G}}(s,t,\phi\mathcal{U}_I\psi)$. ■

Since we have shown soundness for all the STL operators, we also have completeness. We have shown that $\mathcal{G}$ can be used to compute a valid robustness trace for any formula. In particular, we can see that the computation complexity of $\mathcal{G}$ scales linearly with the length of the signal, or quadratically for the Until and Then operations, and also linearly as we apply more temporal operations (i.e., tree depth of $\mathcal{PT}$).

IV. Backpropagation on STL Robustness Formulas

Section III detailed how a computation graph can be used to compute the robustness of a STL formula exactly. However, taking gradients with respect to pSTL parameters is difficult due to the the non-smooth nature of the $\max$ and $\min$ functions. We will show that by making a smooth approximation, we can use a general computation graph library to take the gradient with respect to a parameter using backpropagation.

Algorithm 2 shows how $\mathcal{G}$ can be used to solve for pSTL parameters. This is a typical gradient descent algorithm; the termination criteria depends on the true value of ρ and the gradient, and the relaxation parameter w is annealed over each iteration so that towards the end of optimization process, the approximation of the $\max$ and $\min$ functions more exact.

Data: Timed trace s, pSTL formula $\phi_{\mathcal{P}}$, maximum number of iterations N, robustness tolerance ρ_{tol}, gradient tolerance g_{tol}, annealing function A, learning rate γ.
Result: $\nu(\mathcal{P})$
Initialize $\nu(\mathcal{P})$;
Construct $\mathcal{G}$ (See Section III-B);
for $i \leftarrow 0$ **to** N **do**
  Compute $\rho_e = \rho(s, t, \phi_{\nu(\mathcal{P})})^2$;
  if $\rho_e < \rho_{tol}$ **then**
    break ;
  else
    $w_i \leftarrow A(i)$. Get the scaling parameter from the annealing function;
    Compute the loss $\ell = \rho(s, t, \phi_{\nu(\mathcal{P})}; w_i)^2$;
    Use backpropagation on ℓ to get $\frac{\partial \ell}{\partial \mathcal{P}}$;
    if $\|\frac{\partial \ell}{\partial \mathcal{P}}\|^2 < g_{tol}$ **then**
      break ;
    end
    Update $\nu(\mathcal{P})$ using a step size of γ;
  end
end

Algorithm 2: Algorithm for solving (1) using $\mathcal{G}$ and the differentiable approximations. We take advantage of the in-built autodifferentiation functionality in many ML tookboxes to backpropagte on the computation graph.

Depending on the application and assumptions, absolute exactness of the parameters is not strictly necessary. For example, when pSTL is used for feature extraction, rather than for formal verification and/or model checking, post-analysis methods such as clustering or regression have inherent noise assumptions which can be accounted for by the inexactness. Thus we propose using differentiable approximations of the $\min$ and $\max$ operators in order to make solving (1) more tractable especially when the formula has multiple temporal operators. This differentiable approximation, when coupled with the computation graph representation $\mathcal{G}$ offers the following benefits:

1) The $\mathcal{G}$ representation admits the use of backpropagation to compute gradients even for highly complex pSTL formulas.
2) A smooth differentiable approximation can improve convergence for more complicated pSTL formulas.
3) We can leverage the benefits of PyTorch or Tensorflow (i.e., state-of-the-art modern machine learning software tools). For example, we can directly use their optimization toolkits, easily implement on parallel hardware, and evaluate multiple signals via batching.

Since the proposed approach operates on computation graphs, it can potentially be used in conjunction with existing deep learning frameworks which are generally acting as a "black box", such as for intent prediction over long time horizons since they both operate on computation graphs and rely on backpropagation for optimization.

The proposed method can only optimize spatial parameters, but current work is on extending this to include temporal parameters (i.e., it does not apply to parameters that define the interval I). Optimizing over temporal operators is non-trivial because it is discrete and has a finite domain. However, inspiration can be taken from existing LSTM [22] networks that use the idea of "forget" gates that extract information from relevant times.

A. Min/Max Approximations

Let $\mathbf{x} \in \mathbb{R}^n$ and $w \in \mathbb{R}_{\geq 0}$, then the $\max$ and $\min$ approximations are

$$\widetilde{\max}(\mathbf{x}; w) = \frac{\sum_i^n x_i \exp(w x_i)}{\sum_j^n exp(w x_j)} \tag{2}$$

$$\widetilde{\min}(\mathbf{x}; w) = \frac{\sum_i^n x_i \exp(-w x_i)}{\sum_j^n \exp(-w x_j)}. \tag{3}$$

w is a scaling parameter, and when $w \to \infty$, this approximation approaches the true maximum or minimum, while $w = 0$ gives the average of the values in $\mathbf{x}$. This approximation becomes ill-defined when $w \to \infty$ [23]. However, we restrict w from getting too large (this parameter can be annealed) and ensure that $\mathbf{x}$ is scaled appropriately. The logsumexp max/min approximation may also be used, as also done in [9], [15].

