The International Journal of Robotics Research
XX(X):1–24
©The Author(s) 2023
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Online Control Synthesis for Uncertain Systems under Signal Temporal Logic Specifications

Pian Yu[1,*], Yulong Gao[1,*], Frank J. Jiang[2,3], Karl H. Johansson[2,3], and Dimos V. Dimarogonas[2,3]

Abstract

Signal temporal logic (STL) formulas have been widely used as a formal language to express complex robotic specifications, thanks to their rich expressiveness and explicit time semantics. Existing approaches for STL control synthesis suffer from limited scalability with respect to the task complexity and lack of robustness against the uncertainty, e.g., external disturbances. In this paper, we study the online control synthesis problem for uncertain discrete-time systems subject to STL specifications. Different from existing techniques, we propose an approach based on STL, reachability analysis, and temporal logic trees. First, based on a real-time version of STL semantics, we develop the notion of tube-based temporal logic tree (tTLT) and its recursive (offline) construction algorithm. We show that the tTLT is an under-approximation of the STL formula, in the sense that a trajectory satisfying a tTLT also satisfies the corresponding STL formula. Then, an online control synthesis algorithm is designed using the constructed tTLT. It is shown that when the STL formula is robustly satisfiable and the initial state of the system belongs to the initial root node of the tTLT, it is guaranteed that the trajectory generated by the control synthesis algorithm satisfies the STL formula. We validate the effectiveness of the proposed approach by several simulation examples and further demonstrate its practical usability on a hardware experiment. These results show that our approach is able to handle complex STL formulas with long horizons and ensure the robustness against the disturbances, which is beyond the scope of the state-of-the-art STL control synthesis approaches.

Keywords

Signal temporal logic, uncertain systems, online control synthesis, tube-based temporal logic tree, and reachability analysis

1 Introduction

1.1 Motivation

The rapid growth of robotic applications, such as autonomous vehicles and service robots, has stimulated the need for new control synthesis approaches to safely accomplish more complex objectives such as nondeterministic, periodic, or sequential tasks (Kress-Gazit et al., 2018). Temporal logics, such as linear temporal logic (LTL) (Baier and Katoen, 2008), metric interval temporal logic (MITL) (Koymans, 1990), and signal temporal logic (STL) (Maler and Nickovic, 2004), have shown capability in expressing such objectives for dynamical systems. However, traditional control methods (e.g., linear quadratic regulator, model predictive control, and adaptive control) are originally developed for simple control objectives such as stability and set invariance (Baillieul and Samad, 2021), and these methods are restrictive to handle complex temporal logic tasks. Thus, advanced control methods must be developed to fill this gap.

As a more recently developed temporal logic, STL allows the specification of properties over dense-time. This makes it suitable for expressing complex specifications that may involve specific timing requirements or deadlines. Such specifications include time-constrained reachability (e.g., $\mathsf{F}_{[0,60]}\mathsf{G}_{[0,20]}A$: visit region A within 60 seconds and stay there for another 20 seconds) and time-constrained surveillance (e.g., $\mathsf{GF}_{[10,50]}A \wedge \mathsf{GF}_{[10,50]}B$: visit regions A and B every 10 – 50 seconds). STL was originally evaluated over continuous-time signals in Maler and Nickovic (2004), and then extended to discrete-time signals in Raman et al. (2015). STL contains predicates as the atomic elements and the truth value of each predicate is evaluated through a predicate function. Due to a number of advantages, such as explicitly treating real-valued signals (Maler and Nickovic, 2004), and admitting robustness semantics (Fainekos and Pappas, 2009), control synthesis under STL specifications has gained popularity and many efforts have been devoted for STL control synthesis in the last few years. Nevertheless, existing approaches usually suffer from limited scalability

[1] Department of Computer Science, University of Oxford, UK
[2] Division of Decision and Control Systems, KTH Royal Institute of Technology, Stockholm, Sweden
[3] Digital Futures, Stockholm, Sweden

[*] Pian Yu and Yulong Gao were at the KTH Royal Institute of Technology when this work was conducted.

Corresponding author:
Pian Yu, Department of Computer Science, University of Oxford, UK.
Email: pian.yu@cs.ox.ac.uk

with respect to the task complexity and lack of robustness to the uncertainties from the robotic systems.

The complexity of STL formulas (e.g., the time horizon or nestedness) is in general crucial in deciding the complexity of control synthesis approaches, e.g., optimization-based methods. For example, the number of integer variables in mixed-integer program based approaches grows exponentially with respect to the time horizon. Other popular approaches, e.g., barrier function-based methods, only handle a fragment of STL formulas with non-nested temporal operators. On the other hand, robotic systems are usually corrupted by external disturbances and accompanied by modeling errors. These uncertainties make it challenging to reason about STL specifications due to the encoded time semantics. To the best of our knowledge, few control approaches can efficiently handle system uncertainties with robustness guarantees for STL specifications.

1.2 Related work

1.2.1 LTL or MITL control synthesis

LTL focuses on the Boolean satisfaction of properties by given signals while MITL is a continuous-time extension that allows to express temporal constraints. Existing control approaches that use LTL or MITL mainly rely on a finite abstraction of the system dynamics and a language equivalent automata (Gastin and Oddoux, 2001) or timed-automata (Alur et al., 1996) representation of the LTL or MITL specification. The controller is synthesized by solving a game over the product automata (Belta et al., 2007, 2017; Zhou et al., 2016). Other control approaches include optimization-based (Wolff and Murray, 2016; Fu and Topcu, 2015) and sampling-based methods (Vasile and Belta, 2013; Kantaros and Zavlanos, 2018).

One of the most relevant works is Gao et al. (2022). In this paper, the notion of temporal logic tree (TLT) is proposed for LTL specifications and the corresponding TLT-based control synthesis algorithm is developed. Despite some relevance, it is far from straightforward to extend these results to general STL formulas. Some significant differences between our paper and Gao et al. (2022) are highlighted as follows. *First*, the definitions and semantics of TLT and tTLT are largely different. In particular, the time constraints encoded in STL formulas require a new notion of real-time STL semantics for connecting tTLT and STL. *Second*, the control synthesis algorithms in this paper are largely different from that in Gao et al. (2022). In order to carefully monitor the time constraint satisfaction in the STL formulas, the online synthesis algorithm needs in an appropriate way to track the set node (Algorithm 7), update the tTLT (Algorithm 8), and update the post set (Algorithm 11).

1.2.2 STL control synthesis

Given the extensive literature studying STL, we restrict our attention to the following STL synthesis approaches.

Optimization-based methods Optimization methods leverage on the fact that STL formulas can be encoded as mixed-integer constraints. Based on this, STL control synthesis can be obtained by solving a series of optimization problems (Raman et al., 2015, 2014). To avoid the complexity of integer-based optimization, smooth approximations have been proposed by using sequential quadratic programming (SQP) (Gilpin et al., 2020) or convex-concave programming (Takayama et al., 2023). Recent work studies how to reduce the integer variables using the property of logic operators (Kurtz and Lin, 2022). However, these results are restricted to deterministic systems.

An extension of the mixed-integer formulation is investigated for linear systems with additive bounded disturbances in Sadraddini and Belta (2015), where the model predictive controller is obtained by solving the optimization problem at each time step in a receding horizon fashion. In Farahani et al. (2018), a model predictive controller in the form of shrinking horizon is developed for linear systems with stochastic disturbances under STL constraints. One drawback of these approaches is the exponential computational complexity which makes it difficult to be applied to STL formulas with long time horizons. In addition, a stochastic gradient decent-based method is developed to optimize the probability that the stochastic system satisfies an STL specification (Scher et al., 2022).

Barrier function methods Barrier function methods are mainly used for continuous-time systems. The idea is to transfer the STL formula into one or several (time-varying) control barrier functions, and then obtain feedback control laws by solving quadratic programs (Lindemann and Dimarogonas, 2018). This method is computationally efficient. However, as the existence and design of barrier functions are still open problems, it currently mainly applies to deterministic affine systems. In Yang et al. (2020), the authors consider linear cyber-physical systems with continuous-time dynamics and discrete-time controllers. The proposed offline trajectory planner is based on a mixed integer quadratic program that utilizes control barrier functions to generate satisfying trajectories in continuous-time. Other control synthesis approaches include sampling-based (Vasile et al., 2017a; Karlsson et al., 2020) and learning-based methods (Venkataraman et al., 2020; Kapoor et al., 2020). In addition, control synthesis for multi-agent systems and STL specifications is recently considered in Lindemann and Dimarogonas (2019); Buyukkocak et al. (2021); Sun et al. (2022).

Reachability-based methods Reachability is a fundamental notion in systems and control and reachability analysis has been widely used for simple control objectives, e.g., stability and safety (Bertsekas, 1972). In Roehm et al. (2016), a reachability-based method is proposed for STL model checking by converting an STL formula to reachset temporal logic. This method is refined in (Kochdumper and Bak, 2023) for linear deterministic systems by adequately tuning parameters to enforce over-approximation error to zero. Chen et al. (2018b) recognizes the connection between temporal logic operators and reachability, and then exploits Hamilton-Jacobi reachability for STL control synthesis, which has served as an inspiration to our work. Although there exists close relevance between Chen et al. (2018b) and our paper, some remarkable differences should be highlighted.

First, we find that the connection between temporal logic operators and reachability in Chen et al. (2018b) may not hold for nested STL formulas. We improve and extend this point by introduce the new real-time STL semantics, which is beyond Chen et al. (2018b) and motivates the proposal

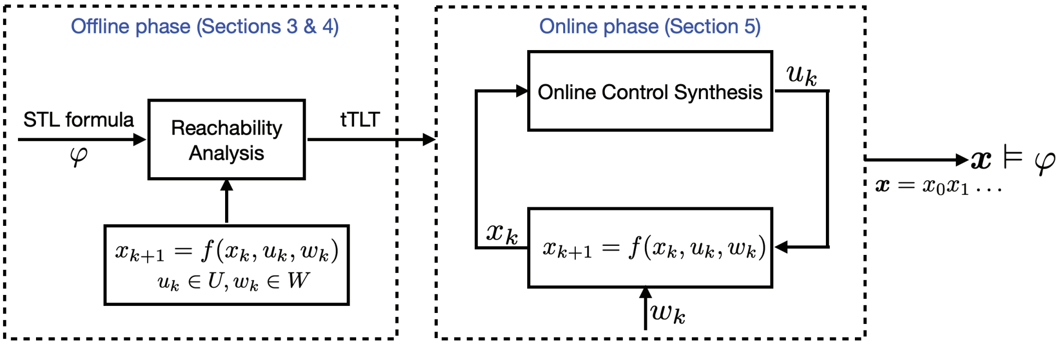

Offline phase (Sections 3 & 4)
STL formula
φ
Reachability Analysis
tTLT
$x_{k+1} = f(x_k, u_k, w_k)$
$u_k \in U, w_k \in W$
Online phase (Section 5)
Online Control Synthesis
u_k
x_k
$x_{k+1} = f(x_k, u_k, w_k)$
w_k
$\boldsymbol{x} \models \varphi$
$\boldsymbol{x} = x_0 x_1 \ldots$

Figure 1. The tTLT-based STL control synthesis framework.

of the tTLTs. Second, while the control design in Chen et al. (2018b) is restricted to non-nested STL formulas, we propose a systemic way to online synthesize the robust controller for more general STL formulas. In our paper, thanks to the semantic relation between the STL formulas and their corresponding tTLTs, we are able to perform control synthesis over the tTLT, instead of the STL formulas, with a correct-by-construction guarantee.

Sampling-based/data-driven/learning-based methods Motivated by the success of sampling-based methods in motion planning, some recent works consider the extension to the STL planing. In Vasile et al. (2017b), an RRT* approach is developed to incrementally construct a tree such that an STL specification is maximally satisfied. Barbosa et al. (2019) uses a cost function to guide exploration for satisfying a restricted fragment of STL formulas. More recent work Ho et al. (2022) integrates automaton theory and sampling-based methods for STL control synthesis of nonlinear deterministic system. When the system model is unknown, a direct data-driven STL synthesis method is studied using behavioral characterization of linear models (van Huijgevoort et al., 2023).

In addition, learning-based methods have been becoming popular for STL verification and synthesis. For example, neural network-based methods are investigated in Liu et al. (2021); Leung and Pavone (2022); Hashimoto et al. (2022). By generating rewards in proper ways from STL specifications, reinforcement learning-based approaches have been proposed in Venkataraman et al. (2020); Kapoor et al. (2020); Singh and Saha (2023); Hamilton et al. (2022).

1.2.3 Other related work Beyond the above literature review, the properties of STL formulas have been studied in different contexts or for different purposes. In Leung et al. (2023), a mechanism is proposed to infuse the logical structure of STL specifications into gradient-based methods by translating STL robustness formulas into computation graphs. In Lindemann et al. (2021a), the STL formulas are interpreted over discrete-time stochastic processes using the the induced risk. Based on this, risk-aware STL control is studied in Lindemann et al. (2021b).

1.3 Contributions

In this paper, we aim at developing an efficient, robust, and sound control synthesis algorithm for uncertain robotic systems under STL specifications. Different from existing STL synthesis techniques, we propose an approach based on STL, reachability analysis, and temporal logic trees. The new framework is shown in Figure 1. It consists of two phases. In the offline phase, we propose to transform an STL formula into a tube-based temporal logic tree (tTLT) by performing reachability analysis on the dynamic system under consideration. As a fundamental notion in systems and control, reachability captures the evolution of dynamic systems under inputs (e.g., control inputs and uncertainties). In the online phase, the constructed tTLT is further used to guide the control synthesis. The contributions of our paper are as follows: (i) We propose a real-time version of STL semantics and establish a correspondence between STL formulas and tTLTs via reachability analysis. (ii) We develop an algorithm that can automatically and recursively construct the tTLT from the corresponding STL formula. We show that the tTLT is an under-approximation for a broad fragment of STL formulas, *i.e.,* all the trajectories that satisfy the tTLT also satisfy the corresponding STL formula. (iii) We develop an online control synthesis algorithm based on the constructed tTLT. We show that the algorithm is robust and sound. (iv) We validate the effectiveness of the proposed approach by several simulation examples and further demonstrate its practical usability on a hardware experiment.

It is worth mentioning that reachability analysis is the core to ensure the robustness of the proposed algorithm, since the uncertainties in the system can be explicitly addressed when performing reachability analysis. Over the past decades, there have been remarkable progresses in the computation of reachable sets for different systems. New software tools on reachability analysis facilitate the usability of the approach proposed in our paper.

We further remark that the robustness here refers to the control ability against system uncertainties. That is, we are interested in synthesizing a controller under which the signals satisfy an STL specification despite the underlying uncertainties. This is different from the notion of quantitative robustness which measures how much a signal satisfies or violates an STL specification.

1.4 Organization and notations

The remainder of the paper is organized as follows. In Section 2, preliminaries and the problem under consideration are formulated. In Section 3, definitions of real-time STL semantics and tTLTs are introduced. Section 4 establishes a semantic connection between STL and tTLT. Section 5 deals with the online control synthesis problem. The results are

Table 1. Notations

$\mathbb{R}$	set of real numbers
$\mathbb{R}_{\geq 0}$	set of nonnegative real numbers
$\mathbb{N}$	set of natural numbers
$\mathbb{R}^n$	Euclidean space of dimension n
$\mathbb{R}^{n\times m}$	space of n-by-m real matrices
$\|x\|$	Euclidean norm of vector x
x^T	transpose of real vector x
$\emptyset$	empty set
$\overline{S}$	complement of a set S
2^S	set of all subsets of S
$\lvert S\rvert$	cardinality of a set S
$\cup$	set union
$\cap$	set intersection
$\neg$	negation operator
$\wedge$	logical operator AND
$\vee$	logical operator OR
$\subset$ ($\subseteq$)	subset (subset or equivalent to)
$\supset$ ($\supseteq$)	super-set (super-set or equivalent to)
$[a, b]$	closed interval with end points a and b
$S_1 \setminus S_2$	set difference of two sets S_1 and S_2

validated by simulations and experiments in Sections 6 and 7. Conclusions are given in Section 8. The notations used in this work are defined in Table 1.

2 Preliminaries and Problem Formulation

2.1 Systems dynamics

Consider an uncertain discrete-time control system of the form

$$x_{k+1} = f(x_k, u_k, w_k), \tag{1}$$

where $x_k := x(t_k) \in \mathbb{R}^n, u_k := u(t_k) \in U, w_k := w(t_k) \in W, k \in \mathbb{N}$ are the state, control input, and disturbance at time t_k, respectively. The time sequence $\{t_k\}$ can be seen as a sequence of sampling instants, which satisfy $0 = t_0 < t_1 < \cdots$. The control input is constrained to a compact set $U \subset \mathbb{R}^m$ and the disturbance is constrained to a compact set $W \subset \mathbb{R}^l$. In the following, let us define the control policy.

Definition 2.1. *A* control policy *$\boldsymbol{\nu} = \nu_0\nu_1 \ldots \nu_k \ldots$ is a sequence of maps $\nu_k : \mathbb{R}^n \to U, \forall k \in \mathbb{N}$. Denote by $\mathcal{U}_{\geq k}$ the set of all control policies that start from time t_k.*

One can see from Definition 2.1 that a control policy $\boldsymbol{\nu}$ is a sequence of time-dependent functions ν_k, each of which is a map from $\mathbb{R}^n$ (i.e., the state space) to the control set U. Given the control policy ν, one can select control input $u_k = \nu_k(x_k)$ for implementation at time instant t_k.

Definition 2.2. *A disturbance signal $\boldsymbol{w} = w_0w_1 \ldots w_k \ldots$ is called* admissible *if $w_k \in W, \forall k \in \mathbb{N}$. Denote by $\mathcal{W}_{\geq k}$ the set of all admissible disturbance signals that start from time t_k.*

The solution of (1) is defined as a discrete-time signal $\boldsymbol{x} := x_0x_1 \ldots$. We call $\boldsymbol{x}$ a *trajectory* of (1) if there exists a control policy $\boldsymbol{\nu} \in \mathcal{U}_{\geq 0}$ and a disturbance signal $\boldsymbol{w} \in \mathcal{W}_{\geq 0}$ satisfying (1), *i.e.,*

$$x_{k+1} = f(x_k, \nu_k(x_k), w_k), \forall k \in \mathbb{N}.$$

We use $\boldsymbol{x}^{\boldsymbol{\nu},\boldsymbol{w}}_{x_0}(t_k)$ to denote the trajectory point reached at time t_k under the control policy $\boldsymbol{\nu}$ and the disturbance $\boldsymbol{w}$ from initial state x_0.

The deterministic system is defined by

$$x_{k+1} = f_d(x_k, u_k) \tag{2}$$

and $\boldsymbol{x}^{\boldsymbol{\nu}}_{x_0}(t_k)$ denotes the solution at time t_k of the deterministic system when the control policy is $\boldsymbol{\nu}$ and the initial state is x_0.

2.2 Signal temporal logic

We use STL to concisely specify the desired system behavior. STL (Maler and Nickovic, 2004) is a predicate logic consisting of predicates μ, which are defined through a predicate function $g_\mu : \mathbb{R}^n \to \mathbb{R}$ as

$$\mu := \begin{cases} \top, & \text{if} \quad g_\mu(x) \geq 0, \\ \bot, & \text{if} \quad g_\mu(x) < 0. \end{cases}$$

The syntax of STL is given by

$$\varphi ::= \top \mid \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathsf{U}_{\mathrm{I}} \varphi_2, \tag{3}$$

where $\varphi, \varphi_1, \varphi_2$ are STL formulas and I is a closed interval of $\mathbb{R}$ of the form $[a, b]$ with $a, b \in \mathbb{R}_{\geq 0}$ and $a \leq b$.

The validity of an STL formula φ is originally defined with respect to a continuous-time signal (Maler and Nickovic, 2004). Later in Raman et al. (2015), the STL semantics with respect to a discrete-time signal has also been proposed. In this work, we study discrete-time control systems. Therefore, we adopt the STL semantics defined in (Raman et al., 2015).

The validity of an STL formula φ with respect to a discrete-time signal $\boldsymbol{x}$ at time t_k, is defined inductively as follows (Raman et al., 2015):

$$\begin{aligned}
(\boldsymbol{x}, t_k) \vDash \mu \quad &\Leftrightarrow \quad g_\mu(\boldsymbol{x}(t_k)) \geq 0, \\
(\boldsymbol{x}, t_k) \vDash \neg\varphi \quad &\Leftrightarrow \quad \neg((\boldsymbol{x}, t_k) \vDash \varphi), \\
(\boldsymbol{x}, t_k) \vDash \varphi_1 \wedge \varphi_2 \quad &\Leftrightarrow \quad (\boldsymbol{x}, t_k) \vDash \varphi_1 \wedge (\boldsymbol{x}, t_k) \vDash \varphi_2, \\
(\boldsymbol{x}, t_k) \vDash \varphi_1 \mathsf{U}_{[a,b]} \varphi_2 \quad &\Leftrightarrow \quad \exists t_{k'} \in [t_k + a, t_k + b] \text{ s.t.} \\
&\qquad (\boldsymbol{x}, t_{k'}) \vDash \varphi_2 \wedge \forall t_{k''} \in [t_k, t_{k'}], \\
&\qquad (\boldsymbol{x}, t_{k''}) \vDash \varphi_1.
\end{aligned}$$

The signal $\boldsymbol{x} = x_0x_1 \ldots$ satisfies φ, denoted by $\boldsymbol{x} \vDash \varphi$ if $(\boldsymbol{x}, t_0) \vDash \varphi$. By using the "negation" operator $\neg$ and the "conjunction" operator $\wedge$, we can define "disjunction" $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$. And by employing the until operator U_{I}, we can define "eventually" $\mathsf{F}_{\mathrm{I}}\varphi = \top\mathsf{U}_{\mathrm{I}}\varphi$ and "always" $\mathsf{G}_{\mathrm{I}}\varphi = \neg\mathsf{F}_{\mathrm{I}}\neg\varphi$.

