Signal Temporal Logic meets Reachability: Connections and Applications

Mo Chen¹, Qizhan Tan², Scott C. Livingston³, and Marco Pavone²

¹ School of Computing Science, Simon Fraser University
mochen@cs.sfu.ca
² Department of Aeronautics and Astronautics, Stanford University
{qtan, pavone}@stanford.edu
³ rerobots, Inc.
scott@rerobots.net

Abstract. Signal temporal logic (STL) and reachability analysis are effective mathematical tools for formally analyzing the behavior of robotic systems. STL is a specification language that uses logic and temporal operators to precisely express real-valued and time-dependent requirements on system behaviors. While recursively defined STL specifications are extremely expressive and controller synthesis methods exist, there has not been work that quantifies the set of states from which STL formulas can be satisfied. Reachability analysis, on the other hand, involves computing the reachable set, that is the set of states from which a system is able to reach a goal while satisfying state and control constraints. While reasoning about system requirements through sets of states is useful for predetermining whether it is possible to satisfy desired system properties and obtaining state feedback controllers, so far the application of reachability has been limited to relatively simple reach-avoid specifications. In this paper, we merge STL and time-varying reachability into a single framework that combines the key advantage of both methods – expressiveness of specifications and set quantification. To do this, we establish a correspondence between temporal and reachability operators, and utilize the idea of least-restrictive feasible controller sets (LRFCSs) to break down controller synthesis for complex STL formulas into a sequence of reachability and elementary set operations. LRFCSs are crucial for avoiding controller conflicts among the different reachability operations. In addition, the synthesized state feedback controllers are guaranteed to satisfy STL specifications if determined to be possible by our framework, and violate specifications minimally if not. Hamilton-Jacobi reachability will be used in this paper for its simplicity, although our method is agnostic with respect to the time-varying reachability method. We demonstrate our method through numerical simulations and robotic experiments.

1 Introduction

In recent years, the number of safety-critical applications of robotics has grown quickly, and safety analysis can be considered a key bottleneck for the development and widespread deployment of autonomous systems. In safety-critical systems, formal guarantees on system behavior are needed under appropriate modeling assumptions such as actuation limits, external disturbances, and dynamic model. State-of-the-art classes of methods making such guarantees include robust planning [23, 33], reachability analysis [5, 10], and temporal logic-based model checking and synthesis [3, 6, 20].

In particular, Signal Temporal Logic (STL) has gained popularity recently [12, 14] due to a number of key advantages. For instance, it explicitly treats real-valued variables and dense-time requirements [26] both of which are essential in practical robotics. Also,

* This work was supported by the Office of Naval Research YIP program (Grant N00014-17-1-2483) and by DARPA under the Assured Autonomy program. M. Chen and Q. Tan contributed equally to this work.
In addition to the usual Boolean semantics, STL has quantitative semantics that provide robustness estimates of satisfaction by system trajectories [12, 14]. The robustness estimate, a real-valued function, indicates satisfaction of an STL formula in terms of zero superlevel sets, which allows the specifications to represent subsets of the continuous state space. Checking the satisfaction of a specification amounts to evaluating a state on such a function [12]. This function representation of specifications allows modern optimization-based techniques to be used for controller synthesis [30, 31].

Viewed from the perspective of reachability and other verification techniques, STL provides a rich language for precisely describing complex system requirements through recursive application of logical and temporal operators. However, without reasoning about behaviors of sets of states as is done in reachability analysis, synthesizing state feedback controllers, as well as efficiently quantifying whether it is possible for any given state to satisfy a specification is challenging, especially for general nonlinear systems.

Reachability analysis provides a complementary perspective in system verification. Here, one is concerned with computing the reachable set (RS), defined as the set of states from which a system, with a given dynamics model, can reach some target set of states while satisfying state and control constraints. Reachability has also been extensively studied in the past couple of decades, with a large variety of methods for computing RSs [13, 15, 16, 21, 28] and many application domains [9, 18, 25]. Reachable sets are sometimes represented by sub- or super-level sets of value functions, referred to as barrier functions in the nonlinear control systems literature [24, 34, 35], although the use of barrier functions is not strictly necessary in reachability analysis.

While our theory is compatible with any time-varying reachability method\footnote{Some time-invariant reachability methods can be made time-varying by augmenting state space with time, as long as the resulting dynamics are still compatible with the reachability method.}, we focus on Hamilton-Jacobi (HJ) reachability, which directly computes time-varying RSs without state augmentation. HJ reachability also provides globally optimal RSs and feedback controllers for general nonlinear systems through numerically solving an HJ variational inequality (VI) [27] for low-dimensional systems [5, 10, 16]. Viewed from the perspective of STL, reachability is concerned with just the reach, avoid, or reach-avoid operators. All these operators can be expressed as simple STL formulas. Crucially, the notion of recursively constructing complex specifications from simpler ones and the associated nuances in controller synthesis have not been explored in reachability.

**Related work:** Prior work includes controller synthesis algorithms based on approximate dynamic programming with co-safe LTL specifications [29]. Being a fragment of LTL, co-safe LTL does not represent real-valued signals, in contrast to STL as we treat in this paper. In other prior work, a finite-state machine representing task behavior is assumed to be given, and controller synthesis for a nonlinear system is performed to realize it [1]. Doing so avoids reasoning about the specification language (e.g., LTL, STL, GR(1)), which provides generality at the cost of conservative solutions.

**Contributions:** In this paper, we connect STL and HJ reachability to take advantage of the features of both methods. From the perspective of STL, our proposed method moves beyond current controller synthesis methods, and provides the set of states, represented by value functions, from which any STL formula can be satisfied. From the perspective of HJ reachability, our method looks beyond single reachability problems, and takes advantage of the expressiveness of STL to define sequences of reachability problems, this is enabled by the least-restrictive feasible controller set (LRFCS), which is the set of controllers that guarantees the satisfaction of an STL formula, and the key concept that both provides a new controller synthesis method for STL formulas,
and allows reachability to take advantage of the richness of specification description in STL. We also interpret value functions in the context of minimum violation, a particularly useful concept in many real-world scenarios such as autonomous driving [32]. We demonstrate our method in numerical simulations and robotic experiments in representative autonomous driving scenarios. A simple illustrative summary of the main features of our framework is shown in Fig. 1. Collectively, the results of this paper provide a single framework that quantifies the set of states from which STL formulas can be satisfied and breaks down controller synthesis of complex STL formulas into a sequence of reachability computations in using existing numerical tools.

Organization: In Sec. 2, we introduce notation, background material on STL and HJ reachability, and problem statement. In Sec. 3, we establish the equivalence among STL formulas, value functions, and reachability operators. Through reachability, we quantify the set of states from which an STL formula can be satisfied. In Sec. 4, we discuss controller synthesis, with emphasis on avoiding control conflicts in complex STL formulas. Sec. 5 summarizes the results of this paper, and allows the reader to compute the set of states satisfying any STL formula, as well as feedback controller synthesis. We present our numerical and robotic experiments on two representative autonomous driving scenarios in Sec. 6. Finally, we conclude and suggest future work in Sec. 7.