For every $\max$ and $\min$ operation in the robustness formula, we replace it by $\widetilde{\max}$ and $\widetilde{\min}$. (2) and (3) uses elementary operations, so it can be directly converted into a computation graph. We write $\rho(s, t, \phi; w)$ to indicate that the robustness is computed using the approximation parametrized by w.

V. Case Study: Systematic Scenario Generation for Autonomous Driving

In order to certify the safety of any cyber-physical system, it must be strenuously stress-tested in a diverse range of scenarios to ensure the system is certifiably (failure occurs within some ϵ-probability) safe and robust. For instance, autonomous cars must be tested for millions of miles on diverse scenarios. Even with simulation, this is impractical and hinders the development of the technology. For this reason, when developing a test suite it is important to understand specifically what factors make a testing scenario easy or difficult to pass.

It is possible to generate test cases from simulation or real-world test data where a failure occurred, but it would be even more valuable to extrapolate to create unseen scenarios that may stress the system in different ways. However, given a large dataset, it can be difficult to find structure and explainability in the data. For instance, it is difficult in general to cluster time-series data, but more so to integrate semantics with unsupervised learning. In general, a designer may have a qualitative understanding of which factors are important, but may not have a strong quantitative understanding of how exactly those factors interact with each other to produce interesting test cases, nor is able to extract the relevant information from large datasets.

Fig. 3: The traffic-weaving scenario in SUMO. The red cars are merging onto the highway, the green cars are exiting, and the blue cars stay on the highway.

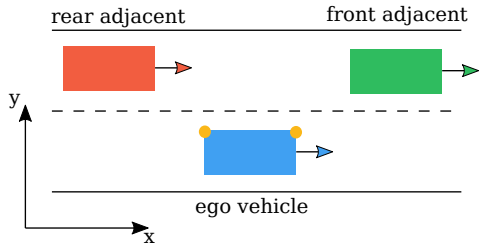


Fig. 4: Definition of front adjacent and rear adjacent vehicle for a given ego vehicle. The adjacent side is defined as the side that the ego vehicle is attempting to change into. The ego vehicle is trying to make a left lane change in this figure.

For our case study, we consider the problem of generating test cases for an autonomous vehicle in a traffic weaving setting, a challenging task where two cars must swap lanes in a short amount of time, emulating a highway on/off-ramp. We aim to use pSTL formulas to identify factors that contribute to making traffic-weaving challenging, and useful in designing challenging driving scenarios.

We begin by using human intuition to identify some potential quantities that may be important for the traffic-weaving task. We believe that "time to collision" (ttc), defined in (4), is an important quantity to monitor while executing a lane change (along with many others).

$$ttc(x_1, x_2, v_1, v_2) = -\frac{x_1 - x_2}{v_1 - v_2} \tag{4}$$

Here, x_i and v_i represents the longitudinal distance and velocity of a car i, assuming both cars are traveling almost parallel. Although human intuition has proposed this as an interesting quantity to monitor, we would like to concretely quantify how ttc actually affects the difficulty of the scenario. As such, we propose the following pSTL formula that capture the ttc value during a lane-change,

$$\phi = in\_start\_lane \, \mathcal{U} \, ttc < \alpha \tag{5}$$

in_start_lane is a formula that describes if the vehicle is still in its starting lane. Specifically, if the vehicle starts in the right lane, $in\_start\_lane = y < y_{line}$, and correspondingly, if the vehicle starts in the left lane, $in\_start\_lane = y > y_{line}$. ϕ can be interpreted as, when the car is no longer in its start lane, the vehicle has $ttc < \alpha$.

We generated about 2500 traffic-weaving trials using SUMO [24], an open source package for the Simulation of Urban MObility. We use SUMO's in-built car-following [25] and lane-changing [26] model. A snapshot of the traffic-weaving simulation is shown in Figure 3. We applied the Equation 5 to the front adjacent and rear adjacent cars of the ego vehicle (see Figure 4) and applied Algorithm 2 to solve for α for each trial. The resulting values, for both front and rear adjacent vehicles are shown in Figures 5a and 5b.