Definition 2.3. *(Dokhanchi et al., 2014) The time horizon $\|\varphi\|$ of an STL formula φ is inductively defined as*

$$\|\varphi\| = \begin{cases} 0, & \text{if } \varphi = \mu, \\ \|\varphi_1\|, & \text{if } \varphi = \neg\varphi_1, \\ \max\{\|\varphi_1\|, \|\varphi_2\|\}, & \text{if } \varphi = \varphi_1 \wedge \varphi_2, \\ b + \max\{\|\varphi_1\|, \|\varphi_2\|\}, & \text{if } \varphi = \varphi_1 \mathsf{U}_{[a,b]} \varphi_2. \end{cases}$$

Definition 2.4. *(Robust satisfiability) Consider the uncertain system (1) and the STL formula φ. We say φ is robustly satisfiable from the initial state x_0 if there exists a control policy $\boldsymbol{\nu}$ such that*

$$\boldsymbol{x}^{\boldsymbol{\nu},\boldsymbol{w}}_{x_0} \vDash \varphi, \forall \boldsymbol{w} \in \mathcal{W}_{\geq 0}.$$

Definition 2.5. *(Satisfiability) Consider the deterministic system (2) and the STL formula φ. We say φ is satisfiable from the initial state x_0 if there exists a control policy $\boldsymbol{\nu}$ such that*

$$\boldsymbol{x}_{x_0}^{\boldsymbol{\nu}} \models \varphi.$$

Given an STL formula φ, the set of initial states from which φ is (robustly) satisfiable is denoted by

$$\mathbb{S}_{\varphi} = \{x_0 \in \mathbb{R}^n \mid \varphi \text{ is (robustly) satisfiable from } x_0\}. \quad (4)$$

We remark that the computation of the set $\mathbb{S}_{\varphi}$ is tailored to the dynamic system under consideration. Here we omit it for notation simplicity.

2.3 Reachability operators

In this section, we define two reachability operators. The natural connection between reachability and temporal operators plays an important role in the approach proposed in this paper. The definitions of maximal and minimal reachable tube are given as follows.

Definition 2.6. *Consider system (1), three sets $\Omega_1, \Omega_2, \mathcal{C} \subseteq \mathbb{R}^n$, and a time interval $[a, b]$. The maximal reachable tube from Ω_1 to Ω_2 is defined as*

$$\mathcal{R}^M(\Omega_1, \Omega_2, \mathcal{C}, [a, b], k) = \left\{ x_k \in \Omega_1 \;\middle|\; \begin{array}{l} \exists \boldsymbol{\nu} \in \mathcal{U}_{\geq k}, \forall \boldsymbol{w} \in \mathcal{W}_{\geq k}, \\ \exists t_{k'} \in [\max\{a, t_k\}, b], \boldsymbol{x}_{x_k}^{\boldsymbol{\nu}, \boldsymbol{w}}(t_{k'}) \in \Omega_2, \\ \forall t_{k''} \in [t_k, t_{k'}], \boldsymbol{x}_{x_k}^{\boldsymbol{\nu}, \boldsymbol{w}}(t_{k''}) \in \mathcal{C} \end{array} \right\}$$

and $t_k \in [0, b]$.

The set $\mathcal{R}^M(\Omega_1, \Omega_2, \mathcal{C}, [a, b], k)$ collects all states in Ω_1 at time t_k from which there exists a control policy $\boldsymbol{\nu} \in \mathcal{U}_{\geq k}$ that, despite the worst disturbance signals, drives the system to the target set Ω_2 at some time instant $t_{k'} \in [\max\{a, t_k\}, b]$ while satisfying constraints defined by $\mathcal{C}$ prior to reaching the target.

Definition 2.7. *Consider system (1), two sets $\Omega_1, \Omega_2 \subseteq \mathbb{R}^n$, and a time interval $[a, b]$. The minimal reachable tube from Ω_1 to Ω_2 is defined as*

$$\mathcal{R}^m(\Omega_1, \Omega_2, [a, b], k) = \left\{ x_k \in \Omega_1 \;\middle|\; \begin{array}{l} \forall \boldsymbol{\nu} \in \mathcal{U}_{\geq k}, \exists \boldsymbol{w} \in \mathcal{W}_{\geq k}, \\ \exists t_{k'} \in [\max\{a, t_k\}, b], \boldsymbol{x}_{x_k}^{\boldsymbol{\nu}, \boldsymbol{w}}(t_{k'}) \in \Omega_2 \end{array} \right\},$$

and $t_k \in [0, b]$.

The set $\mathcal{R}^m(\Omega_1, \Omega_2, [a, b], k)$ collects all states in Ω_1 at time t_k from which no matter what control policy $\boldsymbol{\nu}$ is applied, there exists a disturbance signal that drives the system to the target set Ω_2 at some time instant $t_{k'} \in [\max\{a, t_k\}, b]$. In this definition, the constraint set $\mathcal{C}$ is redundant. The reason is that the minimal reachable tube is used to build a connection with "always" operator $\mathsf{G}_{[a,b]}$, for which the constraint set is not needed.

2.4 Problem formulation

Consider the following fragment of STL formulas, which is inductively defined as

$$\varphi ::= \top \mid \mu \mid \neg\mu \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \phi \mathsf{U}_{\mathrm{I}} \varphi \mid \mathsf{F}_{\mathrm{I}} \varphi \mid \mathsf{G}_{\mathrm{I}} \phi, \quad (5)$$

where $\phi ::= \top \mid \mu \mid \neg\mu \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2$. Here, ϕ_1, ϕ_2 are formulas of class ϕ and φ_1, φ_2 are formulas of class φ given in (5).

Remark 2.1. *The STL fragment defined in (5) includes nested STL formulas of the form $\mathsf{F}_{[a_1,b_1]}\mathsf{G}_{[a_2,b_2]}\phi$ and $\phi_1\mathsf{U}_{[a_1,b_1]}(\mathsf{G}_{[a_2,b_2]}\phi_2)$, while excluding nested STL formulas of the form $\mathsf{G}_{[a_1,b_1]}\mathsf{F}_{[a_2,b_2]}\phi$, $(\mathsf{G}_{[a_1,b_1]}\phi_1)\mathsf{U}_{[a_2,b_2]}\phi_2$. The reason is that according to the semantics of STL, nested STL formulas like $\mathsf{G}_{[a_1,b_1]}\mathsf{F}_{[a_2,b_2]}\phi$ and $(\mathsf{G}_{[a_1,b_1]}\phi_1)\mathsf{U}_{[a_2,b_2]}\phi_2$ require parallel monitoring of their arguments $\mathsf{F}_{[a_2,b_2]}\phi$ and $\mathsf{G}_{[a_1,b_1]}\phi_1$ within the encoded time intervals of the temporal operators $\mathsf{G}_{[a_1,b_1]}$ and $\mathsf{U}_{[a_2,b_2]}$, respectively. Nevertheless, we note that the fragment (5) is more general than most of the fragments considered in the literature studying online control synthesis, e.g., Lindemann and Dimarogonas (2018); Buyukkocak et al. (2022). Such a fragment (5) is expressive enough to specify a large number of robotic tasks, e.g., time-constrained reachability, supply-delivery, and safety.*

Remark 2.2. *It is possible to handle nested STL formulas of the form $\mathsf{G}_{[a_1,b_1]}\mathsf{F}_{[a_2,b_2]}\phi$ and $(\mathsf{G}_{[a_1,b_1]}\phi_1)\mathsf{U}_{[a_2,b_2]}\phi_2$ using the framework proposed in this work. For the formula $\mathsf{G}_{[a_1,b_1]}\mathsf{F}_{[a_2,b_2]}\phi$, one can rewrite it as $\mathsf{F}_{[a_1+a_2,b_1+b_2]}\mathsf{G}_{[0,b_1-a_1]}\phi$. For the formula $(\mathsf{G}_{[a_1,b_1]}\phi_1)\mathsf{U}_{[a_2,b_2]}\phi_2$, one can rewrite it as $\mathsf{G}_{[a_1,b_1+b_2]}\phi_1 \wedge \mathsf{F}_{[a_2,b_2]}\phi_2$. Since $\boldsymbol{x} \models \mathsf{F}_{[a_1+a_2,b_1+b_2]}\mathsf{G}_{[0,b_1-a_1]}\phi$ implies $\boldsymbol{x} \models \mathsf{G}_{[a_1,b_1]}\mathsf{F}_{[a_2,b_2]}\phi$ and $\boldsymbol{x} \models \mathsf{G}_{[a_1,b_1+b_2]}\phi_1 \wedge \mathsf{F}_{[a_2,b_2]}\phi_2$ implies $\boldsymbol{x} \models (\mathsf{G}_{[a_1,b_1]}\phi_1)\mathsf{U}_{[a_2,b_2]}\phi_2$, the soundness of the proposed online control synthesis algorithm preserves. However, we note that this approach introduces conservatism due to the fact that the rewritten formula is not equivalent to the original formula.*

The problem under consideration is formulated as follows.

Problem 2.1. Online control synthesis. *Consider system (1) and an STL task φ in (5). For an initial state x_0, find, if it exists, a sequence of control inputs $\boldsymbol{\nu} = \nu_0(x_0)\nu_1(x_1)\ldots\nu_k(x_k)\ldots$ such that the resulting trajectory $\boldsymbol{x} = x_0x_1\ldots x_k\ldots$ satisfies φ.*

Remark 2.3. *Note that the objective of Problem 2.1 is not to synthesize a closed-form control policy $\boldsymbol{\nu}$, which is in general computationally intractable for systems with continuous spaces. Instead, we aim at finding online a sequence of feedback control inputs in a way that is similar to receding horizon control.*

The key idea to solve Problem 2.1 is as follows. We first transform the STL formula to an alternative tree-based representation, which we call a tube-based temporal logic tree (tTLT), by leveraging reachability analysis as detailed in Section 3. There exists a semantic connection between the STL formula and the corresponding tTLT, thanks to the reachability analysis, which is explained in Section 4. Based on this fact, we can perform control synthesis over the tTLT, instead of the STL formula. An online control synthesis algorithm is provided in Section 5.

3 Real-time STL semantics and tube-based temporal logic tree

In this section, a real-time version of STL semantics and a notion of tTLT are proposed. The real-time STL semantics

$$(\boldsymbol{s}, t_k, t_l) \,|\!\asymp \mu \quad \Leftrightarrow \quad g_\mu(\boldsymbol{s}(t_k)) \geq 0, \quad t_l = t_k; \tag{6a}$$

$$(\boldsymbol{s}, t_k, t_l) \,|\!\asymp \neg\varphi \quad \Leftrightarrow \quad \neg((\boldsymbol{s}, t_k, t_l) \,|\!\asymp \varphi), \quad t_l \in t_k + [0, ||\varphi||]; \tag{6b}$$

$$(\boldsymbol{s}, t_k, t_l) \,|\!\asymp \varphi_1 \wedge \varphi_2 \quad \Leftrightarrow \quad (\boldsymbol{s}, t_k, t_l) \,|\!\asymp \varphi_1 \wedge (\boldsymbol{s}, t_k, t_l) \,|\!\asymp \varphi_2, \quad t_l \in t_k + [0, ||\varphi_1 \wedge \varphi_2||]; \tag{6c}$$

$$(\boldsymbol{s}, t_k, t_l) \,|\!\asymp \varphi_1 \mathsf{U}_{[a,b]} \varphi_2 \quad \Leftrightarrow$$

$$\begin{cases} \exists t_{k'} \in [\max\{t_k + a, t_l\}, t_k + b] \text{ s.t. } (\boldsymbol{s}, t_{k'}, t_l) \,|\!\asymp \varphi_2, & \text{if } \|\varphi_2\| = 0, \\ \exists t_{k'} \in [t_k + a, t_k + b] \text{ s.t. } (\boldsymbol{s}, t_{k'}, t_l) \,|\!\asymp \varphi_2, & \text{otherwise}, \end{cases}$$

$$\wedge \text{ if } t_l \leq t_{k'}, \forall t_{k''} \in [t_l, t_{k'}], (\boldsymbol{s}, t_{k''}, t_l) \,|\!\asymp \varphi_1, \quad t_l \in t_k + [0, \|\varphi_1 \mathsf{U}_{[a,b]} \varphi_2\|]. \tag{6d}$$

establish the satisfaction relation between a real-time signal and the STL formula. Based on these real-time semantics, we propose the tTLT using the close connection between STL and reachability analysis.

3.1 Real-time STL semantics

The real-time STL semantics is defined to capture the satisfaction relation between a real-time signal and an STL formula, which is different from the traditional STL semantics defined in Section 2.2. Before proceeding, the following definition is required.

Definition 3.1. Suffix and Completions. *Given a discrete-time signal $\boldsymbol{x} = x_0 x_1 \ldots$, we say that a partial signal $\boldsymbol{s} = s_l s_{l+1} \ldots, l \in \mathbb{N}$, is a* suffix *of the signal $\boldsymbol{x}$ if $\forall k' \geq l, s_{k'} = x_{k'}$. The set of* completions *of a partial signal $\boldsymbol{s}$, denoted by $C(\boldsymbol{s})$, is given by*

$$C(\boldsymbol{s}) := \{\boldsymbol{x} : \boldsymbol{s} \textit{ is a suffix of } \boldsymbol{x}\}.$$

Given a time instant t_k and a time interval $[a, b]$, define $t_k + [a, b] := [t_k + a, t_k + b]$. The real-time STL semantics is defined as follows.

Definition 3.2. *Let t_k be the starting time of any STL formula φ to be evaluated. Let $t_l \geq t_k$ be the starting time of a partial signal $\boldsymbol{s} = s_l s_{l+1} \ldots$. The real-time satisfaction of φ with respect to the partial signal $\boldsymbol{s}$, denoted by $(\boldsymbol{s}, t_k, t_l) \,|\!\asymp \varphi$, is recursively defined by Eq. (6).*

The real-time satisfaction relation $(\boldsymbol{s}, t_k, t_l) \,|\!\asymp \varphi$ suggests that the partial signal $\boldsymbol{s}$ is the suffix of a satisfying trajectory that starts from t_k, *i.e.,*

$$(\boldsymbol{s}, t_k, t_l) \,|\!\asymp \varphi \Leftarrow \exists \boldsymbol{x} \in C(\boldsymbol{s}), (\boldsymbol{x}, t_k) \vDash \varphi.$$

Using the induction rule, one can define the real-time STL semantics for "disjunction" $\varphi_1 \vee \varphi_2$, "eventually" $\mathsf{F}_{[a,b]}\varphi$, and "always" $\mathsf{G}_{[a,b]}\varphi$.

In parallel with Definitions 2.4 and 2.5, we define the STL satisfiability given a partial signal as follows.

Definition 3.3. *Consider uncertain system (1) and the STL formula φ. We say φ is robustly satisfiable from the state x_k at time t_k if there exists a control policy $\boldsymbol{\nu} \in \mathcal{U}_{\geq k}$ such that*

$$(\boldsymbol{x}_{x_k}^{\boldsymbol{\nu}, \boldsymbol{w}}, t_0, t_k) \,|\!\asymp \varphi, \forall \boldsymbol{w} \in \mathcal{W}_{\geq k}.$$

Definition 3.4. *Consider the deterministic system (2) and the STL formula φ. We say φ is satisfiable from the state x_k at time t_k if there exists a control policy $\boldsymbol{\nu} \in \mathcal{U}_{\geq k}$ such that*

$$(\boldsymbol{x}_{x_k}^{\boldsymbol{\nu}}, t_0, t_k) \,|\!\asymp \varphi.$$

Note that when $t_k = t_0$, Definitions 3.3 and 3.4 reduce to Definitions 2.4 and 2.5, respectively. Given an STL formula φ, the set of states from which φ is robustly satisfiable at t_k is denoted by

$$\mathbb{S}_\varphi(t_k) := \{x_k \in \mathbb{R}^n \mid \varphi \text{ is (robustly) satisfiable from } x_k \text{ at } t_k\}. \tag{7}$$

Then, we have the following results.

Proposition 3.1. *Consider system (1) and predicates μ_1, μ_2. Then, one has*

i) $\mathbb{S}_{\mu_1 \mathsf{U}_{[a,b]} \mu_2}(t_k) = \mathcal{R}^M(\mathbb{R}^n, \mathbb{S}_{\mu_2}, \mathbb{S}_{\mu_1}, [a, b], k)$*;*

ii) $\mathbb{S}_{\mathsf{F}_{[a,b]} \mu_1}(t_k) = \mathcal{R}^M(\mathbb{R}^n, \mathbb{S}_{\mu_1}, \mathbb{R}^n, [a, b], k)$*;*

iii) $\mathbb{S}_{\mathsf{G}_{[a,b]} \mu_1}(t_k) = \overline{\mathcal{R}^m(\mathbb{R}^n, \overline{\mathbb{S}_{\mu_1}}, [a, b], k)}$*,*

where $\mathbb{S}_{\mu_1}$ and $\mathbb{S}_{\mu_2}$ are defined in (4).

Proof. First, we prove item *i)*. Assume that $x_k \in \mathbb{S}_{\mu_1 \mathsf{U}_{[a,b]} \mu_2}(t_k)$. According to the real-time STL semantics, one has that $\exists \boldsymbol{\nu} \in \mathcal{U}_{\geq k}, \forall \boldsymbol{w} \in \mathcal{W}_{\geq k}$ such that

- $\exists t_{k'} \in [\max\{a, t_k\}, b]$,
$$(\boldsymbol{x}_{x_k}^{\boldsymbol{\nu}, \boldsymbol{w}}, t_{k'}, t_k) \,|\!\asymp \mu_2 \Rightarrow \boldsymbol{x}_{x_k}^{\boldsymbol{\nu}, \boldsymbol{w}}(t_{k'}) \in \mathbb{S}_{\mu_2},$$
- $\forall t_{k''} \in [t_k, t_{k'}]$,
$$(\boldsymbol{x}_{x_k}^{\boldsymbol{\nu}, \boldsymbol{w}}, t_{k''}, t_k) \,|\!\asymp \mu_1 \Rightarrow \boldsymbol{x}_{x_k}^{\boldsymbol{\nu}, \boldsymbol{w}}(t_{k''}) \in \mathbb{S}_{\mu_1}.$$

That is, $x_k \in \mathcal{R}^M(\mathbb{R}^n, \mathbb{S}_{\mu_2}, \mathbb{S}_{\varphi_1}, [a, b], k)$.

Now we assume that $x_k \in \mathcal{R}^M(\mathbb{R}^n, \mathbb{S}_{\mu_2}, \mathbb{S}_{\mu_1}, [a, b], k)$. According to Definition 2.6, one has that $\exists \boldsymbol{\nu} \in \mathcal{U}_{\geq k}, \forall \boldsymbol{w} \in \mathcal{W}_{\geq k}$ such that $\exists t_{k'} \in [\max\{a, t_k\}, b], \boldsymbol{x}_{x_k}^{\boldsymbol{\nu}, \boldsymbol{w}}(t_{k'}) \in \mathbb{S}_{\mu_2}$ and $\forall t_{k''} \in [t_k, t_{k'}], \boldsymbol{x}_{x_k}^{\boldsymbol{\nu}, \boldsymbol{w}}(t_{k''}) \in \mathbb{S}_{\mu_1}$. According to (7), one can further get that

$$\boldsymbol{x}_{x_k}^{\boldsymbol{\nu}, \boldsymbol{w}}(t_{k'}) \in \mathbb{S}_{\mu_1} \Rightarrow (\boldsymbol{x}_{x_k}^{\boldsymbol{\nu}, \boldsymbol{w}}, t_{k'}, t_k) \,|\!\asymp \mu_1,$$
$$\boldsymbol{x}_{x_0}^{\boldsymbol{\nu}, \boldsymbol{w}}(t_{k''}) \in \mathbb{S}_{\mu_2} \Rightarrow (\boldsymbol{x}_{x_k}^{\boldsymbol{\nu}, \boldsymbol{w}}, t_{k''}, t_k) \,|\!\asymp \mu_2.$$

Therefore, $x_k \in \mathbb{S}_{\mu_1 \mathsf{U}_{[a,b]} \mu_2}(t_k)$.

The proof of item *ii)* is similar and hence omitted. Next, let us prove item *iii)*.