2 Preliminaries

2.1 System Dynamics and Trajectories

Consider a dynamical system with an ordinary differential equation (ODE) model:

\[ \frac{dx}{ds} := \dot{x} = f(x, u, d), \quad u \in \mathcal{U}, d \in \mathcal{D}, \]  

(1)

where \( x \in \mathcal{X} \) is the system state, \( u \) the control restricted to a compact set \( \mathcal{U} \), and \( d \) the disturbance restricted to a compact set \( \mathcal{D} \). We assume the control function \( u(\cdot) \) is measurable, and if for all \( t, u(t) \in \mathcal{U} \), we also write \( u(\cdot) \in \mathcal{U} \), where \( \mathcal{U} \) is the function space containing all admissible controls. In the analogous fashion, we also write \( d(\cdot) \in \mathcal{D} \).

The system dynamics \( f \) are assumed to be uniformly continuous, bounded, and Lipschitz continuous in \( x \) for fixed \( u \) and \( d \); given a measurable control and disturbance functions \( u(\cdot) \) and \( d(\cdot) \), there exists a unique trajectory solving (1) [11]. The system in (1) can model not only a single robot, but also the joint dynamics of multiple robots. Besides modeling unknown, bounded disturbances, \( d \) is also very useful for representing the control action of a different robot, as shown in our experiments in Sec. 6.
We denote trajectories satisfying (1) starting from state $x$ at time $t$ under control $u(\cdot)$ and disturbance $d(\cdot)$ as $\xi^{u(\cdot),d(\cdot)}_{x,t}(\cdot)$. The trajectory $\xi$ satisfies (1) with an initial condition almost everywhere: $\xi^{u(\cdot),d(\cdot)}_{x,t}(s) = f(s, \xi^{u(\cdot),d(\cdot)}_{x,t}(s), u(s), d(s))$, $\xi^{u(\cdot),d(\cdot)}_{x,t}(t) = x$.

**Remark 1.** We make explicit the dependence of the system trajectory $\xi^{u(\cdot),d(\cdot)}_{x,t}(\cdot)$ on the control and disturbance so that the presentation of the material in the rest of the paper can be made clearer. Under this slightly more cumbersome notation, the state at any given time $s$ is written as $\xi^{u(\cdot),d(\cdot)}_{x,t}(s)$, instead of the usual $x(s)$. In addition, we use $t$ to denote the initial time, and $s \geq t$ to denote a time instant at or after $t$.

### 2.2 Signal Temporal Logic

We focus in this paper on STL, a specification language that expresses requirements directly for real-valued and dense-time signals [26]. STL admits a notion of robustness [12, 14], and recent work has used it for analyzing performance and safety properties of robotic systems. A brief overview of STL is given in this section, mostly following the development in [12]. An introduction to basic concepts can be found in [3].

The syntax of STL formulas is defined recursively by the grammar production

$$
\varphi ::= \text{True} \mid \mu(\cdot) \geq 0 \mid \neg \varphi \mid \varphi \lor \psi \mid \varphi U_{[s,\infty)} \psi,
$$

where $I$ is a closed interval of $\mathbb{R}$ of the form $[s_1, s_2]$ or $[s_1, \infty)$ with $0 \leq s_1 < s_2$, and where $\mu : \mathcal{X} \rightarrow \mathbb{R}$ is a function that maps states to real values. In this paper, we sometimes assume that the STL formula is in Negation Normal Form, i.e., negation ($\neg$) can only operate on predicates $\mu(\cdot) \geq 0$. Note that this is done without loss of generality because every STL formula has an equivalent formula in Negation Normal Form, which can be constructively obtained by the syntactic translation rules of Def. 11 in [14].

In temporal logics that only have Boolean semantics, e.g., linear temporal logic (LTL), the syntax has atomic propositions instead of real-valued functions ($\mu$ in the grammar above). Thus, to apply LTL formulas to the dynamical system (1), some relation between atomic propositions and states must be defined to decide whether a state trajectory satisfies a formula. In contrast, STL formulas can include predicates $\mu(\cdot) \geq 0$ that directly express the relation between states and Boolean semantics. For example, a convex polytope, defined by a set of linear inequalities of the form $Hx \leq K$, for $H \in \mathbb{R}^{m \times n}$, $K \in \mathbb{R}^{m}$, can be equivalently encoded by the STL formula $\mu_1(x) \geq 0 \land \cdots \land \mu_m(x) \geq 0$, where $\mu_i(x) = k_i - h_i x$ for $i = 1, \ldots, m$, and $h_i$ is the $i$-th row of $H$.

STL is defined with two kinds of semantics. First, the **Boolean semantics** is defined as follows. Let $\xi^{u(\cdot),d(\cdot)}_{x,t}(\cdot)$ be the trajectory of (1) starting from state $x$ at time $t$ under control $u(\cdot)$ and disturbance $d(\cdot)$, as defined in the previous section. Then, $\xi^{u(\cdot),d(\cdot)}_{x,t}(\cdot)$ at time $s$ satisfies an STL formula $\varphi$ according to the inductive definition:

1. $(\xi^{u(\cdot),d(\cdot)}_{x,t}(\cdot), s) \models \mu(\cdot) \geq 0$ if and only if $\mu(\xi^{u(\cdot),d(\cdot)}_{x,t}(s)) \geq 0$, \hspace{1cm} (3a)
2. $(\xi^{u(\cdot),d(\cdot)}_{x,t}(\cdot), s) \models \varphi \land \psi$ if and only if $\xi^{u(\cdot),d(\cdot)}_{x,t}(s) \models \varphi$ and $\xi^{u(\cdot),d(\cdot)}_{x,t}(s) \models \psi$, \hspace{1cm} (3b)
3. $(\xi^{u(\cdot),d(\cdot)}_{x,t}(\cdot), s) \models \neg \varphi$ if and only if $\xi^{u(\cdot),d(\cdot)}_{x,t}(s) \not\models \varphi$,
4. $(\xi^{u(\cdot),d(\cdot)}_{x,t}(\cdot), s) \models \varphi U_{[s_1,s_2]} \psi$ if and only if there exists $s' \in [s + s_1, s + s_2]$ such that $\xi^{u(\cdot),d(\cdot)}_{x,t}(s') \models \psi$, \hspace{1cm} (3c)
From the syntax and Boolean semantics, we can derive other commonly used operators, e.g., \( \neg \varphi \lor \neg \psi \equiv \neg(\varphi \land \psi) \) and, of particular interest:

- the operator “eventually” is denoted \( \Diamond_{[s_1,s_2]} \varphi \) and defined by \( \text{True}_{[s_1,s_2]} \varphi \); and
- the operator “always” is denoted \( \square_{[s_1,s_2]} \varphi \) and defined by \( \neg \Diamond_{[s_1,s_2]} \neg \varphi \).