For the front adjacent case (Figure 5a), we see that the distribution is left-skewed, indicating that when a vehicle changes lane, the car ahead of it is typically traveling faster. This agrees with how humans naturally drive—it is uncommon for one to change lanes to be behind a slower moving vehicle on a highway. For the rear adjacent case (Figure 5b), the distribution is bimodal with a dip around $ttc = 0$. Again, this conforms with the intuition that drivers will change lanes when the oncoming car is either traveling slower or that it is sufficiently far away but potentially traveling faster.

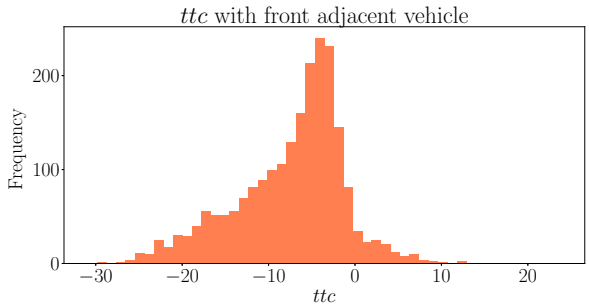
The same analysis is applied to another dataset that was collected from 1105 human-human traffic-weaving trials [27]. Here, only two cars are on the road, where they were initialized with $ttc \in \{1, 2, 3\}$ to ensure the resulting leading car was ambiguous. The corresponding results are shown in Figure 5c and 5d. We see that for the front adjacent case, the results are similar to the ones using the SUMO dataset. For the rear adjacent case, it does not have the $\alpha > 0$ mode due to the nature of how the cars are initialized. However, we see two modes at ≈ -2 and -6, suggesting that (1) drivers often tend to change lanes as soon as possible when the opportunity arises, and (2) drivers also often change lanes a little later when longitudinal gap is significantly larger.

These parameter bounds can be leveraged to design other cars on the road to be more adversarial and hence "stress-test" the ego-car's driving policy and accelerate the evaluation of an autonomous car. For example, one can design a car that purposely drives to keep ttc within some range. This can also be easily extended to other driving scenarios, such as making an unprotected left turn into flowing traffic.

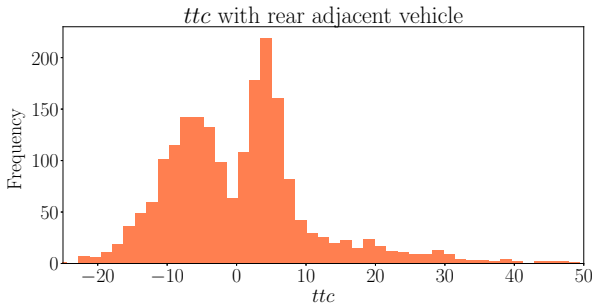
VI. Future Work and Conclusions

In future work, we would like to (1) extend this framework to optimize over temporal parameters, and (2) explore how we can directly embed logical formulas within deep learning architectures now that they both can be computed using the same tools. We believe that a closer link between machine learning tools and logic optimization tools may provide valuable insights to improving the interpretability of AI models.

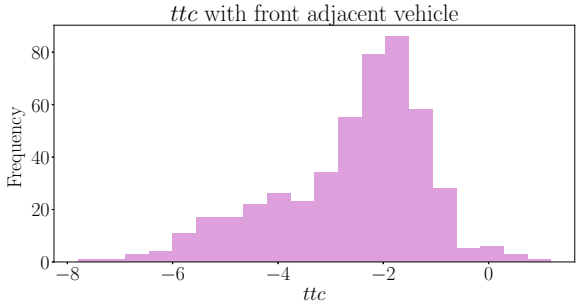
We have developed a framework that allows parameters of logical formulas to be trained with off-the-shelf machine learning tools. By translating STL robustness formulas into computation graphs and utilizing smooth approximations, we can easily optimize pSTL formulas via backpropgation. Further, this enables STLs and deep learned models to be used and/trained in conjunction with one another. A simple autonomous driving case study was carried out to demonstrate how STL formulas, implemented under the proposed framework, can provide actionable insight on how to improve the validation and testing methodologies of safety-critical systems.