Assume that $x_k \in \mathbb{S}_{\mathsf{G}_{[a,b]} \mu_1}(t_k)$. According to the real-time STL semantics, one has that $\exists \boldsymbol{\nu} \in \mathcal{U}_{\geq k}, \forall \boldsymbol{w} \in \mathcal{W}_{\geq k}$ such that $\forall t_{k'} \in [\max\{a, t_k\}, b]$,

$$(\boldsymbol{x}_{x_k}^{\boldsymbol{\nu}, \boldsymbol{w}}, t_{k'}, t_k) \,|\!\asymp \mu_1 \Rightarrow \boldsymbol{x}_{x_k}^{\boldsymbol{\nu}, \boldsymbol{w}}(t_{k''}) \in \mathbb{S}_{\mu_1}.$$

According to Definition 2.7, $\mathcal{R}^m(\mathbb{R}^n, \overline{\mathbb{S}_{\mu_1}}, [a, b], k)$ collects all states in $\mathbb{R}^n$ at time t_k from which no matter what control policy $\boldsymbol{\nu} \in \mathcal{U}_{\geq k}$ is applied, there exists a disturbance signal $\boldsymbol{w} \in \mathcal{W}_{\geq k}$ that drives the system to the set $\overline{\mathbb{S}_{\mu_1}}$ at some time instant $t_{k'} \in [\max\{a, t_k\}, b]$. Therefore, $x_k \in \mathcal{R}^m(\mathbb{R}^n, \overline{\mathbb{S}_{\mu_1}}, [a, b], k)$. The other side can be proved similarly. □

In item *iii)* of Proposition 3.1, the use of the complementary set is motivated by the fact that $\mathsf{G}_{[a,b]}\mu = \neg\mathsf{F}_{[a,b]}(\neg\mu)$ and $\mathbb{S}_{\neg\mu} = \overline{\mathbb{S}_{\mu}}$.

Proposition 3.2. *Consider system (1) and STL formulas φ_1, φ_2. If φ_1 and φ_2 contain no logical operators $\wedge$ and $\vee$, then one has*

i) $\mathbb{S}_{\varphi_1 \wedge \varphi_2}(t_k) \subseteq \mathbb{S}_{\varphi_1}(t_k) \cap \mathbb{S}_{\varphi_2}(t_k)$;
ii) $\mathbb{S}_{\varphi_1 \vee \varphi_2}(t_k) \supseteq \mathbb{S}_{\varphi_1}(t_k) \cup \mathbb{S}_{\varphi_2}(t_k)$;

where $\mathbb{S}_{\varphi_1}(t_k)$ and $\mathbb{S}_{\varphi_2}(t_k)$ are defined in (7).

Proof. Assume that $x_k \in \mathbb{S}_{\varphi_1 \wedge \varphi_2}(t_k)$. According to Definition 3.2 and (7), one has that there exists a control policy $\boldsymbol{\nu} \in \mathcal{U}_{\geq k}$ such that

$$\begin{aligned}&(\boldsymbol{x}_{x_k}^{\boldsymbol{\nu},\boldsymbol{w}}, t_0, t_k) \mid\!\asymp \varphi_1, \forall \boldsymbol{w} \in \mathcal{W}_{\geq k} \\ &\wedge (\boldsymbol{x}_{x_k}^{\boldsymbol{\nu},\boldsymbol{w}}, t_0, t_k) \mid\!\asymp \varphi_2, \forall \boldsymbol{w} \in \mathcal{W}_{\geq k}.\end{aligned}$$

That is, $x_k \in \mathbb{S}_{\varphi_1}(t_k), x_k \in \mathbb{S}_{\varphi_2}(t_k)$. Thus, $x_k \in \mathbb{S}_{\varphi_1 \wedge \varphi_2}(t_k) \Rightarrow x_k \in \mathbb{S}_{\varphi_1}(t_k) \cap \mathbb{S}_{\varphi_2}(t_k)$. The other direction may not hold because it could happen that for a state x_k, there exist two control policies $\boldsymbol{\nu}_1, \boldsymbol{\nu}_2 \in \mathcal{U}_{\geq k}$ such that $(\boldsymbol{x}_{x_k}^{\boldsymbol{\nu}_1,\boldsymbol{w}}, t_0, t_k) \mid\!\asymp \varphi_1, (\boldsymbol{x}_{x_k}^{\boldsymbol{\nu}_2,\boldsymbol{w}}, t_0, t_k) \mid\!\asymp \varphi_2, \forall \boldsymbol{w} \in \mathcal{W}_{\geq k}$ (*i.e.*, $x_k \in \mathbb{S}_{\varphi_1}(t_k) \cap \mathbb{S}_{\varphi_2}(t_k)$). However, there is no control policy which ensures the robust satisfaction of $\varphi_1 \wedge \varphi_2$ at t_k.

Assume now that $x_k \in \mathbb{S}_{\varphi_1}(t_k)$, then one has that there exists a control policy $\boldsymbol{\nu} \in \mathcal{U}_{\geq k}$ such that $(\boldsymbol{x}_{x_k}^{\boldsymbol{\nu},\boldsymbol{w}}, t_0, t_k) \mid\!\asymp \varphi_1, \forall \boldsymbol{w} \in \mathcal{W}_{\geq k}$. Moreover, according to STL syntax, one further has $(\boldsymbol{x}_{x_k}^{\boldsymbol{\nu},\boldsymbol{w}}, t_0, t_k) \mid\!\asymp \varphi_1 \vee \varphi_2, \forall \boldsymbol{w} \in \mathcal{W}_{\geq k}$. That is, $x_k \in \mathbb{S}_{\varphi_1}(t_k) \Rightarrow x_k \in \mathbb{S}_{\varphi_1 \vee \varphi_2}(t_k)$. Similarly, one can also get $x_k \in \mathbb{S}_{\varphi_2}(t_k) \Rightarrow x_k \in \mathbb{S}_{\varphi_1 \vee \varphi_2}(t_k)$. Therefore, $x_k \in \mathbb{S}_{\varphi_1}(t_k) \cup \mathbb{S}_{\varphi_2}(t_k) \Rightarrow x_k \in \mathbb{S}_{\varphi_1 \vee \varphi_2}(t_k)$. The other direction may not hold because it could happen that there exists no state such that either φ_1 or φ_2 is robustly satisfiable from at t_k, *i.e.*, $\mathbb{S}_{\varphi_1}(t_k) = \emptyset$, $\mathbb{S}_{\varphi_2}(t_k) = \emptyset$ and thus $\mathbb{S}_{\varphi_1}(t_k) \cup \mathbb{S}_{\varphi_2}(t_k) = \emptyset$. However, there exists a state x_k^* from which there exists a control policy $\boldsymbol{\nu} \in \mathcal{U}_{\geq k}$ such that

$$\begin{aligned}&(\boldsymbol{x}_{x_k^*}^{\boldsymbol{\nu},\boldsymbol{w}_1}, t_0, t_k) \mid\!\asymp \varphi_1, \forall \boldsymbol{w}_1 \in \mathcal{W}_1 \\ &\wedge (\boldsymbol{x}_{x_k^*}^{\boldsymbol{\nu},\boldsymbol{w}_2}, t_0, t_k) \mid\!\asymp \varphi_2, \forall \boldsymbol{w}_2 \in \mathcal{W}_{\geq k} \setminus \mathcal{W}_1,\end{aligned}$$

where $\mathcal{W}_1 \subset \mathcal{W}_{\geq k}$. In this case, one has $x_k^* \in \mathbb{S}_{\varphi_1 \vee \varphi_2}(t_k)$. □

Propositions 3.1 and 3.2 imply that the real-time satisfiable set of the STL formula can be inferred by set operations and reachability analysis, which makes it reasonable to develop the tTLT, a tree structure consisting of reachable tubes and operators. In the following section, we will detail the definition of tTLT and how to construct a tTLT from a given STL formula using reachability analysis.

3.2 Tube-based temporal logic tree and its construction

In this section, we formally introduce the notion of tTLT and provide its construction algorithm. A tTLT is a variant of the TLT proposed in the recent work (Gao et al., 2022) for LTL formulas. Due to the time-dependent essence of STL formulas, the reachable sets in the TLT are replaced with the reachable tubes in the tTLT, which can explicitly incorporate the time constraints in STL formulas. The intuition of the tTLT is that it indicates how a state trajectory should evolve in order to satisfy the time constraints embedded in an STL formula. In the following, a formal definition of the tTLT is introduced.

Definition 3.5. *A* tTLT *is a tree for which the following holds:*

- *each node is either a* tube *node that maps from the nonnegative time axis, i.e., $\mathbb{R}_{\geq 0}$, to a subset of $\mathbb{R}^n$, or an* operator *node that belongs to $\{\wedge, \vee, \mathsf{U}_I, \mathsf{F}_I, \mathsf{G}_I\}$;*
- *the root node and the leaf nodes are* tube *nodes;*
- *if a* tube *node is not a leaf node, its unique child is an* operator *node;*
- *the children of any* operator *node are* tube *nodes.*

Definition 3.6. *A* complete path $\boldsymbol{p}$ *of a tTLT is a path that starts from the root node and ends at a leaf node.*

The following result shows how to construct a tTLT for any given STL formula using reachability analysis.

Theorem 3.1. *For system (1) and every STL formula φ in (3), a tTLT, denoted by $\mathcal{T}_\varphi$, can be constructed from φ through the reachability operators $\mathcal{R}^M$ and $\mathcal{R}^m$.*

Proof. We follow three steps to construct a tTLT.

Step 1: Rewrite the STL formula φ into the equivalent positive normal form (PNF). It has been proven in Sadraddini and Belta (2015) that each STL formula has an equivalent STL formula in PNF (*i.e.*, negations only occur adjacent to predicates), which can be inductively defined as

$$\begin{aligned}\varphi ::= \top \mid \mu \mid \neg\mu \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \\ \mid \varphi_1 \mathsf{U}_{\mathrm{I}} \varphi_2 \mid \mathsf{F}_{\mathrm{I}}\varphi_1 \mid \mathsf{G}_{\mathrm{I}}\varphi_1.\end{aligned}$$

Step 2: For each predicate μ or its negation $\neg\mu$, construct the tTLT with only one tube node $\mathbb{S}_\mu = \{x : g_\mu(x) \geq 0\}$ or $\overline{\mathbb{S}_\mu}$. The tTLT of $\top$ or $\perp$ has only one tube node, which is $\mathbb{R}^n$ or $\emptyset$.

Step 3: Following the induction rule to construct the tTLT $\mathcal{T}_\varphi$. More specifically, we will show that given STL formulas φ_1 and φ_2, if the tTLTs can be constructed from φ_1 and φ_2, then the tTLTs can be constructed from $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \mathsf{U}_{[a,b]} \varphi_2$, $\mathsf{F}_{[a,b]}\varphi_1$, and $\mathsf{G}_{[a,b]}\varphi_1$.

Case 1: Boolean operators $\wedge$ and $\vee$. Consider two STL formulas φ_1, φ_2 and their corresponding tTLTs $\mathcal{T}_{\varphi_1}, \mathcal{T}_{\varphi_2}$. The root nodes of $\mathcal{T}_{\varphi_1}$ and $\mathcal{T}_{\varphi_2}$ are denoted by $\mathbb{X}_{\varphi_1}(t_k)$ and $\mathbb{X}_{\varphi_2}(t_k)$, respectively. The tTLT $\mathcal{T}_{\varphi_1 \wedge \varphi_2}$ ($\mathcal{T}_{\varphi_1 \vee \varphi_2}$) can be constructed by connecting $\mathbb{X}_{\varphi_1}(t_k)$ and $\mathbb{X}_{\varphi_2}(t_k)$ through the operator node $\wedge$ ($\vee$) and taking the intersection (or union) of the two root nodes, *i.e.*, $\mathbb{X}_{\varphi_1}(t_k) \cap \mathbb{X}_{\varphi_2}(t_k)$ ($\mathbb{X}_{\varphi_1}(t_k) \cup \mathbb{X}_{\varphi_2}(t_k)$), to be the root node. An illustrative diagram for $\varphi_1 \wedge \varphi_2$ is given in Figure 2.

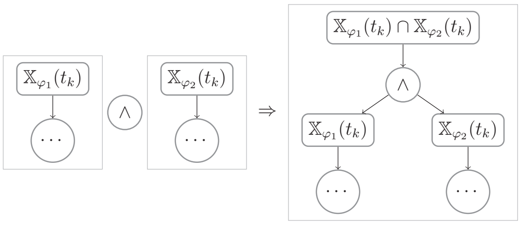

$\mathbb{X}_{\varphi_1}(t_k)$
$\wedge$
$\mathbb{X}_{\varphi_2}(t_k)$
$\Rightarrow$
$\mathbb{X}_{\varphi_1}(t_k) \cap \mathbb{X}_{\varphi_2}(t_k)$
$\wedge$
$\mathbb{X}_{\varphi_1}(t_k)$
$\mathbb{X}_{\varphi_2}(t_k)$
$\cdots$
$\cdots$
$\cdots$
$\cdots$

Figure 2. Illustrative diagram of construction tTLT for $\varphi_1 \wedge \varphi_2$.

Case 2: Until operator $\mathsf{U}_{[a,b]}$. Consider two STL formulas φ_1, φ_2 and their corresponding tTLTs $\mathcal{T}_{\varphi_1}, \mathcal{T}_{\varphi_2}$. The root nodes of $\mathcal{T}_{\varphi_1}$ and $\mathcal{T}_{\varphi_2}$ are denoted by $\mathbb{X}_{\varphi_1}(t_k)$ and $\mathbb{X}_{\varphi_2}(t_k)$, respectively. In addition, the leaf nodes of $\mathcal{T}_{\varphi_1}$ are denoted by $\mathbb{Y}^1_{\varphi_1}(t_k), \cdots, \mathbb{Y}^N_{\varphi_1}(t_k)$, where N is the total number of leaf nodes of $\mathcal{T}_{\varphi_1}$. The tTLT $\mathcal{T}_{\varphi_1 \mathsf{U}_{[a,b]} \varphi_2}$ can be constructed by the following steps: 1) replace each leaf node $\mathbb{Y}^i_{\varphi_1}(t_k)$ by $\mathcal{R}^M(\mathbb{R}^n, \mathbb{X}_{\varphi_2}(t_0), \mathbb{Y}^i_{\varphi_1}(t_0), [a,b], k)$; 2) update $\mathcal{T}_{\varphi_1}$ from the leaf nodes to the root node with the new leaf nodes; and 3) connect each leaf node of the updated $\mathcal{T}_{\varphi_1}$ and the root node of $\mathcal{T}_{\varphi_2}$, *i.e.,* $\mathbb{X}_{\varphi_2}(t_k)$, with the operator node $\mathsf{U}_{[a,b]}$. One illustrative diagram for $\mathsf{U}_{[a,b]}$ is given in Figure 3.

Case 3: Eventually and always operators $\mathsf{F}_{[a,b]}$ and $\mathsf{G}_{[a,b]}$. Consider an STL formula φ_1 and its corresponding tTLT $\mathcal{T}_{\varphi_1}$. The root node of $\mathcal{T}_{\varphi_1}$ is given by $\mathbb{X}_{\varphi_1}(t_k)$. The tTLT $\mathcal{T}_{\mathsf{F}_{[a,b]}\varphi_1}$ ($\mathcal{T}_{\mathsf{G}_{[a,b]}\varphi_1}$) can be constructed by connecting $\mathbb{X}_{\varphi_1}(t_k)$ through the operator $\mathsf{F}_{[a,b]}$ ($\mathsf{G}_{[a,b]}$) and making the tube $\mathcal{R}^M(\mathbb{R}^n, \mathbb{X}_{\varphi_1}(t_0), \mathbb{R}^n, [a,b], k)$ $\overline{(\mathcal{R}^m(\mathbb{R}^n, \overline{\mathbb{X}_{\varphi_1}(t_0)}, [a,b], k)}$) the root node. An illustrative diagram for $\mathsf{G}_{[a,b]}$ is given in Figure 4. □

Based on Theorem 3.1, Algorithm 1 is designed for the construction of tTLT $\mathcal{T}_\varphi$. It takes the syntax tree of the STL formula φ as input. For an STL formula, the nodes of its syntax tree are either predicate or operator nodes. More specifically, all the leaf nodes are predicates and all other nodes are operators.

Algorithm 1 *tTLTConstruction*

```
Input: the syntax tree of STL formula $\varphi$.
Return: the tTLT $\mathcal{T}_\varphi$.
for each leaf node $\mu$ (or $\neg\mu$) of the syntax tree do,
    Replace $\mu$ (or $\neg\mu$) by $\mathbb{S}_\mu$ (or $\mathbb{S}_{\neg\mu}$),
end for
for each operator node of the syntax tree through a bottom-up traversal, do
    Construct $\mathcal{T}_\varphi$ according to Theorem 3.1,
end for
```

Let us use the following example to show how to construct the tTLT.

Example 3.1. *Consider the formula* $\varphi = \mathsf{F}_{[a_1,b_1]}\mathsf{G}_{[a_2,b_2]}\mu_1 \wedge \mu_2 \mathsf{U}_{[a_3,b_3]}\mu_3$, *where* $\mu_i, i = \{1,2,3\}$ *are predicates. The syntax tree of* φ *is shown on the left-hand side of Figure 5. The corresponding tTLT for* φ *(constructed using Algorithm 1) is shown on the right-hand side of Figure 5, where*

$$\mathbb{X}_4(t_k) = \overline{\mathcal{R}^m(\mathbb{R}^n, \overline{\mathbb{S}_{\mu_1}}, [a_2,b_2], k)},$$
$$\mathbb{X}_3(t_k) = \mathcal{R}^M(\mathbb{R}^n, \mathbb{S}_{\mu_3}, \mathbb{S}_{\mu_2}, [a_3,b_3], k),$$
$$\mathbb{X}_2(t_k) = \mathcal{R}^M(\mathbb{R}^n, \mathbb{X}_4(t_0), \mathbb{R}^n, [a_1,b_1], k),$$
$$\mathbb{X}_1(t_k) = \mathbb{X}_2(t_k) \cap \mathbb{X}_3(t_k).$$

Remark 3.1. *Given an STL formula* φ *in PNF, let* K *denote the number of Boolean operators and* L *the number of temporal operators contained in* φ. *Let* $\mathcal{T}_\varphi$ *be the tTLT corresponding to* φ. *Then,* $\mathcal{T}_\varphi$ *has at most* $2K$ *complete paths. In addition, each complete path has at most* $2(K+L)+1$ *nodes, out of which at most* $K+L$ *are non-root tube nodes. Thus, one can conclude that* $\mathcal{T}_\varphi$ *contains at most* $4K(K+L)+1$ *nodes, out of which at most* $2K(K+L)+1$ *are tube nodes.*

4 Semantic Connection between STL and tTLT

In this section, the semantic connection between an STL formula and its corresponding tTLT is derived. We define how a given state trajectory satisfies a tTLT and then show that the tTLT is a semantic under-approximation of the STL formula. Before that, let us first define the *segment* of the complete path.

Definition 4.1. *A complete path of a tTLT can be encoded in the form of* $\boldsymbol{p} = \mathbb{X}_0\Theta_1\mathbb{X}_1\Theta_2\ldots\Theta_{N_f}\mathbb{X}_{N_f}$, *where* N_f *is the number of operator nodes contained in the complete path,* $\mathbb{X}_i : \mathbb{R}_{\geq 0} \to 2^{\mathbb{R}^n}, \forall i \in \{0,1,\ldots,N_f\}$ *represent tube nodes, and* $\Theta_j \in \{\wedge, \vee, \mathsf{U}_I, \mathsf{F}_I, \mathsf{G}_I\}, \forall j \in \{1,\ldots,N_f\}$ *represent operator nodes. Any subsequence of a complete path is called a* segment *of the complete path.*

Now, we define the *maximal temporal segment* for a tTLT, which plays an important role when simplifying the tTLT.

Definition 4.2. *A* maximal temporal segment (MTS) *of a complete path of the tTLT is one of the following types of segment:*

1) *a segment from the root node to the parent of the first Boolean operator node (*$\wedge$ *or* $\vee$*);*
2) *a segment from one child of one Boolean operator node to the parent of the next Boolean operator node;*
3) *a segment from one child of the last Boolean operator node to the leaf node.*

One can conclude from Definition 4.2 that any MTS starts and ends with a tube node and contains no Boolean operator nodes.

Definition 4.3. *A* time coding *of (a complete path of) the tTLT is an assignment of each tube node* $\mathbb{X}_i$ *of (the complete path of) the tTLT an activation time instant* $t_{\kappa_i}, \kappa_i \in \mathbb{N}$.

Now, we further define the satisfaction relation between a trajectory $\boldsymbol{x}$ and a complete path of the tTLT.