Perhaps the most practically useful aspect of STL is its quantitative semantics, represented by a function \( g \) that is defined inductively as follows:

\[
g(\mu(\cdot) \geq 0, s, \xi^{u(\cdot), d(\cdot)}_{x,t}(\cdot)) = \mu(\xi^{u(\cdot), d(\cdot)}_{x,t}(s)), \tag{4a}
\]

\[
g(\neg \varphi, s, \xi^{u(\cdot), d(\cdot)}_{x,t}(\cdot)) = -g(\varphi, s, \xi^{u(\cdot), d(\cdot)}_{x,t}(\cdot)), \tag{4b}
\]

\[
g(\varphi \land \psi, s, \xi^{u(\cdot), d(\cdot)}_{x,t}(\cdot)) = \min\{g(\varphi, s, \xi^{u(\cdot), d(\cdot)}_{x,t}(\cdot)), g(\psi, s, \xi^{u(\cdot), d(\cdot)}_{x,t}(\cdot))\}, \tag{4c}
\]

\[
g(\varphi U_{[s_1,s_2]} \psi, s, \xi^{u(\cdot), d(\cdot)}_{x,t}(\cdot)) = \sup_{s' \in [s_1, s_2]} \min\{g(\psi, s', \xi^{u(\cdot), d(\cdot)}_{x,t}(\cdot))\}. \tag{4d}
\]

As shown in [12], the function \( g \) has the property that its sign implies satisfaction: if \( g(\psi, s, \xi^{u(\cdot), d(\cdot)}_{x,t}(\cdot)) \) is positive, then the trajectory satisfies STL formula \( \varphi \) from time \( s \). Similarly, it does not satisfy \( \varphi \) if \( g \) is negative at time \( s \). Because of its other fundamental property (correctness), the authors of [12] also refer to \( g \) as the robustness estimate.

As with the Boolean semantics, additional operators like \( \square \) (“always”) can be derived for the quantitative semantics [12]. Finally, if \( s = 0 \), then the notation can be abbreviated by omitting \( s \). In this paper, the notation for the quantitative semantics is made more concise by writing the STL formula (i.e., the first parameter of \( g \)) as a subscript; e.g., \( g_{\neg \varphi} \) is the robustness estimate for the STL formula \( \neg \varphi \). This way, we can concisely refer to multiple STL formulas and their corresponding robustness estimates of a trajectory.

We will use \( g(\cdot) \) to denote functions whose zero \emph{superlevel} sets represent satisfaction of an STL formula, and \( h(\cdot) \) to denote functions whose zero \emph{sublevel} sets represent sets used in HJ reachability as in (7) and (9). Also, whenever a function \( g_{\varphi} \) or its negation satisfies the HJ variational inequality (VI) (8) or (11), introduced in Sec. 2.3, we will refer to \( g_{\varphi} \) as a “\emph{value function associated with the STL formula} \( \varphi \)”.

### 2.3 Time-Varying Reachability

To draw a connection between temporal logic and reachability, one must consider time-varying formulations of reachability, to capture the temporal aspects of logical specifications. In general, many (time-invariant) reachability methods can be used for solving time-varying reachability problems by augmenting the state space with time.

In this paper, we focus on HJ reachability [16], which is the most general method in terms of system dynamics and set representation, and incorporates time without state augmentation. Note the following parallel between STL and HJ reachability: reachable sets are solutions to a “game of kind” in which one is interested in determining whether or not the target set can be reached; computation is done by solving a “game of degree” in which one \emph{minimizes a cost function} representing the target sets and constraints; this is analogous to the Boolean semantics and quantitative semantics, respectively, of STL described in Sec. 2.2. Given target and constraint sets \( T(s), C(s) \) as a function of time \( s \), the maximal reachable set (RS) \( R\mathcal{R}A^M(t; T; \cdot, C; \cdot) \) is defined as follows:

\[
R\mathcal{R}A^M(t; T; \cdot, C; \cdot) = \{ x : \exists u(\cdot) \in \mathcal{U}, d(\cdot) \in \mathcal{D}, \exists s \in [t, T], \xi^{u(\cdot), d(\cdot)}_{x,t}(s) \in T(s) \}. \tag{5}
\]

Informally, \( R\mathcal{R}A^M(t; T; \cdot, C; \cdot) \) is the set of states from which there exists a control function \( u(\cdot) \) that, despite the worst disturbance functions\(^3\) \( d(\cdot) \), drives the system to the

\(^3\) With a slight abuse of notation, we use \( \mathcal{D} \) to denote non-anticipative disturbance functions, intuitively control policies that do not depend on the future actions of another agent. See [28].
target set at some time $s \in [t,T]$ while satisfying constraints prior to reaching the target. In the HJ convention, a set $\mathcal{S}$ is represented by the zero sublevel set of some function: $\mathcal{S} = \{ x : h_S(x) < 0 \}$. Such a function always exists [16]. Given $h_T(t,x)$ and $h_C(x)$, define

$$ h_{R,A,M}(t,x) = \inf_{u(\cdot) \in \mathcal{U}} \sup_{d(\cdot) \in \mathcal{D}} \min_{\tau \in [t,T]} \max_{s \in [t,T]} \left\{ h_T(t,x_{s,t}^{u(\cdot),d(\cdot)}(s)), \max_{\tau \in [t,T]} h_C(s_{x,t}^{u(\cdot),d(\cdot)}(s)) \right\}. \tag{6} $$

and we have that the zero sublevel set of $h_{R,A,M}(t,x)$ represents the maximal RS:

$$ \mathcal{R}_{A,M}(t,T; \mathcal{T}(\cdot),C(\cdot)) = \{ x : h_{R,A,M}(t,x) < 0 \}. \tag{7} $$

The value function $h_{R,A,M}(t,x)$ can be computed by solving the following HJ VI:

$$ \max \left\{ \frac{\partial}{\partial t} h_{R,A,M}(t,x) + \min_{u(\cdot) \in \mathcal{U}} \sup_{d(\cdot) \in \mathcal{D}} \nabla h_{R,A,M}(t,x) \cdot f(t,x,u,d), \ h_T(t,x) - h_{R,A,M}(t,x) \right\}, \tag{8} $$

$$ h_C(x) - h_{R,A,M}(t,x) = 0, \quad h_{R,A,M}(T,x) = \max \{ h_T(T,x), h_C(x) \}. \tag{9} $$

In addition, we also define the minimal RS as

$$ \mathcal{R}_M(t,T; \mathcal{T}(\cdot)) = \{ x : \forall u(\cdot) \in \mathcal{U}, \exists d(\cdot) \in \mathcal{D}, \exists s \in [t,T], x_{s,t}^{u(\cdot),d(\cdot)}(s) \in \mathcal{T}(s) \}. \tag{10} $$

Intuitively, the minimal RS $\mathcal{R}_M(t,T; \mathcal{T}(\cdot))$ is the set of states from which no matter what control function $u(\cdot)$ is applied, there exists a disturbance function $d(\cdot)$ that drives the system into the target set within some time horizon. Given $h_T(t,x)$, define

$$ h_{R,A}(t,x) = \sup_{u(\cdot) \in \mathcal{U}} \inf_{d(\cdot) \in \mathcal{D}} \min_{\tau \in [t,T]} \max_{s \in [t,T]} h_T(t,x_{s,t}^{u(\cdot),d(\cdot)}(s)). \tag{11} $$

and we have that the zero sublevel set of the solution $h_{R,A}(t,x)$ represents the minimal RS. The value function $h_{R,A}(t,x)$ can be computed by solving the following HJ VI:

$$ \min \left\{ \frac{\partial}{\partial t} h_{R,A}(t,x) + \max_{u(\cdot) \in \mathcal{U}} \sup_{d(\cdot) \in \mathcal{D}} \nabla h_{R,A}(t,x) \cdot f(t,x,u,d), h_T(t,x) - h_{R,A}(t,x) \right\}, \quad t \in [0,T], \tag{12} $$

$$ h_{R,A}(T,x) = h_T(T,x). \tag{13} $$

The main differences between (8) and (12) are a lack of the outer maximum in (12) due to a lack of constraints in the minimal RS, and reversed optimization over $u$ and $d$.