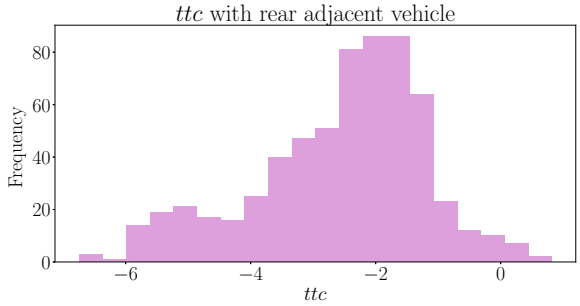
(a) Front adjacent vehicle using the SUMO dataset.



(b) Rear adjacent vehicle using the SUMO dataset.



(c) Front adjacent vehicle using the traffic-weaving dataset.



(d) Rear adjacent vehicle using the traffic-weaving dataset.

Fig. 5: Values for α in Equation 5 using the SUMO and traffic-weaving dataset.

REFERENCES

[1] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, no. 3253, pp. 152–166, 2004.

[2] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Nickovic, and S. Sankaranarayanan, "Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications," *The Handbook of Runtime Verification*, 2018.

[3] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *IEEE Conference on Decision and Control*, 2014.

[4] N. Mehr, D. Sadigh, R. Horowitz, S. Sastry, and S. A. Seshia, "Stochastic predictive freeway ramp metering from signal temporal logic specifications," in *American Control Conference*, 2017.

[5] D. Sadigh and A. Kapoor, "Safe control under uncertainty with probabilistic signal temporal logic," in *Robotics: Science and Systems*, 2017.

[6] S. P. Chinchali, S. C. Livingston, M. Chen, and M. Pavone, "Multi-objective optimal control for proactive decision-making with temporal logic models," *International Journal of Robotics Research*, vol. submitted, 2018.

[7] X. Li, C. I. Vasile, and C. Belta, "Reinforcement learning with temporal logic rewards," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[8] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," 2014.

[9] X. Li, Y. Ma, and C. Belta, "A policy search method for temporal logic specified reinforcement learning tasks," 2018.

[10] M. Vazquez-Chanlatte, J. V. Deshmukh, X. Jin, and S. A. Seshia, "Logical clustering and learning for time-series data," *Computer Aided Verification* , vol. 10426, pp. 305–325, 2017.

[11] G. Bombara, C. I. Vasile, F. Penedo, H. Yasuoka, and C. Belta, "A decision tree approach to data classification using signal temporal logic," 2016.

[12] Z. Kong, A. Jones, A. M. Ayala, E. A. Gol, and C. Belta, "Temporal logic inference for classification and prediction from data," 2014.

[13] E. Asarin, A. Donzé, O. Maler, and D. Nickovic, "Parametric identification of temporal properties," in *International Conference on Runtime Verification*, 2012.

[14] A. Bakhirkin, T. Ferrère, and O. Maler, "Efficient parametric identification for STL," in *Hybrid Systems: Computation and Control*, 2018.

[15] Y. V. Pant, H. Abbas, and R. Mangharam, "Smooth operator: Control using the smooth robustness of temporal logic," 2017.

[16] L. Liebenwein, W. Schwarting, C. I. Vasile, J. DeCastro, J. Alonso-Mora, S. Karaman, and D. Rus, "Compositional and contract-based verification for autonomous driving on road networks," in *International Symposium on Robotics Research*, 2017.

[17] R. Balestriero, "Neural decision trees," *ArXiv e-prints*, February 2017, 1702.07360.

[18] Y. Yang, I. Garcia Morillo, and T. M. Hospedales, "Deep neural decision trees," *ArXiv e-prints*, 2018, 1806.06988.

[19] E. J. Keogh and M. J. Pazzani, "Scaling up dynamic time warping for data-mining applications," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000.

[20] J. V. Deshmukh, R. Majumdar, and V. S. Prabhu, "Quantifying conformance using the skorokhod metric," *Formal Methods in System. Design*, vol. 50, no. 2-3, pp. 168–206, 2017.

[21] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," 2017.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] L. Teow and K. Loe, "An effective learning method for max-min neural networks," in *International Joint Conference on Artificial Intelligence*, 1997.

[24] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, December 2012.

[25] S. Krauß, P. Wagner, and C. Gawron, "Metastable states in a microscopic model of traffic flow," *Physical Review* , vol. 55, no. 304, pp. 5–97, 1997.

[26] J. Erdmann, "Lane-changing model in sumo," in *Proceedings of the SUMO2014 Modeling Mobility with Open Data*, 2014.

[27] E. Schmerling, K. Leung, W. Vollprecht, and M. Pavone, "Multimodal probabilistic model-based planning for human-robot interaction," in *Proc. IEEE Conf. on Robotics and Automation*, Brisbane, Australia, 2018.