Definition 4.4. *Consider a trajectory* $\boldsymbol{x} := x_0x_1\ldots$ *and a complete path* $\boldsymbol{p} = \mathbb{X}_0\Theta_1\mathbb{X}_1\Theta_2\ldots\Theta_{N_f}\mathbb{X}_{N_f}$. *We say* $\boldsymbol{x}$ satisfies $\boldsymbol{p}$, *denoted by* $\boldsymbol{x} \vDash \boldsymbol{p}$, *if there exists a time coding for* $\boldsymbol{p}$ *such that*

i) *if* $\Theta_i \in \{\wedge, \vee\}$, *then* $t_{\kappa_i} = t_{\kappa_{i-1}}$;
ii) *if* $\Theta_i = \mathsf{U}_I$, *then* $t_{\kappa_i} \in t_{\kappa_{i-1}} + I$;
iii) *if* $\Theta_i = \mathsf{G}_I$, *then* $t_{\kappa_i} = \mathrm{argmax}_{t_k}\{t_k \in t_{\kappa_{i-1}} + I\}$;

and

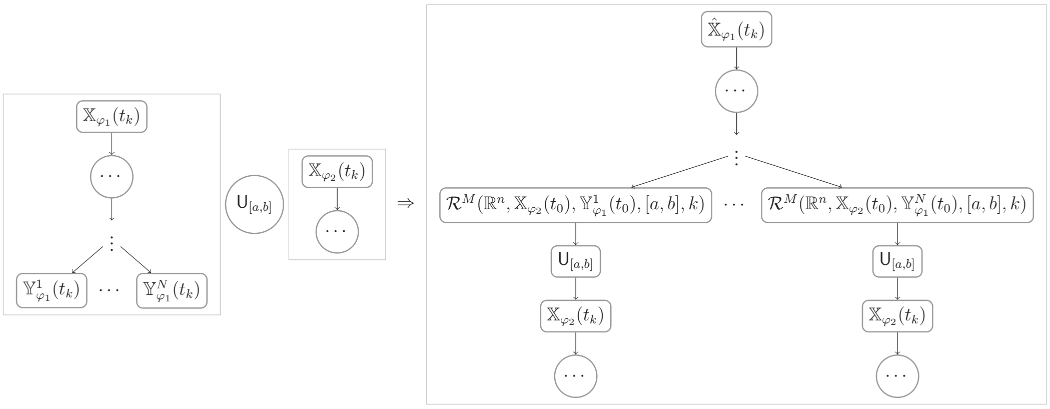

$\mathbb{X}_{\varphi_1}(t_k)$
$\cdots$
$\mathbb{Y}^1_{\varphi_1}(t_k)$
$\cdots$
$\mathbb{Y}^N_{\varphi_1}(t_k)$
$\mathsf{U}_{[a,b]}$
$\mathbb{X}_{\varphi_2}(t_k)$
$\cdots$
$\Rightarrow$
$\hat{\mathbb{X}}_{\varphi_1}(t_k)$
$\cdots$
$\mathcal{R}^M(\mathbb{R}^n, \mathbb{X}_{\varphi_2}(t_0), \mathbb{Y}^1_{\varphi_1}(t_0), [a,b], k)$
$\cdots$
$\mathcal{R}^M(\mathbb{R}^n, \mathbb{X}_{\varphi_2}(t_0), \mathbb{Y}^N_{\varphi_1}(t_0), [a,b], k)$
$\mathsf{U}_{[a,b]}$
$\mathsf{U}_{[a,b]}$
$\mathbb{X}_{\varphi_2}(t_k)$
$\mathbb{X}_{\varphi_2}(t_k)$
$\cdots$
$\cdots$

Figure 3. Illustrative diagram of construction tTLT for $\varphi_1 \mathsf{U}_{[a,b]} \varphi_2$.

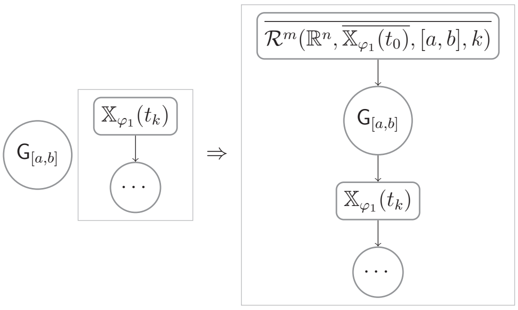

$\mathsf{G}_{[a,b]}$
$\mathbb{X}_{\varphi_1}(t_k)$
$\cdots$
$\Rightarrow$
$\overline{\mathcal{R}^m(\mathbb{R}^n, \overline{\mathbb{X}_{\varphi_1}(t_0)}, [a,b], k)}$
$\mathsf{G}_{[a,b]}$
$\mathbb{X}_{\varphi_1}(t_k)$
$\cdots$

Figure 4. Illustrative diagram of construction tTLT for $\mathsf{G}_{[a,b]}\varphi_1$.

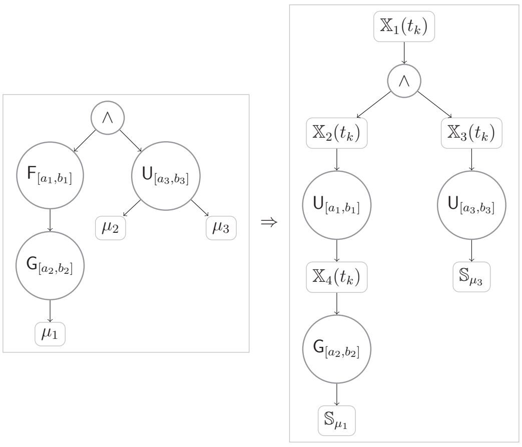


Figure 5. Example 3.1: syntax tree (left) and tTLT (right) for $\varphi = \mathsf{F}_{[a_1,b_1]}\mathsf{G}_{[a_2,b_2]}\mu_1 \wedge \mu_2 \mathsf{U}_{[a_3,b_3]}\mu_3$. Recall that $\mathsf{F}_{[a,b]}\varphi = \top \mathsf{U}_{[a,b]}\varphi$.

iv) $x_k \in \mathbb{X}_i(t_{k-\kappa_i}), \forall k \in [\kappa_i, \kappa_{i+1}], i = 0, \ldots, N_f - 1$;
v) $x_{\kappa_{N_f}} \in \mathbb{X}_{N_f}(t_0)$.

Remark 4.1. *From items i)-iii) of Definition 4.4, one has that $t_{\kappa_0} \leq t_{\kappa_1} \leq \cdots \leq t_{\kappa_{N_f}}$. This means that if a trajectory $\boldsymbol{x} \vDash \boldsymbol{p}$, it must visit each tube node $\mathbb{X}_i$ of the complete path $\boldsymbol{p}$ sequentially. In addition, we can further conclude from items iv)-v) that the trajectory $\boldsymbol{x}$ has to stay in each tube node $\mathbb{X}_i$ for sufficiently long time steps.*

With Definition 4.4, the satisfaction relation between a trajectory $\boldsymbol{x}$ and a tTLT $\mathcal{T}_\varphi$ can be defined as follows.

Definition 4.5. *Consider a trajectory $\boldsymbol{x}$ and a tTLT $\mathcal{T}_\varphi$. We say $\boldsymbol{x}$* satisfies *$\mathcal{T}_\varphi$, denoted by $\boldsymbol{x} \vDash \mathcal{T}_\varphi$, if there exists a time coding $\{t_{\kappa_i}\}$ for $\mathcal{T}_\varphi$ such that the output of Algorithm 2 is* true.

The central idea of Algorithm 2 is to check the Boolean relation among sub-formulas of a given STL formula φ. For instance, assume $\varphi = \wedge_{i=1}^n \varphi_i$, where each $\varphi_i, \forall i = 1, \cdots, n$ contains no Boolean operators. Then one can get from Algorithm 1 that $\mathcal{T}_\varphi$ has n complete paths $\boldsymbol{p}_i, i = 1, \cdots, n$, and each $\boldsymbol{p}_i$ corresponds to a sub-formula φ_i. Then Algorithm 2 dictates that $\boldsymbol{x} \vDash \mathcal{T}_\varphi$ if and only if $\boldsymbol{x}$ satisfies every complete path of $\mathcal{T}_\varphi$. Assume now that $\varphi = \vee_{i=1}^n \varphi_i$, then Algorithm 2 dictates that $\boldsymbol{x} \vDash \mathcal{T}_\varphi$ if and only if $\boldsymbol{x}$ satisfies at least one complete path of $\mathcal{T}_\varphi$.

Algorithm 2 *tTLTSatisfaction*

```
Input: a trajectory $\boldsymbol{x}$, a tTLT $\mathcal{T}_\varphi$, and a time coding $\{t_{\kappa_i}\}$.
Return: true or false.
1:  $\mathcal{T}^c_\varphi \leftarrow$ Compression($\mathcal{T}_\varphi$),
2:  for each complete path $\boldsymbol{p}$ of $\mathcal{T}_\varphi$, do
3:      if $\boldsymbol{x} \models \boldsymbol{p}$ then
4:          set the corresponding leaf node of $\boldsymbol{p}$ in $\mathcal{T}^c_\varphi$ with true,
5:      else
6:          set the corresponding leaf node of $\boldsymbol{p}$ in $\mathcal{T}^c_\varphi$ with false,
7:      end if
8:  end for
9:  set all the non-leaf tube nodes in $\mathcal{T}^c_\varphi$ with false,
10: Backtracking($\mathcal{T}^c_\varphi$),
11: return the root node of $\mathcal{T}^c_\varphi$.
```

Algorithm 2 takes as inputs a trajectory $\boldsymbol{x}$, a tTLT $\mathcal{T}_\varphi$, and a time coding $\{t_{\kappa_i}\}$, and outputs true or false. It works as follows. Given a tTLT $\mathcal{T}_\varphi$, we first compress it via Algorithm 3 (line 1), in this way the resulting compressed tree $\mathcal{T}^c_\varphi$ contains only Boolean operator nodes and tube nodes. Then for each complete path $\boldsymbol{p}$ of $\mathcal{T}_\varphi$, if $\boldsymbol{x} \models \boldsymbol{p}$, one sets the corresponding leaf node of $\boldsymbol{p}$ in $\mathcal{T}^c_\varphi$ (note that $\mathcal{T}^c_\varphi$ and $\mathcal{T}_\varphi$ have the same set of leaf nodes) with true. Otherwise, one sets the corresponding leaf node of $\boldsymbol{p}$ in $\mathcal{T}^c_\varphi$ with false (lines 2-8). After that, we set all the non-leaf tube nodes of $\mathcal{T}^c_\varphi$ with false (line 9) and the resulting tree becomes a Boolean tree (a tree with Boolean operator and Boolean variable nodes). Finally, we backtrack the Boolean tree $\mathcal{T}^c_\varphi$ using Algorithm 4, and return the root node (lines 10-11).

We further detail the *Compression* algorithm (Algorithm 3) and the *Backtracking* algorithm (Algorithm 4) in the following. Algorithm 3 aims at obtaining a simplified tree with Boolean operator nodes and tube nodes only. To do so, we first encode each MTS in the form of $\mathbb{X}_1\Theta_1 \dots \Theta_{N_f-1}\mathbb{X}_{N_f}$ (line 3), and then replace it with one tube node (line 4). Algorithm 4 takes the compressed tree $\mathcal{T}_\varphi^c$ as an input, and then update the parent of each Boolean operator node through a bottom-up traversal. In Algorithm 4, $\mathrm{PA}(\Theta)$ and $\mathrm{CH}_1(\Theta), \mathrm{CH}_2(\Theta)$ represent the parent node and the two children of the Boolean operator node $\Theta \in \{\wedge, \vee\}$, respectively.

Algorithm 3 *Compression*

```
Input: a tTLT $\mathcal{T}_\varphi$.
Return: the compressed tree $\mathcal{T}_\varphi^c$.
1: for each complete path of $\mathcal{T}_\varphi$, do
2:     for each MTS, do
3:         encode the MTS in the form
           of $\mathbb{X}_1(t_k)\Theta_1 \dots \Theta_{N_f-1}\mathbb{X}_{N_f}(t_k)$,
4:         replace the MTS with one tube node $\cup_{i=1}^{N_f}\mathbb{X}_i$,
5:     end for
6: end for
```

Algorithm 4 *Backtracking*

```
Input: a compressed tree $\mathcal{T}_\varphi^c$.
Return: the root node of $\mathcal{T}_\varphi^c$.
for each Boolean operator node $\Theta$ of $\mathcal{T}_\varphi^c$ through a
bottom-up traversal, do
    if $\Theta = \wedge$, then
        $\mathrm{PA}(\Theta) \leftarrow \mathrm{PA}(\Theta) \vee (\mathrm{CH}_1(\Theta) \wedge \mathrm{CH}_2(\Theta))$,
    else
        $\mathrm{PA}(\Theta) \leftarrow \mathrm{PA}(\Theta) \vee (\mathrm{CH}_1(\Theta) \vee \mathrm{CH}_2(\Theta))$,
    end if
end for
```

Example 4.1. *Let us continue with Example 3.1. The tTLT $\mathcal{T}_\varphi$ (right of Figure 5) contains 2 complete paths, i.e.,*

$$\boldsymbol{p}_1 := \mathbb{X}_1 \wedge \mathbb{X}_2 \mathsf{U}_{[a_1,b_1]} \mathbb{X}_4 \mathsf{G}_{[a_2,b_2]} \mathbb{S}_{\mu_1}$$

and

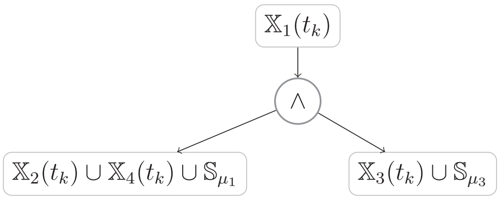
$$\boldsymbol{p}_2 := \mathbb{X}_1 \wedge \mathbb{X}_3 \mathsf{U}_{[a_3,b_3]} \mathbb{S}_{\mu_3}.$$

Let

$$\{t_{\kappa_1}, t_{\kappa_2}, t_{\kappa_4}, t_{\kappa_5}\}$$

be the time coding of the complete path $\boldsymbol{p}_1$, where $t_{\kappa_1}, t_{\kappa_2}, t_{\kappa_4}$, and t_{κ_5} are the activation time instants of the tube nodes $\mathbb{X}_1, \mathbb{X}_2, \mathbb{X}_4$, and $\mathbb{X}_5 := \mathbb{S}_{\mu_1}$, respectively. Then, we have according to Definition 4.4 that a trajectory $\boldsymbol{x} \vDash \boldsymbol{p}_1$ if i) $t_{\kappa_1} = t_{\kappa_2}$; ii) $t_{\kappa_4} \in t_{\kappa_2} + [a_1, b_1]$; iii) $t_{\kappa_5} = \mathrm{argmax}_{t_k}\{t_k \in t_{\kappa_4} + [a_2, b_2]\}$; iv) $x_0 \in \mathbb{X}_1(t_0)$, $x_k \in \mathbb{X}_2(t_{k-\kappa_2}), \forall k \in [\kappa_2, \kappa_4]$, $x_k \in \mathbb{X}_4(t_{k-\kappa_4}), \forall k \in [\kappa_4, \kappa_5]$, and v) $x_{\kappa_5} \in \mathbb{X}_5$.

In addition, the tTLT $\mathcal{T}_\varphi$ contains 3 MTSs, i.e., $\mathbb{X}_1$, $\mathbb{X}_2 \mathsf{U}_{[a_1,b_1]} \mathbb{X}_4 \mathsf{G}_{[a_2,b_2]} \mathbb{S}_{\mu_1}$, and $\mathbb{X}_3 \mathsf{U}_{[a_3,b_3]} \mathbb{S}_{\mu_3}$. The compressed tree $\mathcal{T}_\varphi^c$ is shown in Figure 6. If a trajectory $\boldsymbol{x}$ satisfies both of the complete paths $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, the output of Algorithm 2 is true, *otherwise, the output is* false.



$\mathbb{X}_1(t_k)$
$\wedge$
$\mathbb{X}_2(t_k) \cup \mathbb{X}_4(t_k) \cup \mathbb{S}_{\mu_1}$
$\mathbb{X}_3(t_k) \cup \mathbb{S}_{\mu_3}$

Figure 6. Example 4.1: compressed tree $\mathcal{T}_\varphi^c$, where $\mathcal{T}_\varphi$ is plotted in Figure 5.

Definition 4.6. *(Robustly satisfiable tTLT) A tTLT $\mathcal{T}_\varphi$ is called robustly satisfiable for system (1) with initial state x_0 if there exists a control policy $\boldsymbol{\nu} \in \mathcal{U}_{\geq 0}$ such that $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_\varphi, \forall \boldsymbol{w} \in \mathcal{W}_{\geq 0}$.*

The following theorem provides a formally semantic relation between the STL formula fragment in (5) and the corresponding tTLTs.

Theorem 4.1. *Consider the uncertain system (1) with initial state x_0 and an STL formula φ in (5). Let $\mathcal{T}_\varphi$ be the tTLT corresponding to φ. Then, φ is robustly satisfiable for (1) if $\mathcal{T}_\varphi$ is robustly satisfiable for (1).*

Proof. From Definitions 2.4 and 4.6, one has that to prove Theorem 4.1, it is equivalent to prove $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_\varphi, \forall \boldsymbol{w} \in \mathcal{W}_{\geq 0} \Rightarrow \boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi, \forall \boldsymbol{w} \in \mathcal{W}_{\geq 0}$. Given one instance of disturbance signal $\boldsymbol{w}$, if one has $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_\varphi \Rightarrow \boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi$, then it implies $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu}} \vDash \mathcal{T}_\varphi, \forall \boldsymbol{w} \in \mathcal{W}_{\geq 0} \Rightarrow \boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi, \forall \boldsymbol{w} \in \mathcal{W}_{\geq 0}$. Therefore, it is sufficient to prove $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_\varphi \Rightarrow \boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi$.

In the following, we will first prove $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_\varphi \Leftrightarrow \boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi$ for

i) $\top$, predicates $\mu, \neg\mu$, and $\mu_1 \wedge \mu_2, \mu_1 \vee \mu_2$,
ii) $\mu_1 \mathsf{U}_{[a,b]} \mu_2$, $\mathsf{F}_{[a,b]}\mu_1$, and $\mathsf{G}_{[a,b]}\mu_1$;
iii) $\mu_1 \mathsf{U}_{[a_1,b_1]} \mathsf{G}_{[a_2,b_2]} \mu_2$ and $\mathsf{F}_{[a_1,b_1]} \mathsf{G}_{[a_2,b_2]} \mu_1$;
iv) $\varphi_1 \wedge \varphi_2$;

where φ_1 and φ_2 in item *iv)* are STL formulas belong to items *ii)* or *iii)*.

Case i): For $\top$, predicates $\mu, \neg\mu$, and $\mu_1 \wedge \mu_2, \mu_1 \vee \mu_2$, it is trivial to verify that $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_\varphi \Leftrightarrow \boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi$.

Case ii): We note that the proofs of the three are similar, therefore, in the following, we only consider the case $\varphi = \mu_1 \mathsf{U}_{[a,b]} \mu_2$. The tTLT $\mathcal{T}_\varphi$ can be constructed via Algorithm 1, which is shown in Figure 7.

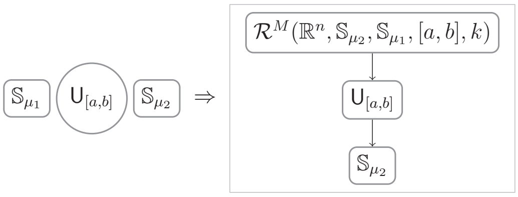
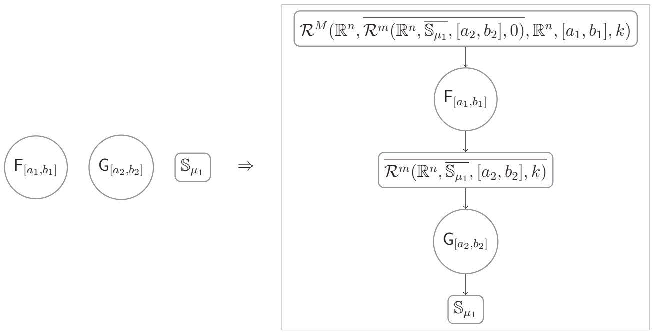


Figure 7. tTLTs $\mathcal{T}_\varphi$ for $\varphi = \mu_1 \mathsf{U}_{[a,b]} \mu_2$.

Assume that $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_\varphi$, then one has from Definition 4.4 that $\exists t_{\kappa_1} \in t_0 + [a,b], x_{\kappa_1} \in \mathbb{S}_{\mu_2}$ and $\forall k \in [0, \kappa_1], x_k \in \mathcal{R}^M(\mathbb{R}^n, \mathbb{S}_{\mu_2}, \mathbb{S}_{\mu_1}, [a,b], k) \subseteq \mathbb{S}_{\mu_1}$, which implies $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi$. That is, $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_\varphi \Rightarrow \boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi$. Assume now that $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi$. Then, one has from STL semantics that i) $\exists t_{k'} \in t_0 + [a,b], x_{k'} \in \mathbb{S}_{\varphi_2}$

and ii) $\forall t_{k''} \in [t_0, t_{k'}], x_{k''} \in \mathbb{S}_{\varphi_1}$. Moreover, from Definition 2.6, one has that i) and ii) together implies $\forall t_{k''} \in [t_0, t_{k'}], x_{k''} \in \mathcal{R}^M(\mathbb{R}^n, \mathbb{S}_{\mu_2}, \mathbb{S}_{\mu_1}, [a,b], k'')$. Therefore, $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi \Rightarrow \boldsymbol{x}_{x_0}^{\boldsymbol{\nu}} \vDash \mathcal{T}_\varphi$.

Case iii): We note that the proofs of the two are similar. In the following, we consider the case $\varphi = \mathsf{F}_{[a_1,b_1]}\mathsf{G}_{[a_2,b_2]}\mu_1$. The tTLT $\mathcal{T}_\varphi$ can be constructed via Algorithm 1, which is shown in Figure 8.



$\mathsf{F}_{[a_1,b_1]}$
$\mathsf{G}_{[a_2,b_2]}$
$\mathbb{S}_{\mu_1}$
$\Rightarrow$
$\mathcal{R}^M(\mathbb{R}^n, \overline{\mathcal{R}^m(\mathbb{R}^n, \overline{\mathbb{S}_{\mu_1}}, [a_2,b_2], 0)}, \mathbb{R}^n, [a_1,b_1], k)$
$\mathsf{F}_{[a_1,b_1]}$
$\overline{\mathcal{R}^m(\mathbb{R}^n, \overline{\mathbb{S}_{\mu_1}}, [a_2,b_2], k)}$
$\mathsf{G}_{[a_2,b_2]}$
$\mathbb{S}_{\mu_1}$

Figure 8. tTLTs $\mathcal{T}_\varphi$ for $\varphi = \mathsf{F}_{[a_1,b_1]}\mathsf{G}_{[a_2,b_2]}\mu_1$.