2.4 Problem Formulation

The goal of this paper is to achieve the following objectives within a single framework:

1. Establish a correspondence between STL operators and reachability operators. This is done by recognizing that logical operators in STL are equivalent to elementary set operations, and that STL temporal operators are equivalent to reachability operators.
2. Leverage HJ reachability in the context of STL to compute value functions that represent the set of states from which any STL formula can be satisfied.
3. Leverage the formalism of STL, along with our proposed notion of the LRFCS to perform controller synthesis through a sequence of reachability computations without introducing any controller conflicts.
4. Recognize the minimum violation interpretation of value functions.

We demonstrate our framework through simulations and experiments that are representative of autonomous driving scenarios.

3 STL Specifications in the HJI context

In this section, we go through the STL semantics and present how HJ reachability can be used to compute value functions that represent the set of states from which STL formulas can be satisfied. Sec. 3.1 describes the connection between logical operators and elementary set operations and provides a correspondence between function representations of specifications in the STL convention and of sets in HJ reachability. Sec. 3.2 describes the connection between temporal operators in STL – the until and always operators – and the maximal and minimal reachability operators in HJI reachability.
3.1 STL logical and elementary set operations, and functional representations

First given an atomic proposition $\mu$, we define a set $S_\mu := \{ x : g_\mu(x) > 0 \}$. In addition, given an STL formula, for example one denoted $\varphi$ or $\psi$, we define $S_\varphi$ and $S_\psi$ to denote the set of states that satisfy $\varphi$ or $\psi$, respectively.

\[
\begin{align*}
\langle s^{\mu}(\cdot), d^{\cdot}(\cdot), s \rangle \models \mu & \iff s^{\mu}(\cdot), d^{\cdot}(\cdot) \in S_\mu \\
\langle s^{\mu}(\cdot), d^{\cdot}(\cdot), s \rangle \models \varphi & \iff s^{\mu}(\cdot), d^{\cdot}(\cdot) \in S_\varphi \\
\langle s^{\mu}(\cdot), d^{\cdot}(\cdot), s \rangle \models \psi & \iff s^{\mu}(\cdot), d^{\cdot}(\cdot) \in S_\psi
\end{align*}
\]  

(12a) \hspace{2.5cm} (12b) \hspace{2.5cm} (12c)

Then, we have the following correspondence in terms of functions that represent satisfaction of STL formulas and functions that represent sets used in HJ reachability.

\[
g_\mu(x) > 0 \iff h_{S_\mu}(x) < 0, \quad g_\varphi(x) > 0 \iff h_{S_\varphi}(x) < 0, \quad g_\psi(x) > 0 \iff h_{S_\psi}(x) < 0, \quad (13)
\]

Based on the set definitions above, the logical conjunction, disjunction, and negation are equivalent to set intersection, union, and complement, respectively:

\[
\begin{align*}
\langle s^{\mu}(\cdot), d^{\cdot}(\cdot), s \rangle \models \varphi \land \psi & \iff s^{\mu}(\cdot), d^{\cdot}(\cdot) \in S_\varphi \cap S_\psi \\
\langle s^{\mu}(\cdot), d^{\cdot}(\cdot), s \rangle \models \varphi \lor \psi & \iff s^{\mu}(\cdot), d^{\cdot}(\cdot) \in S_\varphi \cup S_\psi \\
\langle s^{\mu}(\cdot), d^{\cdot}(\cdot), s \rangle \models \neg \varphi & \iff s^{\mu}(\cdot), d^{\cdot}(\cdot) \in S^C_\varphi
\end{align*}
\]  

(14a) \hspace{2.5cm} (14b) \hspace{2.5cm} (14c)

In terms of the functional representation of formulas and sets, we have

\[
\begin{align*}
g_{\varphi \land \psi}(x) & = \min\{g_\varphi(x), g_\psi(x)\}, \quad h_{\varphi \land \psi}(x) = \max\{h_{S_\varphi}(x), h_{S_\psi}(x)\} \\
g_{\varphi \lor \psi}(x) & = \max\{g_\varphi(x), g_\psi(x)\}, \quad h_{\varphi \lor \psi}(x) = \min\{h_{S_\varphi}(x), h_{S_\psi}(x)\} \\
g_{\neg \varphi}(x) & = -g_\varphi(x), \quad h_{\neg \varphi}(x) = -h_{S_\varphi}(x)
\end{align*}
\]  

(15a) \hspace{2.5cm} (15b) \hspace{2.5cm} (15c)

3.2 “Until” and “always” as reachability operators

We now look at the important connections between the until and always operators and reachability. The until operator, defined in Equation (3e), can be interpreted as a constrained reachability operator. For example, in $\varphi U_1 \psi$, one is interested in reaching states that satisfy $\psi$ within some time horizon, while satisfying the constraints $\varphi$. We now formally state this reachability interpretation.

Proposition 1 (The until operator and constrained reachability).

Define $T = t + s_2$ and the sets $T_\varphi(\cdot)$ and $C_\varphi$:

\[
T_\varphi(s) = \begin{cases} 
\{ x : (x(\cdot), s) \models \psi \}, & \text{if } s \in [t + s_1, t + s_2], \\
\emptyset, & \text{otherwise},
\end{cases} \quad C_\varphi = \{ x : (x(\cdot), s) \models \varphi \}. 
\]

(16)

Then, we have

\[
\exists u(\cdot) \in U, \forall d(\cdot) \in D, \langle \xi^{T_\varphi(\cdot), d^{\cdot}(\cdot)}, t \rangle \models \varphi U_{t, s_2} \psi \iff x \in T_\varphi U_{t, s_2} G(t, T; T_\varphi(\cdot), C_\varphi). 
\]

(17)

In addition, define $g_{\varphi U_{t, s_2} \psi}(t, x) = -h_{T_\varphi(t, x)}$, where $h_{T_\varphi(t, x)}$ is such that $T_\varphi(t, t + s_2; T_\varphi(\cdot), C_\varphi) = \{ x : h_{T_\varphi(t, x)} < 0 \}$. Then, we have

\[
\exists u(\cdot) \in U, \forall d(\cdot) \in D, \langle \xi^{T_\varphi(\cdot), d^{\cdot}(\cdot)}, t \rangle \models \varphi U_{t, s_2} \psi \iff g_{\varphi U_{t, s_2} \psi}(t, x) > 0.
\]

(18)

Note that $-g_{\varphi U_{t, s_2} \psi}(t, x)$ satisfies the HJI (8), so $g_{\varphi U_{t, s_2} \psi}$ is a value function associated with $\varphi U_{t, s_2} \psi$. 
Prop. 1 follows from the definition of the until operator and the maximal RS, and establishes an equivalence between the until operator and the maximal reachabilility operator. It also provides a single function \( k_\varphi U_{[s_1,s_2]}(s, x) \) that captures the set of states from which there exists a controller to guarantee the satisfaction of \( \varphi U_{[s_1,s_2]} \varphi \) regardless of disturbances. Controller synthesis will be discussed in Sec. 4 in Lem. 1.

Note that the eventually operator corresponds to an unconstrained reachability problem, which is the one presented in Prop. 1 with \( \varphi = \text{True} \), or equivalently, \( C_\varphi = \emptyset \).

The always operator can be indirectly interpreted as an unconstrained reachability operator. For example, in \( \Box_{[s_1,s_2]} \varphi \), one is interested in staying in states that satisfy \( \varphi \). In the language of reachability, one would equivalently stay out of states that may lead to a violation of \( \varphi \). We now formally state this reachability interpretation.

**Proposition 2** (Reachability interpretation of the always operator).

Define \( T = s + s_2 \), and

\[
T_\varphi(s) = \begin{cases} 
\{ x : (x(s), s) \not\in \varphi \}, & \text{if } s \in [s + s_1, s + s_2] \\
\emptyset, & \text{otherwise.}
\end{cases}
\]  

(19)

Then, we have

\[
\exists u(\cdot) \in U, \forall d(\cdot) \in D, \text{ if } (\xi_{s,s_1}^{u(\cdot),d(\cdot)}(\cdot), s) \models \Box_{[s_1,s_2]} \varphi \Leftrightarrow x \not\in R_\Box(s, T; T_\varphi(\cdot)).
\]  

(20)

In addition, define \( g_{\Box_{[s_1,s_2]} \varphi}(s, x) = h_{\Box_{[s_1,s_2]} \varphi}(s, x) \), where \( h_{\Box_{[s_1,s_2]} \varphi}(s, x) \) is such that \( R_{\Box_{[s_1,s_2]} \varphi}(s, T; T_\varphi(\cdot)) = \{ x : h_{\Box_{[s_1,s_2]} \varphi}(s, x) < 0 \} \). Then, we have

\[
\exists u(\cdot) \in U, \forall d(\cdot) \in D, \text{ if } (\xi_{s,s_1}^{u(\cdot),d(\cdot)}(\cdot), s) \models \Box_{[s_1,s_2]} \varphi \Leftrightarrow g_{\Box_{[s_1,s_2]} \varphi}(s, x) > 0.
\]  

(21)

Note that \( g_{\Box_{[s_1,s_2]} \varphi}(s, x) \) satisfies the HJ VI (8), so \( g_{\Box_{[s_1,s_2]} \varphi} \) is a value function associated with \( \Box_{[s_1,s_2]} \).

Prop. 2 follows from the definition of the always operator and minimal RS. Controller synthesis will be discussed in Sec. 4 in Lem. 1.

4 Controller Synthesis

In this section, we provide a controller synthesis technique that is different from the optimal controller synthesis typically found in works related to HJ reachability. Instead of computing the optimal control, we propose to consider the set of controllers that satisfy a given STL formula by viewing the value function as a Lyapunov-like function. We call this set of controllers the least-restrictive feasible controller set (LRFCS), which ensures the value function does not decrease along trajectories, an idea that is core to many reachability methods other than HJ [21, 23].

In the context of STL, the key benefit of considering the LRFCS is that a single controller synthesis procedure can be repeatedly used for satisfying recursively defined STL formulas. Thus, the LRFCS is what allows HJ reachability to leverage the complexity of specifications in the STL framework. The following lemma formalizes the LRFCS:

**Lemma 1** (Least-restrictive feasible controller set (LRFCS) for satisfying an STL formula). Let \( \varphi \) be any STL formula, and \( g_\varphi \) be a value function associated with \( \varphi \). Suppose that at state \( s \), the system (1) is at a state \( x \) such that \( g_\varphi(t, x) \geq c \). Define

\[
U_\varphi := \{ u(\cdot) \} ; \forall s \geq t, u(\cdot) \in \bar{U}_\varphi(s, x), \text{ where}
\]

\[
\bar{U}_\varphi(s, x) = \begin{cases} 
U_\varphi(s, x), & \text{if } g_\varphi(s, x) \leq c, \\
\emptyset, & \text{otherwise.}
\end{cases}
\]  

(22b)

\[
U_\varphi(s, x) = \{ u : \frac{\partial}{\partial s} g_\varphi(s, x) + \min_{d \in D} \nabla g_\varphi(s, x) \cdot f(s, x, u, d) \geq 0 \}.
\]  

(22c)
Then, \( u(\cdot) \in \mathcal{U}_\varphi \) implies \( \forall s \geq t, \forall d(\cdot) \in \mathcal{D}_\varphi, g_\varphi(s, x_{s,t}^{u(\cdot),d(\cdot)}(s)) \geq c \).

**Proof:** We start with the expression in (22c):

\[
0 \leq \frac{\partial}{\partial s} g_\varphi(s, x) + \min_{d \in \mathcal{D}} \nabla g_\varphi(s, x) \cdot f(s, x, u, d)
\]

\[
= \min_{d \in \mathcal{D}} \frac{\partial}{\partial s} g_\varphi(s, x) + \nabla g_\varphi(s, x) \cdot f(s, x, u, d) = \min_{d \in \mathcal{D}} g_\varphi(s, x)
\]

The minimization of \( d \) corresponds to the disturbance behaving adversarially to drive the system away from the satisfaction of \( \varphi \). Therefore, for all \( d \), we have \( g_\varphi(s, x) \geq 0 \). In addition, since for all \( s \geq t \), we have \( u(s) \in \mathcal{U}_\varphi(s) \) whenever \( g_\varphi(s, x) \leq c \) and in particular when \( g_\varphi(s, x) = c \), we must have that \( \forall s \geq t, \forall d(\cdot), g_\varphi(s, x_{s,t}^{u(\cdot),d(\cdot)}(s)) \geq c \).

**Corollary 1.** If \( c > 0 \), then Lem. 1 is equivalent to

\[
u(\cdot) \in \mathcal{U}_\varphi \Rightarrow \forall d(\cdot), (x_{s,t}^{u(\cdot),d(\cdot)}(\cdot), s) \models \varphi.
\]

**Remark 2.** Lem. 1 provides a set of controllers for satisfying \( \varphi \mathcal{U}_{[\tau_1, \tau_2]} \psi \) using the value function \( g_\varphi \mathcal{U}_{[\tau_1, \tau_2]} \psi \) in Prop. 1 and for \( \Box_{[\tau_1, \tau_2]} \varphi \) using \( g_{\Box_{[\tau_1, \tau_2]} \varphi} \) in Prop. 2.

Note that since \( g_\varphi \) is a value function associated with \( \varphi \), the expression in (22c) is always non-empty. This is true by construction of the value function in either (8) or (11), in which the optimal controller is selected in a dynamic programming framework. The associated optimal controller, which we have omitted in this paper, guarantees the existence of a feasible controller; for a more thorough discussion of the optimal controller, please see one of [4, 10, 16].

We now continue our discussion of repeated controller synthesis via the LRFCG for recursively defined STL formulas. We first consider cases which do not involve control conflicts in Sec. 4.1; here, we provide a simple additional control logic to tie together multiple controllers from STL formulas that make up a more complex STL formula. Next, in 4.2, we address the cases involving control conflicts by describing why control conflicts arise, and how they can be resolved.

### 4.1 Negation, logical disjunction, and “always”: no control conflicts

In this section we consider \( \neg, \lor, \land, \Box_{[\tau_1, \tau_2]} \), for which the discussion of controller synthesis is relatively straightforward. We first note that to synthesize a controller that guarantees the satisfaction of \( \neg \varphi \), one would simply first negate \( \varphi \), and then perform controller synthesis recursively. This can be considered a "pre-processing step" when STL formulas are in Negation Normal Form (cf. Sec. 2.2).