Assume that $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_\varphi$, then one has from Definition 4.4 that $t_{\kappa_1} \in t_0 + [a_1,b_1], t_{\kappa_2} = \text{argmax}_{t_k}\{t_{\kappa_1} + [a_2,b_2]\}$. In addition, $\forall k \in [\kappa_1, \kappa_2], x_k \in \overline{\mathcal{R}^m(\mathbb{R}^n, \overline{\mathbb{S}_{\mu_1}}, [a_2,b_2], k-\kappa_1)}$, which implies $x_k \in \mathbb{S}_{\mu_1}, \forall k \in [\kappa_1,\kappa_2]$. That is, $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_\varphi \Rightarrow \boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi$. Assume now that $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi$. Then, one has from STL semantics that $\exists t_{k'} \in t_0 + [a_1,b_1]$ such that $x_{k''} \in \mathbb{S}_{\mu_1}, \forall t_{k''} \in t_{k'} + [a_2,b_2]$, which implies $\forall t_{k''} \in t_{k'} + [a_2,b_2], x_{k''} \in \overline{\mathcal{R}^m(\mathbb{R}^n, \overline{\mathbb{S}_{\mu_1}}, [a_2,b_2], k''-k')}$. Therefore, $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi \Rightarrow \boldsymbol{x}_{x_0}^{\boldsymbol{\nu}} \vDash \mathcal{T}_\varphi$.

Case iv): $\varphi = \varphi_1 \wedge \varphi_2$. Assume that $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_\varphi$, then one has from Definition 4.4 that $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_{\varphi_1}$ and $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_{\varphi_2}$. Moreover, since φ_1 and φ_2 belong to items ii) or iii), then one can conclude from *Case ii)* and *Case iii)* that $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_{\varphi_i} \Rightarrow \boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi_i, i = \{1,2\}$, which implies $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi_1 \wedge \varphi_2$. That is, $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_\varphi \Rightarrow \boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi$. The proof of the other direction is similar and hence omitted.

Then, we prove $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_\varphi \Rightarrow \boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi$ for

v) $\varphi_1 \vee \varphi_2$,

where φ_1 and φ_2 are STL formulas belong to items *ii)* or *iii)*.

Case v): $\varphi = \varphi_1 \vee \varphi_2$. The proof of $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_\varphi \Rightarrow \boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi$ is similar to *Case iv)*. The other direction does not hold because for an uncertain system, it is possible that there exists a trajectory $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}}$ such that $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi$, however, the initial state $x_0 \notin \mathbb{X}_{\text{root}}^{\varphi}(t_0)$ (due to Proposition 3.2), where $\mathbb{X}_{\text{root}}^{\varphi}$ denotes the root node of $\mathcal{T}_\varphi$. In this case, $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}}$ does not satisfy $\mathcal{T}_\varphi$.

The proof of $\boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \mathcal{T}_\varphi \Rightarrow \boldsymbol{x}_{x_0}^{\boldsymbol{\nu},\boldsymbol{w}} \vDash \varphi$ for other STL formulas φ in (5) can be completed inductively by combining *Cases i)-v)*. Therefore, the conclusion follows. □

Thanks to the semantic relation between the STL formulas in (5) and their corresponding tTLTs, we are able to perform control synthesis over the tTLT, instead of the STL formulas, with correct-by-construction guarantees. The details of this control synthesis are provided in the next section.

5 Online Control Synthesis

In this section, we study the STL control synthesis problem in Problem 2.1. In the following, an online control synthesis algorithm and its sub-algorithms are designed over the tTLT such that the tTLT $\mathcal{T}_\varphi$ is satisfied (in the sense of Definitions 4.4 and 4.5). From Theorem 4.1, one can see that to guarantee the satisfaction of the STL formula φ in (5), it is sufficient to find a control policy $\boldsymbol{\nu}$ that guarantees the (robust) satisfaction of the corresponding tTLT $\mathcal{T}_\varphi$. To this end, the tTLT-based control synthesis approach is sound.

5.1 Definitions and notations

Before proceeding, the following definitions and notations are needed.

Definition 5.1. *The time horizon $|\Theta|$ of an STL operator $\Theta \in \{\wedge, \vee, \mathsf{U}_{[a,b]}, \mathsf{F}_{[a,b]}, \mathsf{G}_{[a,b]}\}$ is defined as*

$$|\Theta| = \begin{cases} 0, & \textit{if}\, \Theta = \{\wedge, \vee\}, \\ \hat{b}, & \textit{if}\, \Theta \in \{\mathsf{U}_{[a,b]}, \mathsf{F}_{[a,b]}, \mathsf{G}_{[a,b]}\}, \end{cases}$$

where $\hat{b} = \text{argmax}_{t_k}\{a \leq t_k \leq b\}$.

Definition 5.2. *A segment of a complete path of a tTLT is called* a Boolean segment *if it starts and ends with a tube node and contains only Boolean operator nodes. We say a tube node $\mathbb{X}_j$ is* reachable *from $\mathbb{X}_i$* by a Boolean segment *if there exists a Boolean segment that starts with $\mathbb{X}_i$ and ends with $\mathbb{X}_j$.*

Definition 5.3. *If each node of a tree is either a set node that is a subset of U or an operator node that belongs to $\{\wedge, \vee, \mathsf{U}_I, \mathsf{F}_I, \mathsf{G}_I\}$, then the tree is called a* control tree.

Each tube node $\mathbb{X}_i$ of the tTLT $\mathcal{T}_\varphi$ is characterized by the following two parameters:

- $t_a(\mathbb{X}_i)$: the activation time of $\mathbb{X}_i$,
- $t_h(\mathbb{X}_i)$: the time horizon of $\mathbb{X}_i$, *i.e.*, the time that $\mathbb{X}_i$ is deactivated.

Denote by $\mathcal{T}_\varphi(t_k)$ the resulting tree of $\mathcal{T}_\varphi$ at time instant t_k. It is obtained by fixing the value of each tube node $\mathbb{X}_i$ according to the activation time $t_a(\mathbb{X}_i)$ (*i.e.*, $\mathcal{T}_\varphi(t_k)$ contains either set nodes or operator nodes). Let $S_i(t_k)$ be the i-th set node of $\mathcal{T}_\varphi(t_k)$, where $S_i(t_k)$ corresponds to the tube node $\mathbb{X}_i$. The relationship between $S_i(t_k)$ and $\mathbb{X}_i$ can be described as follows:

$$S_i(t_k) = \begin{cases} \mathbb{X}_i(t_0), & \text{if } t_k \leq t_a(\mathbb{X}_i), \\ \mathbb{X}_i(t_k - t_a(\mathbb{X}_i)), & \text{if } t_k > t_a(\mathbb{X}_i). \end{cases} \tag{8}$$

Moreover, one has that

$$t_a(S_i(t_k)) = t_a(\mathbb{X}_i), t_h(S_i(t_k)) = t_h(\mathbb{X}_i), \forall k \geq 0.$$

At each time instant t_k, $\mathcal{T}_\varphi(t_k)$ is characterized by

- $P(t_k)$: the set which collects all the set nodes of $\mathcal{T}_\varphi(t_k)$, *i.e.*, $P(t_k) = \cup_i S_i(t_k)$,
- Θ: the set which collects all the operator nodes of $\mathcal{T}_\varphi(t_k)$, which is time invariant.

For a node $N_i(t_k) \in P(t_k) \cup \Theta$, define

- CH$(N_i(t_k))$: the set of children of node $N_i(t_k)$,
- PA$(N_i(t_k))$: the set of parents of node $N_i(t_k)$,
- Post$(N_i(t_k)) :=$ CH(CH$(N_i(t_k))$),
- Pre$(N_i(t_k)) :=$ PA(PA$(N_i(t_k))$).

Given a state-time pair (x_k, t_k), define $L : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to 2^{P(t_k)}$ as the labelling function, given by

$$L(x_k, t_k) = \{S_i(t_k) \in P(t_k) : x_k \in S_i(t_k), t_k \leq t_h(S_i(t_k))\}, \tag{9}$$

which maps (x_k, t_k) to a subset of $P(t_k)$. Moreover, define the function $B : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to 2^{P(t_k)}$, which maps (x_k, t_k) to a set of valid set nodes in $P(t_k)$. The function $L(x_k, t_k)$ computes the subset of set nodes of $P(t_k)$ that contains x_k at time t_k (without the consideration of history trajectory) while the function $B(x_k, t_k)$ is further introduced to capture the fact that given the history trajectory, not all set nodes in $L(x_k, t_k)$ are valid at time t_k. A rule for determining $B(x_k, t_k)$ given $L(x_k, t_k)$ is detailed in Algorithm 7 in the next subsection.

5.2 Online control synthesis

In the following, we present the online control synthesis algorithm (and its sub-algorithms), and then present an example to further explain how each sub-algorithm works.

Algorithm 5 *onlineControlSynthesis*

```
Input: The tTLT $\mathcal{T}_\varphi$ and $(x_0, t_0)$.
Return: NExis or $(\boldsymbol{\nu}, \boldsymbol{x})$ with $\boldsymbol{\nu} = \nu_0\nu_1\ldots\nu_k\ldots$ and $\boldsymbol{x} = x_0x_1\ldots x_k\ldots$.
1: $(t_a, t_h, \texttt{Post}(B(x_{-1}, t_{-1}))) \leftarrow initialization(\mathcal{T}_\varphi)$,
2: $B(x_k, t_k) \leftarrow trackingSetNode(\texttt{Post}(B(x_{k-1}, t_{k-1})))$,
3: for each $S_i(t_k) \in B(x_k, t_k)$, do
4:     if $t_a(S_i(t_k)) = \bowtie$, then
5:         $t_a(\mathbb{X}_i) \leftarrow t_k$,
6:     end if
7: end for
8: $\mathcal{T}_\varphi(t_{k+1}) \leftarrow updatetTLT(\mathcal{T}_\varphi(t_k), t_a, B(x_k, t_k))$,
9: $\mathcal{T}_u(t_k) \leftarrow buildControlTree(\mathcal{T}_\varphi(t_k), B(x_k, t_k), \mathcal{T}_\varphi(t_{k+1}))$,
10: $\mathcal{T}_u^c(t_k) \leftarrow Compression(\mathcal{T}_u(t_k))$,
11: $\mathbb{U}(x_k, t_k) \leftarrow Backtracking*(\mathcal{T}_u^c)$,
12: if $\mathbb{U}(x_k, t_k) = \emptyset$, then
13:     stop and return NExis,
14: else
15:     choose $\nu_k \in \mathbb{U}(x_k, t_k)$,
16:     implement $\nu_k$ and measure $x_{k+1}$,
17:     $\texttt{Post}(B(x_k, t_k)) \leftarrow postSet(B(x_k, t_k), t_a, \mathcal{T}_\varphi(t_{k+1}))$,
18:     update $k = k + 1$ and go to line 2.
19: end if
```

Algorithm 6 *initialization*

```
Input: The tTLT $\mathcal{T}_\varphi$.
Return: $t_a, t_h, \texttt{Post}(B(x_{-1}, t_{-1}))$.
1: $t_a(\mathbb{X}^\varphi_{\text{root}}) \leftarrow t_0, t_h(\mathbb{X}^\varphi_{\text{root}}) \leftarrow t_0 + |\text{CH}(\mathbb{X}^\varphi_{\text{root}})|$,
2: for each non-root and non-leaf tube node $\mathbb{X}_i$ through a top-down traversal, do
3:     $t_a(\mathbb{X}_i) \leftarrow \bowtie, t_h(\mathbb{X}_i) \leftarrow t_h(\text{Pre}(\mathbb{X}_i) + |\text{CH}(\mathbb{X}_i)|$,
4: end for
5: for each leaf node $\mathbb{X}_i$, do
6:     $t_a(\mathbb{X}_i) \leftarrow \bowtie, t_h(\mathbb{X}_i) \leftarrow \infty$,
7: end for
8: $\texttt{Post}(B(x_{-1}, t_{-1})) \leftarrow \mathbb{X}^\varphi_{\text{root}}(t_0)$,
9: for each $\mathbb{X}_j$ that is reachable from $\mathbb{X}^\varphi_{\text{root}}$ by a Boolean segment (see Definition 5.2), do
10:     $\texttt{Post}(B(x_{-1}, t_{-1})) \leftarrow \texttt{Post}(B(x_{-1}, t_{-1})) \cup \mathbb{X}_j(t_0)$,
11:     $t_a(\mathbb{X}_j) \leftarrow t_0$,
12: end for
```

Algorithm 7 *trackingSetNode*

```
Input: $\texttt{Post}(B(x_{k-1}, t_{k-1}))$.
Return: $B(x_k, t_k)$.
1: Compute $L(x_k, t_k)$ according to (9),
2: $B(x_k, t_k) \leftarrow L(x_k, t_k) \cap \texttt{Post}(B(x_{k-1}, t_{k-1}))$,
3: for each $S_i(t_k) \in B(x_k, t_k)$ do,
4:     if $\exists S_j(t_k) \in B(x_k, t_k)$ s.t. $S_j(t_k) = \text{Post}(S_i(t_k))$, then
5:         $B(x_k, t_k) \leftarrow B(x_k, t_k) \setminus S_i(t_k)$,
6:     end if
7: end for
```

The online control synthesis algorithm is outlined in Algorithm 5. Before implementation, an initialization process (line 1) is required, which is outlined in Algorithm 6. Here, t_a and t_h are two functions that map each tube node $\mathbb{X}_i$ to its activation time and time horizon, respectively. If $t_a(\mathbb{X}_i)$ or $t_h(\mathbb{X}_i)$ is unknown for $\mathbb{X}_i$, its value will be set as $\bowtie$. Then, at each time instant t_k, a feasible control set $\mathbb{U}(x_k, t_k)$ is synthesized (lines 2-11). This process contains the following steps: 1) find the subset of set nodes in $P(t_k)$ that are valid at time t_k, *i.e.,* $B(x_k, t_k)$, via Algorithm 7 (line 2); 2) determine the activation time of $\mathbb{X}_i$, whose corresponding set node $S_i(t_k) \in B(x_k, t_k)$ (if $t_a(\mathbb{X}_i)$ is unknown, *i.e.,* being visited for the first time, it is set as t_k; otherwise, *i.e.,* being visited before, it is unchanged) (lines 3-7); 3) calculate $\mathcal{T}_\varphi(t_{k+1})$ via Algorithm 8 (line 8); 4) build a control tree $\mathcal{T}_u(t_k)$ (Definition 5.3) via Algorithm 9 (line 9), compress it via Algorithm 3 (line 10), and then the feasible control set $\mathbb{U}(x_k, t_k)$ is given by backtracking the compressed control tree $\mathcal{T}_u^c(t_k)$ via Algorithm 10 (line 11). If the obtained feasible control set $\mathbb{U}(x_k, t_k) = \emptyset$, the control synthesis process stops and returns NExis (lines 12-13); otherwise, the control input ν_k can be chosen as any element of $\mathbb{U}(x_k, t_k)$ (one example is to choose ν_k as $\min_{\nu_k \in \mathbb{U}(x_k, t_k)}\{\|\nu_k\|\}$) (line 15). Then, we implement the chosen ν_k, measure x_{k+1} (line 16), and finally compute the subset of set nodes that are possibly available at the next time instant t_{k+1}, *i.e.,* $\texttt{Post}(B(x_k, t_k))$, via Algorithm 11 (line 17).

We further detail Algorithms 6-11 in the following.

- Algorithm 6 calculates the functions t_a and t_h (lines 1-7) and $\texttt{Post}(B(x_{-1}, t_{-1}))$ (lines 8-12).
- Algorithm 7 outlines the procedure of finding the subset of set nodes in $P(t_k)$ that are valid at time t_k, *i.e.,* $B(x_k, t_k)$. This is the most important step of the control synthesis, and it relates to Algorithm 11 *postSet*. Firstly, one needs to compute the subset of set nodes of $P(t_k)$ that contains x_k at time t_k, *i.e.,* $L(x_k, t_k)$ (line 1). Then, one has from Definition

Algorithm 8 *updatetTLT*

```
Input: $\mathcal{T}_\varphi(t_k)$, $t_a$ and $B(x_k, t_k)$.
Return: $\mathcal{T}_\varphi(t_{k+1})$.
1: for each set node $S_i(t_k)$ of $\mathcal{T}_\varphi(t_k)$, do
2:     if $S_i(t_k) \in B(x_k, t_k) \wedge t_a(S_i(t_k)) + |\mathrm{CH}(S_i(t_k))| \geq t_{k+1}$, then
3:         $S_i(t_{k+1}) \leftarrow \mathbb{X}_i(t_{k+1} - t_a(S_i(t_k)))$,
4:     else
5:         $S_i(t_{k+1}) \leftarrow S_i(t_k)$,
6:     end if
7: end for
```

Algorithm 9 *buildControlTree*

```
Input: $\mathcal{T}_\varphi(t_k)$, $B(x_k, t_k)$, and $\mathcal{T}_\varphi(t_{k+1})$.
Return: A control tree $\mathcal{T}_u(t_k)$.
1: Initialize $\mathcal{T}_u(t_k)$ as $\mathcal{T}_\varphi(t_k)$,
2: for each $S_i(t_k) \in B(x_k, t_k)$ do
3:     if $S_i(t_k)$ is a leaf node then,
4:         $S_i(t_k) \leftarrow \mathbb{U}(S_i(t_k)) := U$,
5:     else
6:         $S_i(t_k) \leftarrow \mathbb{U}(S_i(t_k)) := \{u_k \in U : f_k(x_k, u_k, w_k) \in S_i(t_{k+1}), \forall w_k \in W\}$,
7:     end if
8: end for
9: for each $S_i(t_k) \notin B(x_k, t_k)$ do
10:     $S_i(t_k) \leftarrow \emptyset$,
11: end for
```

Algorithm 10 *Backtracking**

```
Input: a compressed tree $\mathcal{T}_u^c(t_k)$.
Return: the root node of $\mathcal{T}_u^c(t_k)$.
1: for each Boolean operator node $\Theta$ of $\mathcal{T}_u^c(t_k)$ through a
   bottom-up traversal, do
2:     if $\Theta = \wedge$, then
3:         $\mathrm{PA}(\Theta) \leftarrow \mathrm{PA}(\Theta) \cup (\mathrm{CH}_1(\Theta) \cap \mathrm{CH}_2(\Theta))$,
4:     else
5:         $\mathrm{PA}(\Theta) \leftarrow \mathrm{PA}(\Theta) \cup (\mathrm{CH}_1(\Theta) \cup \mathrm{CH}_2(\Theta))$,
6:     end if
7: end for
```

Algorithm 11 *postSet*

```
Input: $B(x_k, t_k), t_a$ and $\mathcal{T}_\varphi(t_{k+1})$.
Return: $\mathtt{Post}(B(x_k, t_k))$.
1: Initialize $\mathtt{Post}(S_i(t_k)) = \emptyset, \forall S_i(t_k) \in B(x_k, t_k)$.
2: for each $S_i(t_k) \in B(x_k, t_k)$, do
3:     switch the children of $S_i(t_k)$ do
4:         case $\mathrm{CH}(S_i(t_k)) \in \{\wedge, \vee\}$,
5:             $\mathtt{Post}(S_i(t_k)) \leftarrow S_i(t_{k+1})$,
6:             for each $S_j(t_k)$ that is reachable from $S_i(t_k)$ by a Boolean segment, do
7:                 $\mathtt{Post}(S_i(t_k)) \leftarrow \mathtt{Post}(S_i(t_k)) \cup S_j(t_{k+1})$,
8:             end for
9:         case $\mathrm{CH}(S_i(t_k)) \in \{\mathsf{U}_{[a,b]}, \mathsf{F}_{[a,b]}\}$,
10:             if $t_k > t_a(\mathsf{Pre}(S_i(t_k)) + a\}$, then
11:                 $\mathtt{Post}(S_i(t_k)) \leftarrow S_i(t_{k+1}) \cup \mathsf{Post}(S_i(t_{k+1}))$,
12:             end if
13:         case $\mathrm{CH}(S_i(t_k)) \in \{\mathsf{G}_{[a,b]}\}$,
14:             if $t_k > t_a(\mathsf{Pre}(S_i(t_k)) + b\}$, then
15:                 $\mathtt{Post}(S_i(t_k)) \leftarrow S_i(t_{k+1}) \cup \mathsf{Post}(S_i(t_{k+1}))$,
16:             end if
17: end for
```

4.4 that if a trajectory $\boldsymbol{x}$ satisfies one complete path of the tTLT, it must i) visit each tube node of the complete path sequentially and ii) stay in each tube node for sufficiently long time steps (Remark 4.1). Based on these two requirements, Algorithm 11 is designed to predict the subset of set nodes that are possibly available at the next time instant, *i.e.*, $\mathtt{Post}(B(x_{k-1}, t_{k-1}))$. Note that $B(x_k, t_k)$ must belong to $L(x_k, t_k)$ and $\mathtt{Post}(B(x_{k-1}, t_{k-1}))$ at the same time. Therefore, we let $B(x_k, t_k) \leftarrow L(x_k, t_k) \cap \mathtt{Post}(B(x_{k-1}, t_{k-1}))$ (line 2). The rest of Algorithm 7 (lines 3-7) guarantees that $B(x_k, t_k)$ contains at most one set node for each complete path of $\mathcal{T}_\varphi(t_k)$.