Next, we provide a joint controller for satisfying \( \varphi \lor \psi \), given two controllers that respectively guarantee the satisfaction of \( \varphi \) and \( \psi \). Intuitively, Prop. 3 first states that applying a controller (in \( \mathcal{U}_\varphi \)) that satisfies \( \varphi \) also implies satisfaction of \( \varphi \lor \psi \). On the other hand, if \( \varphi \) cannot be satisfied, then by assumption \( \psi \) can be satisfied using a controller drawn from \( \mathcal{U}_\psi \), thereby also satisfying \( \varphi \lor \psi \).

**Proposition 3 (Joint controller for logical disjunction).** Suppose (24) holds, and \( u(\cdot) \in \mathcal{U}_\varphi \Rightarrow \forall d(\cdot), (x_{s,t}^{u(\cdot),d(\cdot)}(\cdot), s) \models \psi \). Then \( (x_{s,t}^{u_{\varphi \lor \psi}(\cdot),d(\cdot)}(\cdot), s) \models \varphi \lor \psi \), where

\[
u_{\varphi \lor \psi}(\cdot) \in \begin{cases} \mathcal{U}_\varphi, & \text{if (24) holds,} \\ \mathcal{U}_\psi, & \text{otherwise.} \end{cases}
\]

Finally, given a set of controllers \( \mathcal{U}_\varphi \) that guarantees satisfaction of \( \varphi \), and a set of controllers \( \mathcal{U}_{\Box_{[\tau_1, \tau_2]} \varphi} \) that guarantees satisfaction of \( \Box_{[\tau_1, \tau_2]} \varphi \), we provide the set of controllers that first drives the system into a state from which \( \varphi \) can be satisfied using a controller in \( \mathcal{U}_{\Box_{[\tau_1, \tau_2]} \varphi} \), and then drives the system to satisfy \( \varphi \) using a controller in \( \mathcal{U}_\varphi \).
Proposition 4 (Compound controller for the always operator). Define some $s' \in [s + s_1, s + s_2]$. Suppose (2A) holds, and $u(\cdot) \in \bigcup_{[s_1, s_2]} \mathcal{U} \Rightarrow \forall d(\cdot), \varepsilon(\cdot \in \mathcal{G}) \mathcal{E}(\cdot) \subseteq \mathcal{S}_{[s_1, s_2]}$. Then, $(\varepsilon_{x,s}^{(\cdot),d(\cdot)(s)}, s) \models \varphi$, where $\bar{u}(\cdot) = \begin{cases} \bigcup_{[s_1, s_2]} \mathcal{U}, & \text{if } s < s', \\ \mathcal{U}_\varphi, & \text{if } s \geq s'. \end{cases}$

The reasoning behind Prop. 4 is as follows: If one applies the controller drawn from $\bigcup_{[s_1, s_2]} \mathcal{U}$, until some time $s'$ between $s + s_1$ and $s + s_2$, the system is then in a position to satisfy $\varphi$. The satisfaction of $\varphi$ is guaranteed by applying a controller drawn from $\mathcal{U}_\varphi$ at and after time $s'$.

4.2 Logical conjunction and "until": avoiding control conflicts
In this section, we consider the operators $\land$ and $\bigcup$, which require more careful treatment due to the possibility of control conflicts. For example, even if controllers drawn from $\mathcal{U}_\varphi$ and $\mathcal{U}_\psi$ allow the system to independently satisfy $\varphi$ and $\psi$ respectively, there may not exist a controller that leads to the satisfaction of $\varphi \land \psi$. This is because the system may only use a single control at any given time, and if $\mathcal{U}_\varphi$ and $\mathcal{U}_\psi$ do not intersect, then there would not be a controller that guarantees simultaneous satisfaction of $\varphi$ and $\psi$. An analogous argument also applies for $\varphi \bigcup_{[s_1, s_2]} \psi$.

Therefore, controller synthesis for satisfaction of $\psi$ in the expressions $\varphi \land \psi$ or $\varphi \bigcup_{[s_1, s_2]} \psi$ must use the LRFCs with respect to $\varphi$ so the satisfaction of $\varphi$ is guaranteed. This requirement can be formalized by restating Prop. 1 and 2 with the modified control constraint $u(\cdot) \in \mathcal{U}_\varphi$ instead of $u(\cdot) \in \mathcal{U}$. The full restatements can be found in the appendix. Here, we highlight that optimizing over a restricted function set $u(\cdot) \in \mathcal{U}_\varphi$, or equivalently, over a restricted control signal set $\mathcal{U}_\varphi$, does not significantly increase the computation burden for control and disturbance affine systems. To see this, let the dynamics (1) be $\dot{x} = f_x(t, x) + f_d(t, x)u + f_d(t, x)d$. Then, expression in (22c) becomes

$$\frac{\partial}{\partial s} g_\varphi(s, x) + \nabla g_\varphi(s, x) \cdot f_x(t, x) + \min_{d \in D} \nabla g_\varphi(s, x)f_d(t, x)d + \nabla g_\varphi(s, x)f_d(t, x)u \geq 0,$$  

which is a single affine control constraint. When solving the HJ VIs (8) and (11), the optimization $u \in \mathcal{U}$ becomes $u \in \mathcal{U}_\varphi$, which, according to (26), at most involves adding an affine constraint. In typical scenarios in which $u \in \mathcal{U}$ is a box constraint, adding (26) would result in a polytopic constraint, and therefore the optimization $u \in \mathcal{U}_\varphi$ does not involve significantly more computation than the optimization $u \in \mathcal{U}$.

5 Practical Summary of Theoretical Results
In this section, we compile all the theory in this paper into a list of corresponding STL and reachability operators. Given this list, one would be able to recursively compute the set of states that satisfies an arbitrary STL formula, and to synthesize the corresponding feedback controller. We also provide an illustrative example and brief discussion on minimum violation.

5.1 List of corresponding STL, set, and reachability operators
Using Table 1, a value function representing any STL formula $\varphi$ can be computed through a sequence of reachability computations as well as point-wise negation, minimization, and maximization of functions. The number of reachability computations required is the number of until and always operators in the STL formula $\varphi$, and each individual reachability computation scales according to the reachability method. In the case of the HJ method, computational complexity of a single reachability computation
is exponential with the number of system dimensions in (1). Practically speaking, this means that systems with up to 5D can be tractably analyzed.

In theory, any controller in the LRFCS can be used to satisfy $g_{\varphi}$, as long as the state $x(s)$ is such that $g(x, s) > 0$. In practice, additional criteria can be used for choosing the controller within the LRFCS. One simple choice, which is typically used in HJ reachability, is to choose the controller that maximizes $g_{\varphi}$ in Eq. (22c).