- Algorithm 8 outlines the procedure of calculating $\mathcal{T}_\varphi(t_{k+1})$, given $\mathcal{T}_\varphi(t_k)$, t_a and $B(x_k, t_k)$. It is designed based on (8).
- Algorithm 9 outlines the procedure of building a control tree $\mathcal{T}_u(t_k)$, which is then used for control set synthesis. It is initialized as $\mathcal{T}_\varphi(t_k)$ (line 1). Then, for those set nodes $S_i(t_k)$ that belongs to $B(x_k, t_k)$, it is replaced with the feasible control set (lines 2-8), otherwise, it is replaced with $\emptyset$ (lines 9-11).
- Algorithm 10 is similar to Algorithm 4, which outlines the procedure of backtracking a compressed tree.
- Algorithm 11 outlines the procedure of finding the subset of set nodes that are possibly available at the next time instant t_{k+1} given $B(x_k, t_k)$, t_a and $\mathcal{T}_\varphi(t_{k+1})$. It is designed based on Definition 4.4, where the three cases (lines 4-8, 9-12, 13-16) correspond to items *i)-iii)* of Definition 4.4, respectively. It guarantees that the resulting trajectory visits each tube node of $\mathcal{T}_\varphi$ sequentially and stays in each tube node for sufficiently long time steps (as we discussed in Algorithm 7).

Next, an example is given to illustrate one iteration of the control synthesis algorithm (Algorithm 5).

Example 5.1. *Consider the single-integrator control system $\dot{x} = u + w$ with a sampling period of one second. The corresponding discrete-time system is given by*

$$x_{k+1} = x_k + u_k + w_k,$$

where $x_k \in \mathbb{R}^2, u_k \in U := \{u : ||u|| \leq 1\} \subset \mathbb{R}^2, w_k \in W := \{w : ||w|| \leq 0.1\} \subset \mathbb{R}^2, \forall k \in \mathbb{N}$. *The task specification φ is given in Example 3.1, i.e.,* $\varphi = \mathsf{F}_{[a_1,b_1]}\mathsf{G}_{[a_2,b_2]}\mu_1 \wedge \mu_2 \mathsf{U}_{[a_3,b_3]}\mu_3$, *where* $[a_1, b_1] = [5, 10]$, $[a_2, b_2] = [0, 10]$, $[a_3, b_3] = [0, 8]$, $g_{\mu_1}(x) = 1 - \|x\|$, $g_{\mu_2}(x) = 5 - \|x - [4, 4]^T\|$, *and* $g_{\mu_3}(x) = 1 - \|x - [3, 5]^T\|$. *Then, one has*

$$\mathbb{S}_{\mu_1} = \{x_0 : \|x_0\| \leq 1\},$$

$$\mathbb{S}_{\mu_2} = \{x_0 : \|x_0 - [4, 4]^T\| \leq 5\},$$

$$\mathbb{S}_{\mu_3} = \{x_0 : \|x_0 - [3,5]^T\| \le 1\}.$$

The tTLT that corresponds to φ is plotted in Figure 5. Using Definitions 2.6 and 2.7, one can calculate that

$$\begin{aligned} \mathbb{X}_4(t_k) &= \{x_k : \|x_k\| \le 0.9\}, \\ \mathbb{X}_3(t_k) &= \{x_k : \|x_k - [3,5]^T\| \le 8.1 - k \\ &\qquad\qquad \wedge \|x_k - [4,4]^T\| \le 5\}, \\ \mathbb{X}_2(t_k) &= \{x_k : \|x_k\| \le 9.9 - k\}, \\ \mathbb{X}_1(t_k) &= \mathbb{X}_2(t_k) \cap \mathbb{X}_3(t_k). \end{aligned}$$

The initial state $x_0 = [0.5, 0.8]^T$, for which $x_0 \in \mathbb{X}^{\varphi}_{root}(t_0)$. Firstly, an initialization process is required, and one can get from Algorithm 6 that

$$\begin{aligned} &t_h(\mathbb{X}_1) = 0, t_h(\mathbb{X}_2) = 10, t_h(\mathbb{X}_3) = 8, \\ &t_h(\mathbb{X}_4) = 20, t_h(\mathbb{S}_{\mu_1}) = \infty, t_h(\mathbb{S}_{\mu_3}) = \infty, \end{aligned}$$

and

$$\mathtt{Post}(B(x_{-1}, t_{-1})) = \{\mathbb{X}_1(t_0), \mathbb{X}_2(t_0), \mathbb{X}_3(t_0)\}.$$

Now, let us see how the feasible control set $\mathbb{U}(x_0, t_0)$ is synthesized at time instant t_0.

1) Find $B(x_0, t_0)$ via Algorithm 7. First, $L(x_0, t_0)$ is computed according to (9),

$$L(x_0, t_0) = \{\mathbb{X}_1(t_0), \mathbb{X}_2(t_0), \mathbb{X}_3(t_0), \mathbb{X}_4(t_0), \mathbb{S}_{\mu_1}\}.$$

Then, after running lines 2-7, one has

$$B(x_0, t_0) = \{S_2(t_0), S_3(t_0)\}.$$

2) Determine the activation time. Initially, both $t_a(\mathbb{X}_2)$ and $t_a(\mathbb{X}_3)$ are unknown, therefore, $t_a(\mathbb{X}_2) = t_a(\mathbb{X}_3) = t_0$.

3) Update the TLT (thus obtain $\mathcal{T}_\varphi(t_1)$) via Algorithm 8. The output $\mathcal{T}_\varphi(t_1)$ is given by

$$\begin{aligned} &S_1(t_1) = \mathbb{X}_1(t_0), S_2(t_1) = \mathbb{X}_2(t_1), \\ &S_3(t_1) = \mathbb{X}_3(t_1), S_4(t_1) = \mathbb{X}_4(t_0), \end{aligned}$$

and the leaf nodes $\mathbb{S}_{\mu_1}$ and $\mathbb{S}_{\mu_3}$ are unchanged.

4) Build the control tree $\mathcal{T}_u(t_0)$, compress it to obtain $\mathcal{T}^c_u(t_0)$, and then get $\mathbb{U}(x_0, t_0)$. This process is illustrated in Figure 9, and $\mathbb{U}(x_0, t_0) = \mathbb{U}(S_2(t_0)) \cap \mathbb{U}(S_3(t_0))$.

Since $\mathbb{U}(x_0, t_0) \neq \emptyset$, the online control synthesis continues, and we can further compute $\mathtt{Post}(B(x_0, t_0))$ via Algorithm 11, which gives

$$\mathtt{Post}(B(x_0, t_0)) = \{S_2(t_1), S_3(t_1), \mathbb{S}_{\mu_3}\}.$$

The following theorem shows the applicability and soundness of Algorithm 5.

Theorem 5.1. *Consider uncertain system (1) with initial state x_0 and an STL formula φ in (5). Assume that φ is robustly satisfiable for (1) and $x_0 \in \mathcal{T}^{\varphi}_{root}(t_0)$. Then, implementing the online control synthesis algorithm (Algorithm 5) guarantees that*

(i) the control set $\mathbb{U}(x_k, t_k)$ is nonempty for all $k \in \mathbb{N}$;
(ii) the resulting trajectory $\boldsymbol{x} \models \varphi$.

Proof. The proof follows from the construction of tTLT and Algorithms 5-11. The existence of a controller ν_k at each time step t_k, is guaranteed by the definition of maximal and minimal reachable sets (Definitions 2.6 and 2.7), and the construction of tTLT (Propoition 3.1, Theorem 3.1 and Algorithm 1). Moreover, the design of Algorithms 5-11 guarantees that the resulting trajectory $\boldsymbol{x}$ satisfies the tTLT $\mathcal{T}_\varphi$, *i.e.*, $\boldsymbol{x} \models \mathcal{T}_\varphi$, which implies $\boldsymbol{x} \models \varphi$ as proven in Theorem 4.1. □

Remark 5.1. *The tTLT construction relies on the computation of backward reachable tubes. Over the past decade, new approaches (e.g., decomposition-based approach (Chen et al., 2018a) and learning-based approaches (Allen et al., 2014; Bansal and Tomlin, 2021)) and software tools (e.g., Hamilton-Jacobi Toolbox (Mitchell and Templeton, 2005) and CORA Toolbox (Althoff, 2015)), have been developed for improving the efficiency of computing backward reachable tubes. Moreover, we remark that the computation of reachable tubes in our work for constructing of the tTLT can be performed* offline*, which may mitigate the online computational burden. On the other hand, although the exact computation of backward reachable sets/tubes is in general nontrivial for high-dimensional nonlinear systems, efficient algorithms exist for linear systems with polygonal input and disturbance sets (Kurzhanski and Pravin, 2014).*

Remark 5.2. *The online control synthesis algorithm (Algorithm 5) contains 7 sub-algorithms, i.e., Algorithm 3 and Algorithms 6-11. The computational complexity is determined by Algorithm 9, in which one-step feasible control sets need to be computed. The computational complexity of Algorithms 3, 6, 7, 8, 10, 11 is $\mathcal{O}(1)$. Note that in Algorithm 8, the computation of reachable sets, which is required for set node update, is done offline when constructing the tTLT.*

Remark 5.3. *Different from the mixed-integer programming formulation for STL control synthesis (Raman et al., 2015, 2014), where an entire control policy has to be synthesized at each time step, the control synthesis in our work is reactive in the sense that only the control input at the current time step is generated at each time step.*

6 Numerical Simulations

In this section, two examples illustrating the theoretical results are provided. We first perform a numerical simulation for car overtaking. We then apply our algorithms to motion planning of a mobile robot over a group of STL specifications and test the scalability of our algorithms with respect to the growing STL complexity.

6.1 Car overtaking example

We first consider a car overtaking example. This example will specify an overtaking task as an STL formula and then show how to synthesize an overtaking controller with safety guarantees.

As shown in Figure 10, we consider a scenario where an automated vehicle Veh$_1$ plans to move to a target set $\mathbb{S}_{\mu_1}$ within 80 seconds. Since there is a broken vehicle Veh$_2$ in front of Veh$_1$ and there is another vehicle Veh$_3$ that moves in

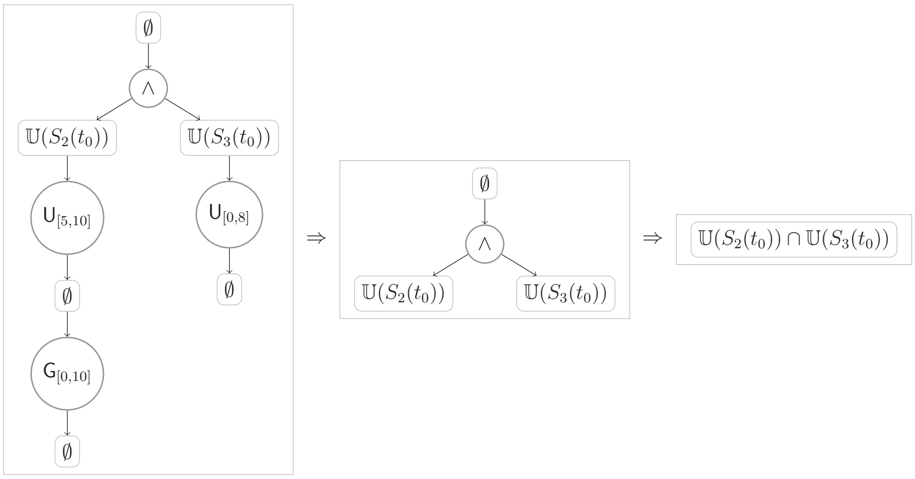


Figure 9. Left: $\mathcal{T}_u(t_0)$, Middle: $\mathcal{T}_u^c(t_0)$, Right: root node of $\mathcal{T}_u^c(t_0)$ after implementing Algorithm 10, where $\mathbb{U}(S_2(t_0)) = U = \{u : ||u|| \leq 1\}, \mathbb{U}(S_3(t_0)) = U \cap \{u : \|u - [3.4, 3.1]^T\| \leq 5\}$.

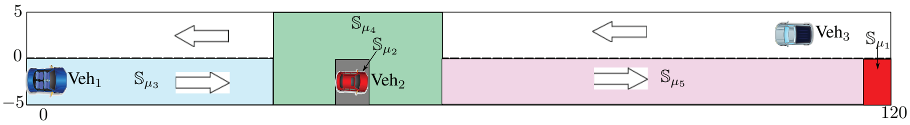


Figure 10. Scenario illustration: an automated vehicle plans to reach a target set $\mathbb{S}_{\mu_1}$ while overtaking a broken vehicle Veh$_2$ in front of it in the same lane and avoiding Veh$_3$ moving in an opposite direction in the other lane.

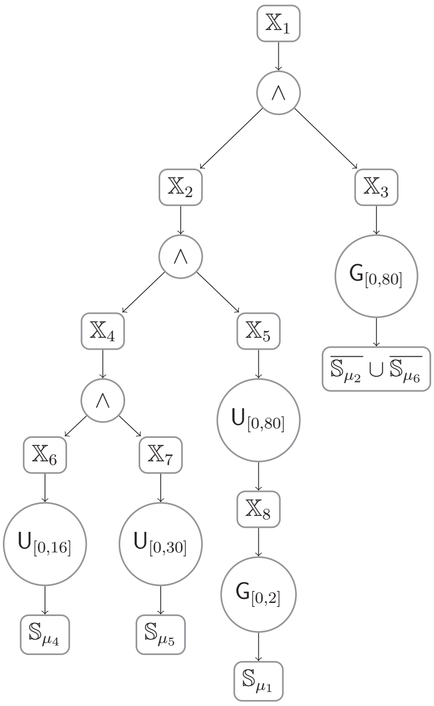


Figure 11. The constructed tTLT $\mathcal{T}_{\varphi_{\text{fast\_overtake}}}$.

an opposite direction in the other lane, Veh$_1$ must overtake Veh$_2$ for reaching $\mathbb{S}_{\mu_1}$ and avoid Veh$_3$ for safety.

We describe the dynamics of the vehicle Veh$_1$ as in Murgovski and Sjöberg (2015):

$$x_{k+1} = \underbrace{\begin{bmatrix} 1 & 0 & \delta \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{A} x_k + \underbrace{\begin{bmatrix} 0 & 0 \\ \delta & 0 \\ 0 & \delta \end{bmatrix}}_{B} u_k + w_k,$$

where $x_k = [p^x(k), p^y(k), v^x(k)]^T$, $u_k = [v^y(k), a^x(k)]^T$, and δ is the sampling period. The working space is $X = \{z \in \mathbb{R}^3 \mid [0, -5, -3]^T \leq z \leq [120, 5, 3]^T\}$, the control constraint set is $U = \{z \in \mathbb{R}^2 \mid [-1, -1]^T \leq z \leq [1, 1]^T\}$, the disturbance set is $W = \{z \in \mathbb{R}^3 \mid [-0.05, -0.05, -0.05]^T \leq z \leq [0.05, 0.05, 0.05]^T\}$, and the target region is $\mathbb{S}_{\mu_1} = \{z \in \mathbb{R}^2 \mid [115, -5, 0.5]^T \leq z \leq [120, 0, 0.5]^T\}$.

We use $\mathbb{S}_{\mu_2} = \{z \in \mathbb{R}^3 \mid [45, -5, -\infty]^T \leq z \leq [50, 0, \infty]^T\}$ to denote the state set that contains the occupancy of Veh$_2$. We describe the dynamics of the vehicle Veh$_3$ as

$$\bar{x}_{k+1} = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\bar{A}} x_k + \underbrace{\begin{bmatrix} \delta & 0 \\ 0 & \delta \end{bmatrix}}_{\bar{B}} \bar{u}_k,$$

where $x_k = [\bar{p}^x(k), \bar{p}^y(k)]^T$, $\bar{u}_k = [\bar{v}^x(k), \bar{v}^y(k)]^T$, We assume that it moves at a constant velocity $\bar{u}_k = [\bar{v}^x, 0]^T$. The initial state of Veh$_3$ is $\bar{x}_0 = [\bar{p}^x_{ini}, 2.5]^T$. Then, we have that its position of x-axis is $\bar{p}^x_k = \bar{p}^x_{ini} + \delta \times (k-1) \times \bar{v}^x$.

To formulate the overtaking task, we define the following three sets as shown in Figure 10: $\mathbb{S}_{\mu_3} = \{z \in \mathbb{R}^3 \mid [0, -5, -3]^T \leq z \leq [35, 0, 3]^T\}$, $\mathbb{S}_{\mu_4} = \{z \in \mathbb{R}^3 \mid [35, -5, -3]^T \leq z \leq [60, 5, 3]^T\}$, and $\mathbb{S}_{\mu_5} = \{z \in \mathbb{R}^3 \mid [60, -5, -3]^T \leq z \leq [120, 0, 3]^T\}$.

Let us choose the sampling period as $\delta = 0.2s$(seconds). To respect the time constraint and the input constraint for Veh$_1$, we consider two possible solutions to the previous reachability problem: (1) fast overtaking: overtake Veh$_2$ before Veh$_3$ passes Veh$_2$; (2) slow overtaking: wait until Veh$_3$ passes Veh$_2$ and then overtake Veh$_2$. The fast

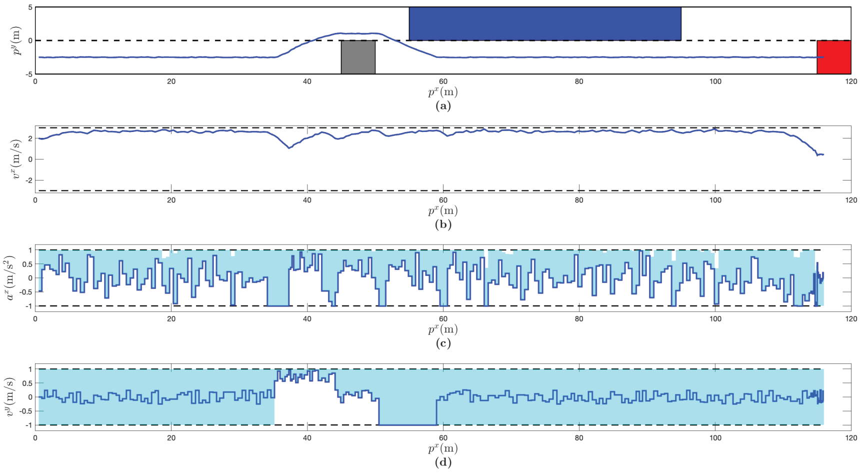


Figure 12. Trajectories for one realization of disturbance signal in the fast overtaking: (a) position trajectory; (b) velocity trajectory of x-axis; (c) control trajectory of x-axis; (d) control trajectory of y-axis.

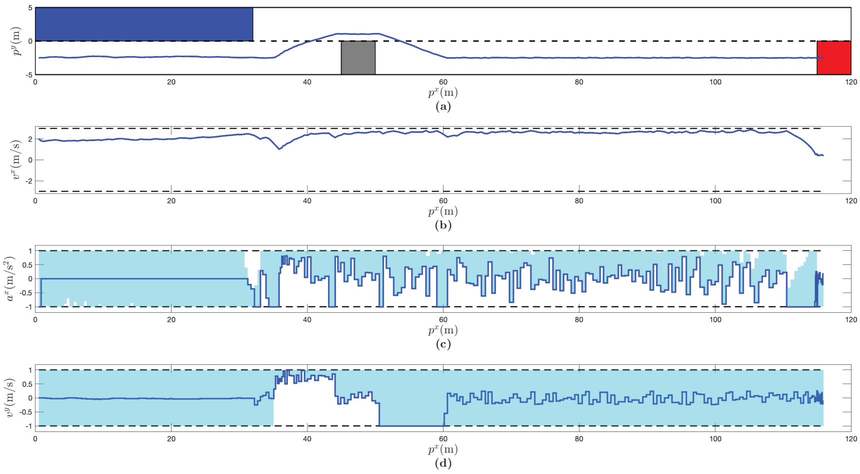


Figure 13. Trajectories for one realization of disturbance signal in the slow overtaking: (a) position trajectory; (b) velocity trajectory of x-axis; (c) control trajectory of x-axis; (d) control trajectory of y-axis.

overtaking can be encoded into an STL formula:

$$\begin{aligned}\varphi_{\text{fast\_overtake}} &= \mu_3 \mathsf{U}_{[0,16]}\mu_4 \wedge (\mu_3 \vee \mu_4)\mathsf{U}_{[0,30]}\mu_5 \\ &\wedge (\mu_3 \vee \mu_4 \vee \mu_5)\mathsf{U}_{[0,80]}\mathsf{G}_{[0,2]}\mu_1 \wedge \mathsf{G}_{[0,80]}\neg(\mu_2 \vee \mu_6),\end{aligned}$$

where $\mathbb{S}_{\mu_6} = \{z \in \mathbb{R}^6 \mid [\bar{p}^x(16), 0, -\infty]^T \leq z \leq [\bar{p}^x(0), 5, \infty]^T\}$. Note that $\mathbb{S}_{\mu_6}$ denotes the reachable set for the vehicle Veh$_3$ within the time interval $[0, 16]$ seconds and 16 (that corresponds to the sampling index $k = 80$) is the maximal time instant that the vehicle Veh$_1$ can reach the set $\mathbb{S}_{\mu_5}$ in the sprit of φ_1. Using Algorithm 1, one can construct the tTLT $\mathcal{T}_{\varphi_{\text{fast\_overtake}}}$ (see Figure 11), where

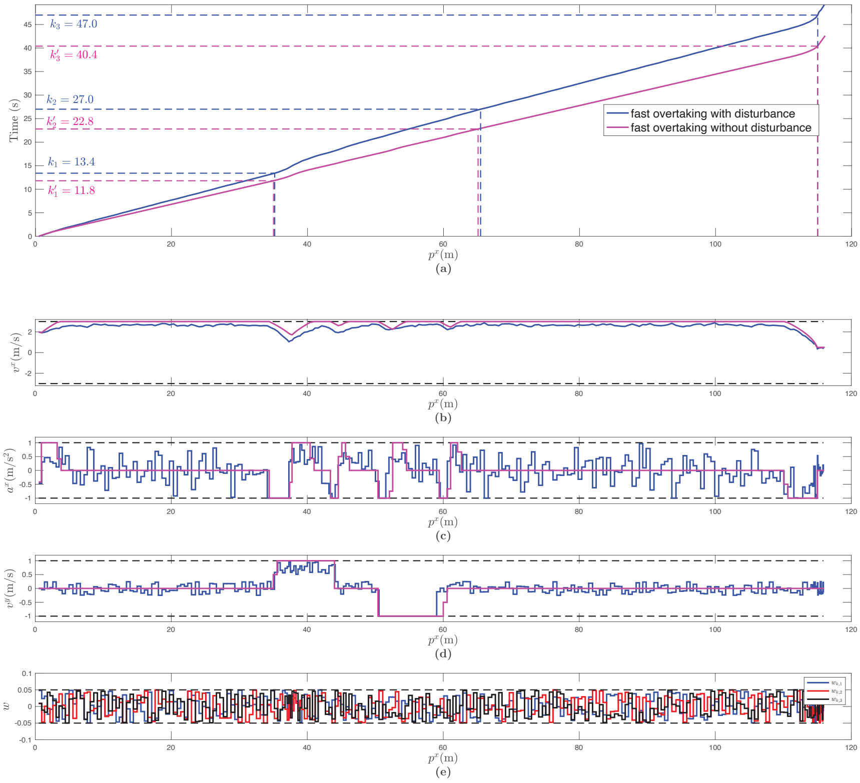
$$\begin{aligned}\mathbb{X}_6(t_k) &= \mathcal{R}^M(X, \mathbb{S}_{\mu_4}, \mathbb{S}_{\mu_3}, [0,16], k),\\ \mathbb{X}_7(t_k) &= \mathcal{R}^M(X, \mathbb{S}_{\mu_5}, \mathbb{S}_{\mu_3} \cup \mathbb{S}_{\mu_4}, [0,30], k),\\ \mathbb{X}_8(t_k) &= \overline{\mathcal{R}^m(X, \overline{\mathbb{S}_{\mu_1}}, [0,2], k)},\\ \mathbb{X}_4(t_k) &= \mathbb{X}_6(t_k) \cap \mathbb{X}_7(t_k),\\ \mathbb{X}_5(t_k) &= \mathcal{R}^M(X, \mathbb{X}_8(t_0), \mathbb{S}_{\mu_3} \cup \mathbb{S}_{\mu_4} \cap \mathbb{S}_{\mu_5}, [0,80], k),\\ \mathbb{X}_2(t_k) &= \mathbb{X}_4(t_k) \cap \mathbb{X}_5(t_k),\\ \mathbb{X}_3(t_k) &= \overline{\mathcal{R}^m(X, \mathbb{S}_{\mu_2} \cap \mathbb{S}_{\mu_6}, [0,80], k)}, \text{ and}\\ \mathbb{X}_1(t_k) &= \mathbb{X}_2(t_k) \cup \mathbb{X}_3(t_k).\end{aligned}$$

The slow overtaking can be encoded into an STL formula

$$\begin{aligned}\varphi_{\text{slow\_overtake}} &= \mu_3 \mathsf{U}_{[16,32]}\mu_4 \wedge (\mu_3 \vee \mu_4)\mathsf{U}_{[0,45]}\mu_5 \\ &\wedge (\mu_3 \vee \mu_4 \vee \mu_5)\mathsf{U}_{[0,80]}\mathsf{G}_{[0,2]}\mu_1 \wedge \mathsf{G}_{[0,80]}\neg(\mu_2 \vee \mu_7)\end{aligned}$$

where $\mathbb{S}_{\mu_7} = \{z \in \mathbb{R}^2 \mid [-\infty, 0, -\infty]^T \leq z \leq [\bar{p}^x(16), 5, \infty]^T\}$. Note that $\mathbb{S}_{\mu_7}$ denotes the reachable set for the vehicle Veh$_3$ within the time interval $[16, +\infty)$ and 16 (that corresponds to the sampling index $k = 80$) is the minimal time instant that the vehicle Veh$_1$ can reach the set $\mathbb{S}_{\mu_4}$ in the sprit of φ_2. The tTLT $\mathcal{T}_{\varphi_{\text{slow\_overtake}}}$ can be constructed similar to $\mathcal{T}_{\varphi_{\text{fast\_overtake}}}$.

In the following, two simulation cases are considered and the online control synthesis algorithm is implemented. In the fast overtaking, we choose the initial position $\bar{p}^x_{ini} = 95$



(a)

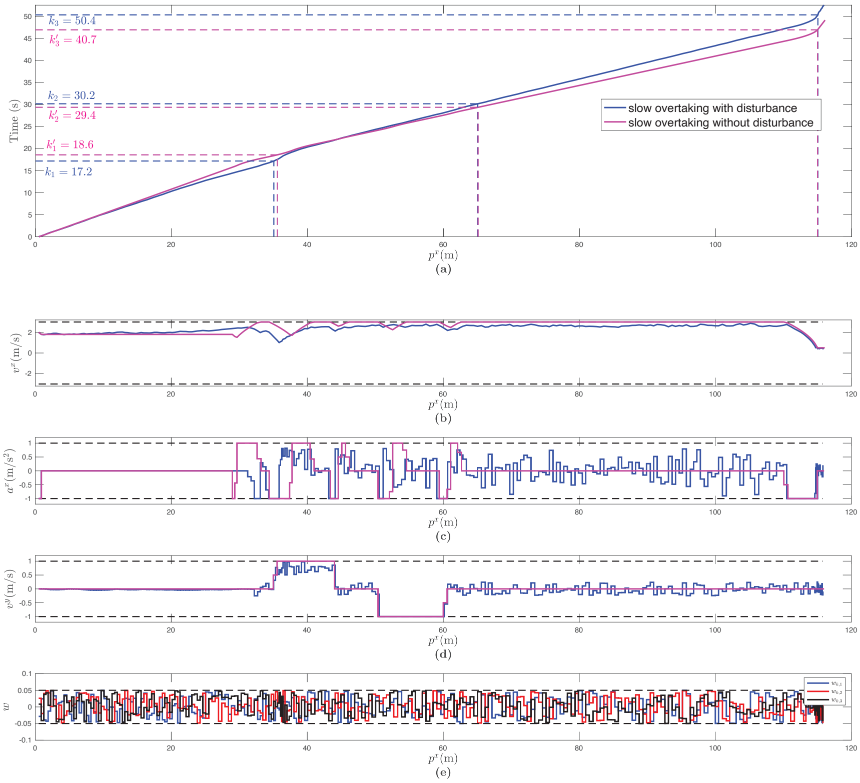
(b)

(c)

(d)

(e)

Figure 14. Comparison of robust control with noise and deterministic control without disturbance signal in the fast overtaking: (a) position-time trajectory; (b) velocity trajectory of x-axis; (c) control trajectory of x-axis; (d) control trajectory of y-axis; (e) disturbance signals.



(a)

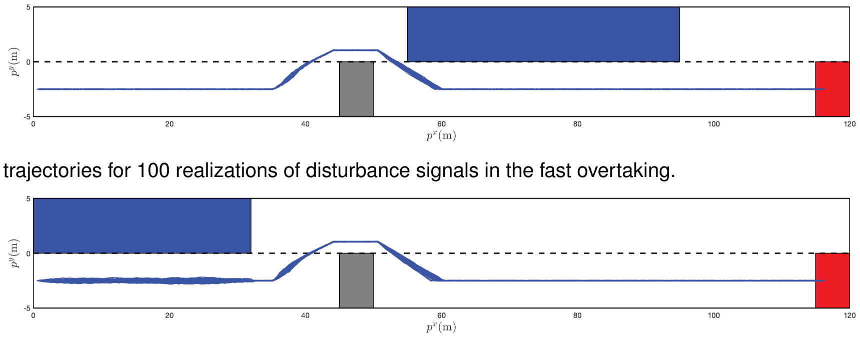
(b)

(c)

(d)

(e)

Figure 15. Comparison of robust control with noise and deterministic control without disturbance signal in the fast overtaking in the slow overtaking: (a) position-time trajectory; (b) velocity trajectory of x-axis; (c) control trajectory of x-axis; (d) control trajectory of y-axis; (e) disturbance signals.



(a) Position trajectories for 100 realizations of disturbance signals in the fast overtaking.

(b) Position trajectories for 100 realizations of disturbance signals in the slow overtaking.

Figure 16. Position trajectories for 100 realizations of disturbance signals.

and the moving velocity $\bar{v}^x = -2$ for the vehicle Veh$_3$ and the initial position $x_0 = [0.5, -2.5, 2]^T$ for Veh$_1$. One can verify that the specification $\varphi_{\text{slow\_overtake}}$ is infeasible in this case. Figure 12 (a) shows the position trajectories, from which we can see that the whole specification is fulfilled. The blue region denotes the set $\mathbb{S}_{\mu_6}$. Figure 12 (b) shows the velocity trajectory of v^x and Figures 12 (c)–(d) show the corresponding control inputs, where the dashed lines denote the control bounds. The cyan regions represent the synthesized control sets and the blue lines are the control trajectories. In the slow overtaking, we choose the initial position $\bar{p}^x_{ini} = 80$ and the moving velocity $\bar{v}^x = -3$ for the vehicle Veh$_3$ and the same initial position $x_0 = [0.5, -2.5]^T$ for Veh$_1$. In this case one can verify that $\varphi_{\text{fast\_overtake}}$ is infeasible. Figure 13 (a) shows the position trajectories, from which we can see that the whole specification is fulfilled. The blue region denotes the intersection between the set X and the set $\mathbb{S}_{\mu_7}$. Figure 13 (b) shows the velocity trajectory of v^x and Figures 13 (c)–(d) show the corresponding control input trajectories of a^x and v^y.

To highlight the effect of disturbances, we compare the trajectories with and without disturbances in the fast overtaking, which are shown in Figure 14(a)–(e). In Figure 14(a), we show the evolution of the x-axis position along the time. We use k_1, k_2, and k_3 (or k_1', k_2', and k_3') to denote the minimal time instants that Veh$_1$ reaches the sets $\mathbb{S}_{\mu_4}$, $\mathbb{S}_{\mu_5}$, and $\mathbb{S}_{\mu_1}$ for the noisy scenario (or for the deterministic scenario). We can see that the disturbances slightly delay the reaching time, while both two position trajectories satisfy the time intervals encoded in φ_1. The differences of the velocity trajectory of v^x and the corresponding control input trajectories of a^x and v^y are highlighted in Figures 13 (c)–(d), respectively. The disturance realizations of w_k are shown in Figure 13(e). In the deterministic scenario, the controller is aggressive in the sense that the velocity can actively reach the maximum velocity. As a comparison, the controller in the noisy scenario is more cautious in the sense that some gaps always exist between the actual velocity and the maximum velocity. In order to reject the disturbance, more frequent changing of the control inputs occurs in the noisy scenario. Similar observations are applied to the slow overtaking, whose comparisons are shown in Figure 15. Furthermore, in order to show the robustness, we run 100 realizations of the disturbance trajectories in the fast overtaking and in the slow overtaking, respectively. The position trajectories for such 100 realizations of two cases are shown in Figure 16.

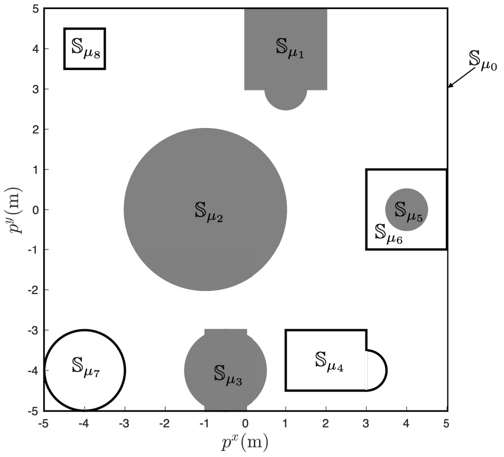


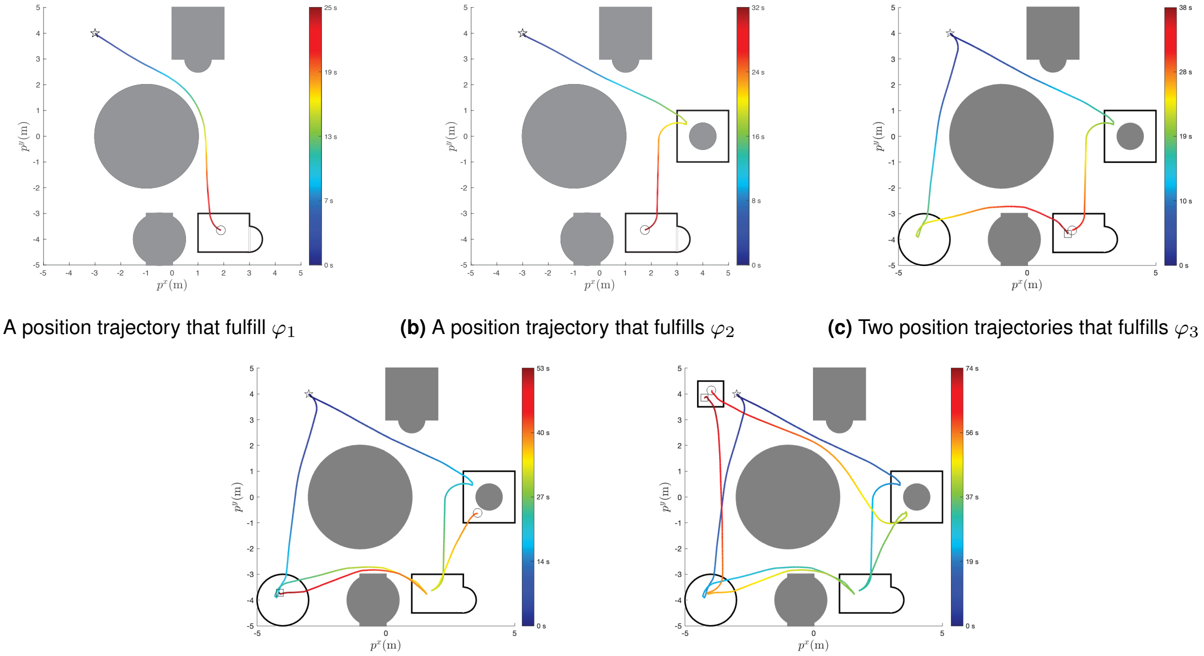
Figure 17. Scenario illustration: an automated vehicle needs to enter into the parking lot, park in the designated parking spot (blue), and leave the parking lot, while avoiding any collisions.

Finally, we report the computation time of this example, which was run in Matlab R2016a with MPT toolbox (Herceg et al., 2013) on a Dell laptop with Windows 7, Intel i7-6600U CPU 2.80 GHz and 16.0 GB RAM. We perform reachability analysis for constructing the tTLT offline, which takes 59.10 seconds. For online control synthesis, the minimal computation time at a single time step over 100 realizations is 0.23 seconds, while the maximal computation time is 1.07 seconds. The average time of each time step is 0.31 seconds. We remark that the mixed-integer formulation is difficult to implement in this example. This is because the computational complexity of mixed-integer programming grows exponentially with the horizon of the STL formula, which in this example reaches up to 400 sampling instants, much longer than the horizons considered in the simulation examples of Raman et al. (2015, 2014); Sadraddini and Belta (2015).

6.2 Motion planning example

In this section, we consider the motion planning of a mobile robot in an environment, as shown in Figure 17, under a group of STL specifications with growing complexity. We describe the underlying continuous dynamics of the

$$
\begin{aligned}
\varphi_1 &= (\mu_0 \wedge \neg(\mu_1 \vee \mu_2 \vee \mu_3))\, \mathsf{U}_{[0,30]} \mathsf{G}_{[0,2]} \mu_4 && \text{(10a)}\\
\varphi_2 &= \phi_{21} \wedge \phi_{22} && \text{(10b)}\\
\phi_{21} &= (\mu_0 \wedge \neg(\mu_1 \vee \mu_2 \vee \mu_3 \vee \mu_4 \vee \mu_5))\, \mathsf{U}_{[0,20]} \mathsf{G}_{[0,2]} \mu_6 &&\\
\phi_{22} &= \big((\mu_0 \wedge \neg(\mu_1 \vee \mu_2 \vee \mu_3))\, \mathsf{U}_{[0,35]} \mathsf{G}_{[0,2]} \mu_4\big) &&\\
\varphi_3 &= (\phi_{21} \vee \phi_{31}) \wedge \phi_{22} && \text{(10c)}\\
\phi_{31} &= (\mu_0 \wedge \neg(\mu_1 \vee \mu_2 \vee \mu_3 \vee \mu_4 \vee \mu_5))\, \mathsf{U}_{[0,20]} \mathsf{G}_{[0,3]} \mu_7 &&\\
\varphi_4 &= (\phi_{21} \wedge \phi_{22} \wedge \phi_{41}) \vee (\phi_{31} \wedge \phi_{22} \wedge \phi_{42}) && \text{(10d)}\\
\phi_{41} &= (\mu_0 \wedge \neg(\mu_1 \vee \mu_2 \vee \mu_3 \vee \mu_5))\, \mathsf{U}_{[35,55]} \mathsf{G}_{[0,2]} \mu_6 &&\\
\phi_{42} &= (\mu_0 \wedge \neg(\mu_1 \vee \mu_2 \vee \mu_3 \vee \mu_5))\, \mathsf{U}_{[35,55]} \mathsf{G}_{[0,3]} \mu_7 &&\\
\varphi_5 &= \varphi_4 \wedge \phi_{51} && \text{(10e)}\\
\phi_{51} &= (\mu_0 \wedge \neg(\mu_1 \vee \mu_2 \vee \mu_3 \vee \mu_5))\, \mathsf{U}_{[55,75]} \mathsf{G}_{[0,2]} \mu_8 &&
\end{aligned}
$$



(a) A position trajectory that fulfill φ_1

(b) A position trajectory that fulfills φ_2

(c) Two position trajectories that fulfills φ_3

(d) Two position trajectories that fulfills φ_4

(e) Two position trajectories that fulfills φ_5

Figure 18. The position trajectories that fulfill the STL formulas φ_i, $i = 1, \ldots, 5$. The time information is indicated using different colors.

automated vehicle as:

$$
f(x,u,w) = \begin{bmatrix} \dot{p}^x \\ \dot{p}^y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \sigma \end{bmatrix} + w,
$$

where $x = [p^x, p^y, \theta]^T$ is the vehicle's x position, y position, and heading, respectively. The control input is $u = [v, \sigma]^T$, where v is the vehicle's velocity and σ is the angular velocity. The working space is $X = \{z \in \mathbb{R}^3 \mid [-5,-5,-\pi]^T \le z \le [-5,5,\pi]^T\}$, the control set is $U = \{z \in \mathbb{R}^2 \mid [-0.5, -\pi/5]^T \le z \le [0.5, \pi/5,]^T\}$, and the disturbance set is $W = \{z \in \mathbb{R}^3 \mid [-0.1,-0.1,-0.1]^T \le z \le [0.1, 0.1, 0.1]^T\}$. For constructing the tTLT, we discretize the above dynamics using a simple zero-order hold estimation. Let Δ be the sampling period, then we describe the discrete dynamics of the automated vehicle as

$$
x_{k+1} = x_k + f(x_k, u_k, w_k)\Delta.
$$

For the parking task, we set $\Delta = 0.05$s.

We consider the following five STL formulas φ_i, $i = 1, \ldots, 5$, as defined in Eq.(10). These five formulas have increasing complexity, e.g., longer horizon and more operators. We report the computation time of this example, which was run in Matlab R2022b with the Level Set Method Toolbox (Mitchell and Templeton, 2005). The offline computation time for constructing the tTLT and the online computation time for synthesizing the controller are summarized in Table 2. As expected, the offline computation time typically increases with respect to the complexity of STL formulas. Note that the formulas φ_3 and φ_4 have the same computation time (134.49s) since the computation of reachable sets for ϕ_3 can be directly reused to construct the tTLT of ϕ_4, despite that ϕ_4 looks more complex than ϕ_3. On the other hand, the online computation for control synthesis, measured by the computation time per time step, is very

Table 2. Computation time under different STL formulas

STL formula	Offline com. time (tTLT construction)	Online com. time (control synthesis)
φ_1 in (10a)	71.60s	0.0180s
φ_2 in (10b)	82.70s	0.0178s
φ_3 in (10c)	134.49s	0.0193s
φ_4 in (10d)	134.49s	0.0159s
φ_5 in (10e)	187.81s	0.0164s

efficient for all the formulas. The position trajectories are plotted in Figure 18, where the initial position is indicated by the star and the end position is the circle. The time information over the trajectories is illustrated by the color map.