<table>
<thead>
<tr>
<th>Formula</th>
<th>Value function computation</th>
<th>Controller</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varphi \lor \psi$</td>
<td>Set $g_{\varphi \lor \psi} = \max {g_{\varphi}, g_{\psi}}$</td>
<td>$u(\cdot) \in U_{\varphi \lor \psi}$ in (25)</td>
<td>Eq. (15b), Prop. 3</td>
</tr>
<tr>
<td>$\varphi \land \psi$</td>
<td>Given $U_{\varphi}$. 1. compute $g_{\varphi}$ with constraint $u(\cdot) \in U_{\varphi}$, 2. set $g_{\varphi \land \psi} = \min{g_{\varphi}, g_{\psi}}$.</td>
<td>$u(\cdot) \in U_{\varphi}$ in (22) with $\varphi = \psi, g_{\varphi} = g_{\psi}, c \geq 0$</td>
<td>Eq. (15a), Sec. 4.2</td>
</tr>
<tr>
<td>$\neg \varphi$</td>
<td>Set $g_{\neg \varphi}(x) = -g_{\varphi}(x)$</td>
<td>Synthesized recursively from negated formula</td>
<td>Eq. (15c), Sec. 4</td>
</tr>
<tr>
<td>$\psi_{[s_1, s_2]}^\varphi$</td>
<td>Given $U_{\varphi}, \psi_{[s_1, s_2]}^\varphi$. 1. solve the HJ VI (8) with $T_{\varphi}, g_{\varphi}$ given in (16) and constraint $u(\cdot) \in U_{\varphi}$ to obtain the RS represented by $g_{R1M}$. 2. set $g_{\psi_{[s_1, s_2]}^\varphi} = -g_{R1M}$.</td>
<td>$u(\cdot) \in U_{\varphi}$ in (22) with $\varphi = \psi_{[s_1, s_2]}^\varphi, g_{\varphi} = g_{\varphi_{[s_1, s_2]}^\varphi}, c \geq 0$</td>
<td>Prop. 1, Sec. 4.2</td>
</tr>
<tr>
<td>$\psi_{[s_1, s_2]}^\varphi$</td>
<td>Given $U_{\varphi}, g_{\varphi}$, $\psi_{[s_1, s_2]}^\varphi$. 1. solve the HJ VI (11) with $T_{\varphi}$ given in (19) and constraint $u(\cdot) \in U_{\varphi}$ to obtain the $g_{R1M}$. 2. set $g_{\psi_{[s_1, s_2]}^\varphi} = g_{R1M}$.</td>
<td>$u(\cdot) \in U_{\varphi}$ in (22) with $\varphi = \psi_{[s_1, s_2]}^\varphi, g_{\varphi} = g_{\varphi_{[s_1, s_2]}^\varphi}, c \geq 0$ to satisfy $\psi_{[s_1, s_2]}^\varphi$. 3. Use $u(\cdot) \in U_{\varphi}$ when $s \in [s_1, s_2 + \epsilon]$ to satisfy $\psi$.</td>
<td>Prop. 2, Sec. 4.1</td>
</tr>
</tbody>
</table>

Table 1. Summary of theoretical results. Given any STL formula $\varphi$, this table can be used to identify the sequence of reachability computations, and point-wise negation, minimization, and maximization of functions that produces a value function $g_{\varphi}$ and the corresponding LRFCS $U_{\varphi}$.

5.2 Illustrative Example

As a concrete example, consider the formula $\varphi = \square \mu_1 \Diamond_{I_1} \mu_1 \land (\mu_2 \mu_3 \Diamond_{I_2} \mu_3)$, for functions $\mu_1, \mu_2, \mu_3$ representing atomic propositions, and time intervals $I_1, I_2, I_3$. Using Table 1, one could perform the following steps, which involve three reachability computations in total, to obtain a value function representing the set of states from which $\varphi$ can be satisfied, and the corresponding LRFCS:

1. Compute $g_{\square \mu_1}$ with the control constraint $u(\cdot) \in U$ (no control constraints other than that given by the dynamical system) according to the fifth row of Table 1.
2. Compute $g_{\Diamond_{I_1} \mu_1}$ and $U_{\Diamond_{I_1} \mu_1}$, with the control constraint $u(\cdot) \in U$ (no control constraints other than that given by the dynamical system) according to the sixth row of Table 1.
3. Compute $g_{\mu_2}U_{\mu_3}$ and $U_{\mu_2}U_{\mu_3}$ with the control constraint $u(\cdot) \in U_{\Diamond_{I_3} \mu_1}$ (extra control constraint to ensure satisfaction of $\Diamond_{I_3} \Diamond_{I_2} \mu_1$) according to the fifth row of Table 1. Note that this is also step 1 in the third row of Table 1.
4. Set $g_{\varphi} = \min(g_{\square \mu_1}, g_{\Diamond_{I_1} \mu_1}, g_{\mu_2}U_{\mu_3})$. The set of states from which $\varphi$ can be satisfied is then given by $\{s : g_{\varphi} > 0\}$ (step 2 in the third row of Table 1).
5. The LRFCS for satisfying $\varphi$ is given by $U_{\mu_2}U_{\mu_3}$, which already accounts for the satisfaction of $\Diamond_{I_1} \Diamond_{I_2} \mu_1$.

5.3 Minimum violation interpretation

Lem. 1 and Cor. 1 establish the condition to guarantee the continued satisfaction of an STL formula $\varphi$ using the LRFCS in (22), as long as the formula is satisfied at some state.
x and time t. However, in some situations, the system may be in a situation in which a desired STL formula \( \varphi \) cannot be satisfied. Mathematically, this is characterized by 
\[ g_\varphi(s, x) \leq 0. \]
Such a situation could occur if the controller in (22) is not used, or if there is a sudden change in the STL formula the system needs to satisfy, perhaps as a result of a sudden change in the objective of the system.

Since Lem. 1 holds for any value of c, the controller in (22) is also the “minimum violation controller” in the situation in which \( g_\varphi(s, x) \leq 0. \) By using the controller in (22), one can guarantee that \( g_\varphi(s, x) \) never decreases, which is interpreted as “the situation is guaranteed to not become worse, even under the worst-case disturbance”. On the other hand, if the disturbance does not behave in worst-case fashion, \( \dot{g}_\varphi(s, x) \) may be positive, and \( g_\varphi(s, x) \) may evolve to become positive so that \( \varphi \) is satisfied at a later time. An example of a system “recovering” from a state that initially does not satisfy a desired STL formula is shown in the appendix.

5.4 Computational Scalability

For every temporal operator (eventually, until, or always), a value function must be computed to quantify the set of states from which the temporal operator can be satisfied and to synthesize a corresponding controller. Therefore, the computational complexity scales linearly with the number of temporal operators.

In addition, in this paper, each value function computation involves solving an HJVI, which scales exponentially with the number of state space dimensions. Currently, this means that value function computations for systems of more than five state space dimensions are intractable; computational burden is a general challenge in formal verification methods. Incorporating computationally scalable methods, such as [2, 19, 21] which maintain guarantees of STL formula satisfaction, while balancing computational burden, conservatism, and nonlinearity of system dynamics is part of future work.

6 Simulations and Experiments

In this section, we demonstrate our methodology on an example representative of a common autonomous driving scenario. We will consider a car overtaking maneuver in a 2-lane highway. As this example involves the logical conjunction of compound STL formulas, we will use the notion of LRFCS in Sec. 4.2 to avoid control conflicts.

In addition to this example, a toy numerical example, and a different numerical example and robotic experiment related to autonomous driving are presented in the appendix. The toy numerical example validates the notion of LRFCS. In the example involving autonomous driving in the appendix, we considered an autonomous car attempting to make a left turn in a four-way intersection just after the traffic light has transitioned to yellow. We performed the control synthesis described in Sec. 5. Here, the desired outcome is either making a successful left turn or stopping in initial lane, depending on the behavior of another car travelling in the opposite direction. We also demonstrated the notion of minimum violation introduced in Sec. 5.3 when the desired outcome is not initially guaranteed. For all examples, after obtaining the LRFCS that guarantees the satisfaction of the desired STL formula, we simply chose the controller that maximizes the \( g_\varphi \) expression in (22c).

6.1 Hardware Setup

We performed all computations\(^6\) on a laptop with a Intel Core i5-6300HQ CPU and 8GB of RAM. Reachability computations which produced the value functions were done

\(^6\) The code used in this paper is available at \url{https://github.com/StanfordASL/st1hj}
using the beacs toolbox\textsuperscript{7}. Numerical simulations for both examples were performed in MATLAB using the helperOC\textsuperscript{8} and Level Set Methods\textsuperscript{9} [27] toolboxes. For the hardware experiments\textsuperscript{10}, we used TurtleBots in place of cars and tracked their positions using a Vicon motion capture system. With the exception of using MATLAB for the real-time evaluation of the optimal controller, all other processes including message-passing between devices and low-level controls were implemented in ROS using Python.

6.2 Highway example

We now consider a highway overtaking scenario. Initially, the autonomous car is behind a ‘slow car’ which moves at a constant speed $v_3$. The autonomous car attempts to overtake the slow car while avoiding a collision with an ‘adjacent car’ which travels in the left lane with a possible range of velocities. In addition to avoiding collisions with both cars, the autonomous car will also have to be able to re-enter the right lane within a short duration of time. This additional safety constraint is to plan for situations such as when emergency vehicles require passage through the left lane.

The adjacent car, which acts as the disturbance as described in Sec. 2.3, has its position represented by $y_2$. Since the slow car travels at a constant speed, we express the joint dynamics of the three vehicles in the reference frame of the slow car:

\[
\begin{align*}
\dot{x}_R &= v_R \cos \theta_R, & \quad \dot{y}_R &= v_R \sin \theta_R - v_3, & \quad \dot{\theta}_R &= \omega_R, & \quad \dot{v}_R &= a_R, & \quad \dot{y}_2 &= v_2 - v_3.
\end{align*}
\]  

(27)

The autonomous car’s primary objective is to overtake the slow car, which is captured by the proposition $\psi^{\text{pass}}$. It needs to adhere to the traffic laws and avoid colliding with either the slow car or the adjacent car. This constraint is represented by $\varphi$. If such a collision is possible or the overtaking maneuver cannot be achieved within the specified time frame, the autonomous car will then have the secondary objective of remaining behind the slow car, $\psi^{\text{stay}}$. We also include an additional recurrence requirement for the autonomous car to always be within 5 seconds of re-entering the right lane, represented by $\psi^{\text{lane}}$. The function representations of the STL propositions can be found in the appendix. Overall, the robot must satisfy the following STL formula:

\[
\square_{[0,25]} (\square_{[0,5]} \psi^{\text{lane}} \land (\varphi \square_{[0,25]} \psi^{\text{pass}} \lor \varphi \square_{[0,25]} \psi^{\text{stay}})), \quad \psi^{\text{avoid}} = \varphi^{\text{off-road}} \land \varphi^{\text{on-road}} \land \varphi^{\text{avoid}}
\]  

(28)

The results from numerical simulations and experiments with TurtleBots are shown in Fig. 2 and Fig. 3. After obtaining the value function and LRFCs for $\square_{[0,25]} (\square_{[0,5]} \psi^{\text{lane}})$, the LRFCs is used as an additional constraint for the reachability computations associated with $\varphi \square_{[0,25]} \psi^{\text{pass}} \lor \varphi \square_{[0,25]} \psi^{\text{stay}}$ to avoid control conflicts as discussed in Sec. 4.2. This process is described in greater detail in the appendix. The white circle represents the autonomous car, while the red and green car represents the adjacent car and slow car respectively. The colors represent the values of $\max(\eta_{\varphi \square_{[0,25]} \psi^{\text{pass}}}, \eta_{\varphi \square_{[0,25]} \psi^{\text{stay}}})$. In Fig. 2, the autonomous car has sufficient time to pass the slow car as the adjacent car passes by quickly. It waits until the adjacent car crosses a threshold position, determined automatically from the reachability computations, before committing to the overtaking maneuver. This takes into account the possibility that the adjacent car may slow down drastically at any time. We can observe that the value at the autonomous car’s position increases from $t = 0s$ to $8s$ just as the car begins the overtaking maneuver; this is a result of the adjacent car not behaving in the worst-case manner.

\textsuperscript{7} https://github.com/HJReachability/beacs

\textsuperscript{8} https://github.com/HJReachability/helperOC

\textsuperscript{9} http://www.cs.ubc.ca/~mitchell/ToolboxLS/

\textsuperscript{10} The videos can be viewed at https://www.youtube.com/playlist?list=PL8-2mtIlFIJoNhxcG17s1WX2W3kEW-9Pb
In Fig. 3, the autonomous car is unable to pass the slow car within the time horizon and stops behind the slow car due to the adjacent car moving too slowly to reach the time-specific threshold position. The value at the autonomous car's position remains low. Computation of the value function took approximately 8 hours. The erratic movement seen in Fig. 3 is caused by the coarse state discretization coupled with the bang-bang controller; this numerical artifact can potentially be alleviated using a finer resolution grid along with GPU parallelization to offset additional computational burden.

Fig. 2. The autonomous car (white) successfully overtakes the slow car (green) as the adjacent car (red) moves quickly to "make room" for the passing maneuver. Left: Contour plot of the value function. Right: Time-lapse of the experiment.

Fig. 3. The autonomous car (white) remains behind the slow car (green); the adjacent car (red) moves too slowly. Left: Contour plot of the value function. Right: Time-lapse of the experiment.

7 Conclusions and Future Work
In this paper, we presented the combination of Signal Temporal Logic (STL) and Hamilton-Jacobi (HJ) reachability as a versatile verification technique that inherits the strengths of both. From the perspective of STL, our method moves beyond controller synthesis methods that generate single trajectories; instead, we provide guarantees in terms of sets of states, and produce state feedback controllers. From the perspective of HJ reachability, our method looks beyond single reachability problems, and takes advantage of the expressiveness of STL to define sequences of reachability problems. In terms of computational complexity, the number of reachability computations required for set computation and controller synthesis is the same as the number of temporal operators in the STL formula; for any individual reachability computation, our method inherits the same computational complexity as the reachability method. Our approach has been validated in three numerical examples and two robotic experiments.
Our work spawns a number of future research directions. For example, one could attempt to alleviate the effects of bang-bang control by using the LRFCS in conjunction with a different controller synthesis framework such as model predictive control. Also, in general, there may be many different orderings of reachability operators that produce LRFCS for an STL formula; thus, it is important to investigate the effect of different prioritization of STL formulas when using the LRFCS.

Lastly, the ideas in this paper, although presented in the context of STL and HJ reachability, are transferable to other temporal logics and reachability methods; exploring other formulations of temporal logic and reachability would help make trade-offs among computational scalability, conservatism, generality of system dynamics, and expressiveness of logical formulas. In particular, there are several suggested ways to alleviate the “curse of dimensionality” in formal verification. First, preliminary studies on GPU parallelization has been promising, allowing value functions to be computed on a coarse grid for a 7D system [22]. Second, several system decomposition methods [7, 8] may be used to reduce dimensionality for specific types of systems. Finally, reachability methods other than the HJ method can be used [2, 17, 21, 23].

References