7 Car Parking Experiment

In this section, we consider a car parking example. This example will specify a parking task as an STL formula and then show how our algorithms perform on real hardware. We will first perform reachability analysis for constructing the tTLT offline and then we use the tTLT to synthesize a parking controller for the Small-Vehicles-for-Autonomoy (SVEA) platform (Jiang et al., 2022).

As shown in Figure 19, we consider a scenario where an automated vehicle must enter the parking lot $\mathbb{S}_{\mu_1}$, park in the designated parking spot $\mathbb{S}_{\mu_2}$, and leave the parking lot through the exit $\mathbb{S}_{\mu_4}$, where each step of the scenario has a specific deadline. Additionally, throughout the scenario, the vehicle must stay safe and avoid collisions with the parking lot walls and parked vehicles $\mathbb{S}_{\mu_3}$.

We describe the underlying continuous dynamics of the automated vehicle as:

$$f(x,u,w) = \begin{bmatrix} \dot{p}^x \\ \dot{p}^y \\ \dot{\theta} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v\tan\delta}{L} \\ a \end{bmatrix} + w, \tag{11}$$

where $x = [p^x, p^y, \theta, v]^T$ is the vehicle's x position, y position, heading, and velocity, respectively. $u = [\delta, a]^T$ is the vehicle's steering and acceleration inputs. The working space is $X = \{z \in \mathbb{R}^4 \mid [-2,-3,-\pi,-0.6]^T \leq z \leq [2,2,\pi,0.6]^T\}$, the control set is $U = \{z \in \mathbb{R}^2 \mid [-\pi/5,-0.5]^T \leq z \leq [\pi/5,0.5]^T\}$, and the disturbance set is $W = \{z \in \mathbb{R}^4 \mid [-0.01,-0.01,-\pi/72,-0.01]^T \leq z \leq [0.01,0.01,\pi/72,0.01]^T\}$. For constructing the tTLT, we discretize (11) using a simple zero-order hold estimation. Let δ be the sampling period, then we describe the discrete dynamics of the automated vehicle as

$$x_{k+1} = x_k + f(x_k,u_k,w_k)\Delta.$$

For the parking task, we set $\Delta = 0.05$s. We define the state sets in Figure 19 as $\mathbb{S}_{\mu_1} = \{z \in \mathbb{R}^4 \mid [-2,-3,-\pi,-0.6]^T \leq z \leq [2,0,\pi,0.6]^T\}$, $\mathbb{S}_{\mu_2} = \{z \in \mathbb{R}^4 \mid [1.3,-2,-\pi,-0.6]^T \leq z \leq [2,-1.5,\pi,0.6]^T\}$, $\mathbb{S}_{\mu_4} = \{z \in \mathbb{R}^4 \mid [0.5,0,-\pi,-0.6]^T \leq z \leq [1,1,\pi,0.6]^T\}$, and $\mathbb{S}_{\mu_3} = \mathbb{S}_{\mu_{3,1}} \cup \mathbb{S}_{\mu_{3,2}} \cup \mathbb{S}_{\mu_{3,3}}$, where $\mathbb{S}_{\mu_{3,1}} = \{z \in \mathbb{R}^4 \mid [-2,-3,-\pi,-0.6]^T \leq z \leq [-1.3,0,\pi,0.6]^T\}$, $\mathbb{S}_{\mu_{3,2}} = \{z \in \mathbb{R}^4 \mid [1.3,-1.5,-\pi,-0.6]^T \leq z \leq [2,0,\pi,0.6]^T\}$, $\mathbb{S}_{\mu_{3,3}} = \{z \in \mathbb{R}^4 \mid [1.3,-3,-\pi,-0.6]^T \leq z \leq [2,-2,\pi,0.6]^T\}$.

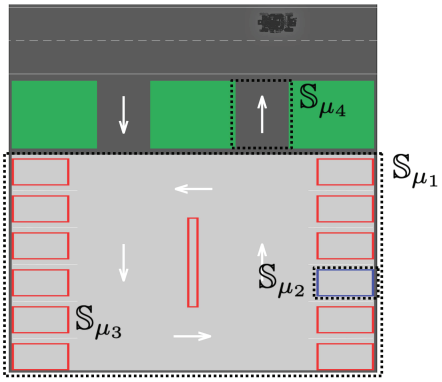


Figure 19. Scenario illustration: an automated vehicle needs to enter into the parking lot, park in the designated parking spot (blue), and leave the parking lot, while avoiding any collisions.

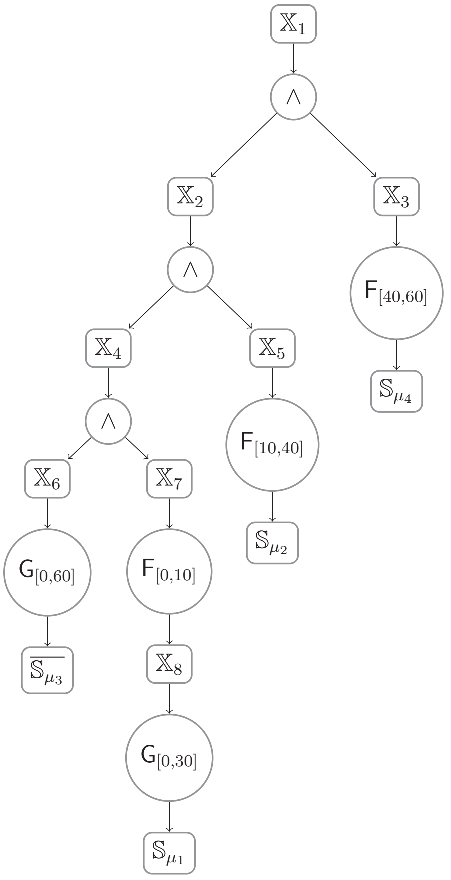


Figure 20. The constructed tTLT $\mathcal{T}_{\varphi_{\text{parking}}}$.

We let the full scenario be 60 seconds long and specify that the vehicle needs to enter the parking lot, park in the designated spot, and leave the parking lot within 10 seconds, 40 seconds, and 60 seconds, respectively. Then, this parking task can be encoded into the following STL formula:

$$\varphi_{\text{parking}} = \mathsf{G}_{[0,60]}\neg\mu_3 \wedge \mathsf{F}_{[0,10]}\mathsf{G}_{[0,30]}\mu_1 \wedge \mathsf{F}_{[10,40]}\mu_2 \\ \wedge \mathsf{F}_{[40,60]}\mu_4.$$

First, we use Algorithm 1 to construct the corresponding tTLT $\mathcal{T}_{\varphi_{\text{parking}}}$ (see Figure 20), where the tube nodes $\mathbb{X}_i, i =$

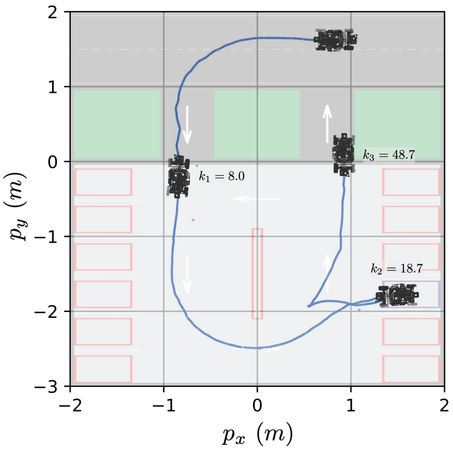


Figure 21. The position trajectory of a SVEA vehicle performing the parking task φ_{parking}.

$1, \cdots, 8$ are computed in a bottom-up manner as in the previous example. Then, we implement the online control synthesis algorithm (Algorithm 5) on a SVEA vehicle using $\mathcal{T}_{\varphi_{\text{parking}}}$. For choosing a control policy within the constraints of the synthesized control sets, we apply the same approach as described in Section IV.C of Jiang et al. (2020).

For our evaluation, we initialize the SVEA vehicle with the initial state of $x_0 = [1, 1.75, -\pi, 0]$. At this initial state, φ_3 is robustly satisfiable. Figure 21 shows the position trajectory, where one can see that the specification is fulfilled. In Figure 22, we show the control input trajectories for acceleration and steering. We use k_1, k_2, k_3 to denote the minimal time instants that the automated vehicle reaches sets $\mathbb{S}_{\mu_1}$, $\mathbb{S}_{\mu_2}$, and $\mathbb{S}_{\mu_4}$. Using the synthesized controller, the SVEA vehicle realized $k_1 = 8.0$, $k_2 = 18.7$, and $k_3 = 48.7$, as illustrated in both Figures 21 and 22, confirming the satisfaction of φ_{parking}. For our evaluation, we initialize the SVEA vehicle with the initial state of $x_0 = [1, 1.75, -\pi, 0]$. At this initial state, φ_3 is robustly satisfiable. Figure 21 shows the position trajectory, where one can see that the specification is fulfilled. In Figure 22, we show the control input trajectories for acceleration and steering. We use k_1, k_2, k_3 to denote the minimal time instants that the automated vehicle reaches sets $\mathbb{S}_{\mu_1}$, $\mathbb{S}_{\mu_2}$, and $\mathbb{S}_{\mu_4}$. Using the synthesized controller, the SVEA vehicle realized $k_1 = 8.0$, $k_2 = 18.7$, and $k = 48.7$, as illustrated in both Figures 21 and 22, confirming the satisfaction of φ_{parking}.

Finally, we report the computation time of this example, which was run in Matlab R2022b with the Level Set Method Toolbox (Mitchell and Templeton, 2005). We perform reachability analysis for constructing the tTLT offline on a Dell laptop with Ubuntu 20.04, Intel i7-4600U CPU 2.10GHz and 8.0 GB RAM, which takes 2371.81 seconds. We note that the offline computation time for constructing the tTLT can be significantly reduced by using the python implementation (Bui et al., 2022). Throughout the parking task, we perform the online control synthesis on an NVIDIA Jetson TX2 embedded computer onboard the SVEA vehicle. The average time step of the online control synthesis is 0.001 seconds. A video demonstration of this experiment can be found at `https://bit.ly/STL-TLT`.

8 Conclusion

A novel approach for the online control synthesis of uncertain discrete-time systems under STL specifications was proposed in this paper. First, a real-time version of STL semantics and a notion of tTLT were introduced. Then the formal semantic connection between an STL formula and its corresponding tTLT was derived, *i.e.*, a trajectory satisfying a tTLT also satisfies the corresponding STL formula. Finally, an online control synthesis algorithm was designed for the uncertain systems based on the connection between STL and tTLT. For the fragment of STL formulas under consideration, the soundness of the algorithm was proven. In the future, the control synthesis for multi-agent systems under local and/or global STL specifications is of interest.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the Swedish Research Council (VR) (Distinguished Professor Grant 2017-01078 and International Postdoc Grant 2021-06727), Knut and Alice Wallenberg Foundation (Wallenberg Scholar Grant and Wallenberg Academy Fellow), the ERC COG LEAFHOUND (Grant agreement ID: 864720), and the ERC ADG FUN2MODEL (Grant agreement ID: 834115).

References

Allen RE, Clark AA, Starek JA and Pavone M (2014) A machine learning approach for real-time reachability analysis. In: *Proceedings of IEEE/RSJ international conference on intelligent robots and systems*. pp. 2202–2208.

Althoff M (2015) An introduction to CORA 2015. In: *Proceedings of the Workshop on Applied Verification for Continuous and Hybrid Systems*.

Alur R, Feder T and Henzinger TA (1996) The benefits of relaxing punctuality. *Journal of the ACM* 43(1): 116–146.

Baier C and Katoen JP (2008) *Principles of Model Checking*. MIT press.

Baillieul J and Samad T (2021) *Encyclopedia of Systems and Control*. Springer.

Bansal S and Tomlin CJ (2021) Deepreach: A deep learning approach to high-dimensional reachability. In: *Proceedings of IEEE International Conference on Robotics and Automation*. pp. 1817–1824.

Barbosa FS, Duberg D, Jensfelt P and Tumova J (2019) Guiding autonomous exploration with signal temporal logic. *IEEE Robotics and Automation Letters* 4(4): 3332–3339.

Belta C, Bicchi A, Egerstedt M, Frazzoli E, Klavins E and Pappas GJ (2007) Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE Robotics & Automation Magazine* 14(1): 61–70.

Belta C, Yordanov B and Gol EA (2017) *Formal Methods for Discrete-time Dynamical Systems*, volume 89. Springer.

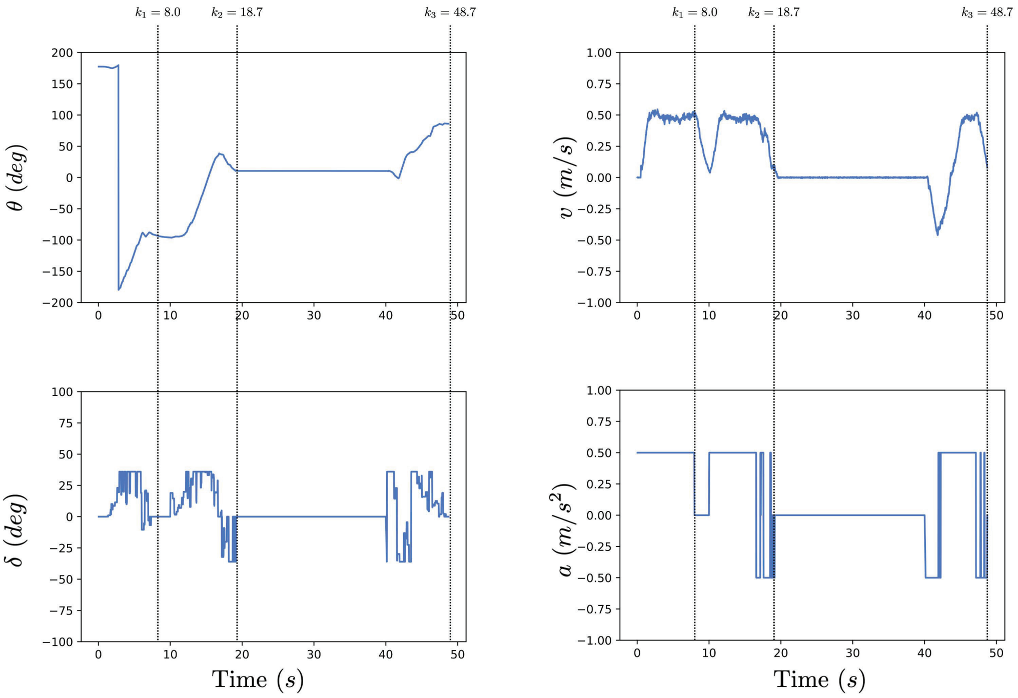


Figure 22. The velocity and heading trajectories in response to the acceleration and steering inputs throughout the parking task φ_{parking}.

Bertsekas D (1972) Infinite time reachability of state-space regions by using feedback control. *IEEE Transactions on Automatic Control* 17(5): 604–613.

Bui M, Giovanis G, Chen M and Shriraman A (2022) OptimizedDP: An efficient, user-friendly library for optimal control and dynamic programming. DOI:10.48550/ARXIV.2204.05520. URL https://arxiv.org/abs/2204.05520.

Buyukkocak AT, Aksaray D and Yazıcıoğlu Y (2021) Planning of heterogeneous multi-agent systems under signal temporal logic specifications with integral predicates. *IEEE Robotics and Automation Letters* 6(2): 1375–1382.

Buyukkocak AT, Aksaray D and Yazıcıoğlu Y (2022) Control barrier functions with actuation constraints under signal temporal logic specifications. In: *Proceedings of European Control Conference*.

Chen M, Herbert SL, Vashishtha MS, Bansal S and Tomlin CJ (2018a) Decomposition of reachable sets and tubes for a class of nonlinear systems. *IEEE Transactions on Automatic Control* 63(11): 3675–3688.

Chen M, Tam Q, Livingston SC and Pavone M (2018b) Signal temporal logic meets Hamilton-Jacobi reachability: connections and applications. In: *Proceedings of Workshop on Algorithmic Foundations of Robotics*.

Dokhanchi A, Hoxha B and Fainekos G (2014) On-line monitoring for temporal logic robustness. In: *Proceedings of International Conference on Runtime Verification*. pp. 231–246.

Fainekos GE and Pappas GJ (2009) Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science* 410(42): 4262–4291.

Farahani SS, Majumdar R, Prabhu VS and Soudjani S (2018) Shrinking horizon model predictive control with signal temporal logic constraints under stochastic disturbances. *IEEE Transactions on Automatic Control* 64(8): 3324–3331.

Fu J and Topcu U (2015) Computational methods for stochastic control with metric interval temporal logic specifications. In: *Proceedings of 54th IEEE Conference on Decision and Control*. pp. 7440–7447.

Gao Y, Abate A, Jiang FJ, Giacobbe M, Xie L and Johansson KH (2022) Temporal logic trees for model checking and control synthesis of uncertain discrete-time systems. *IEEE Transactions on Automatic Control* 67(10): 5071–5086.

Gastin P and Oddoux D (2001) Fast LTL to Büchi automata translation. In: *Proceedings of International Conference on Computer Aided Verification*. Springer, pp. 53–65.

Gilpin Y, Kurtz V and Lin H (2020) A smooth robustness measure of signal temporal logic for symbolic control. *IEEE Control Systems Letters* 5(1): 241–246.

Hamilton N, Robinette PK and Johnson TT (2022) Training agents to satisfy timed and untimed signal temporal logic specifications with reinforcement learning. In: *International Conference on Software Engineering and Formal Methods*. Springer, pp. 190–206.

Hashimoto W, Hashimoto K and Takai S (2022) Stl2vec: Signal temporal logic embeddings for control synthesis with recurrent neural networks. *IEEE Robotics and Automation Letters* 7(2): 5246–5253.

Herceg M, Kvasnica M, Jones CN and Morari M (2013) Multi-parametric toolbox 3.0. In: *Proceedings of European Control*

Conference. pp. 502–510.

Ho QH, Ilyes RB, Sunberg ZN and Lahijanian M (2022) Automaton-guided control synthesis for signal temporal logic specifications. In: *2022 IEEE 61st Conference on Decision and Control (CDC)*. pp. 3243–3249.

Jiang FJ, Al-Janabi M, Bolin T, Johansson KH and Mårtensson J (2022) SVEA: an experimental testbed for evaluating V2X use-cases. In: *Proceedings of IEEE 25th International Conference on Intelligent Transportation Systems*. pp. 3484–3489. DOI: 10.1109/ITSC55140.2022.9922544.

Jiang FJ, Gao Y, Xie L and Johansson KH (2020) Ensuring safety for vehicle parking tasks using Hamilton-Jacobi reachability analysis. In: *Proceedings of 59th IEEE Conference on Decision and Control*. pp. 1416–1421. DOI:10.1109/CDC42340.2020.9304186.

Kantaros Y and Zavlanos MM (2018) Sampling-based optimal control synthesis for multirobot systems under global temporal tasks. *IEEE Transactions on Automatic Control* 64(5): 1916–1931.

Kapoor P, Balakrishnan A and Deshmukh JV (2020) Model-based reinforcement learning from signal temporal logic specifications. *arXiv preprint arXiv:2011.04950* .

Karlsson J, Barbosa FS and Tumova J (2020) Sampling-based motion planning with temporal logic missions and spatial preferences. *IFAC-PapersOnLine* 53(2): 15537–15543.

Kochdumper N and Bak S (2023) Fully automated verification of linear time-invariant systems against signal temporal logic specifications via reachability analysis. *arXiv preprint arXiv:2306.04089* .

Koymans R (1990) Specifying real-time properties with metric temporal logic. *Real-Time Systems* 2(4): 255–299.

Kress-Gazit H, Lahijanian M and Raman V (2018) Synthesis for robots: Guarantees and feedback for robot behavior. *Annual Review of Control, Robotics, and Autonomous Systems* 1: 211–236.

Kurtz V and Lin H (2022) Mixed-integer programming for signal temporal logic with fewer binary variables. *IEEE Control Systems Letters* 6: 2635–2640.

Kurzhanski AB and Pravin V (2014) *Dynamics and Control of Trajectory Tubes: Theory and Computation*. Springer.

Leung K, Aréchiga N and Pavone M (2023) Backpropagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods. *The International Journal of Robotics Research* 42(6): 356–370.

Leung K and Pavone M (2022) Semi-supervised trajectory-feedback controller synthesis for signal temporal logic specifications. In: *2022 American Control Conference (ACC)*. pp. 178–185.

Lindemann L and Dimarogonas DV (2018) Control barrier functions for signal temporal logic tasks. *IEEE Control Systems Letters* 3(1): 96–101.

Lindemann L and Dimarogonas DV (2019) Feedback control strategies for multi-agent systems under a fragment of signal temporal logic tasks. *Automatica* 106: 284–293.

Lindemann L, Matni N and Pappas GJ (2021a) Stl robustness risk over discrete-time stochastic processes. In: *2021 60th IEEE Conference on Decision and Control (CDC)*. pp. 1329–1335.

Lindemann L, Pappas GJ and Dimarogonas DV (2021b) Reactive and risk-aware control for signal temporal logic. *IEEE Transactions on Automatic Control* 67(10): 5262–5277.

Liu W, Mehdipour N and Belta C (2021) Recurrent neural network controllers for signal temporal logic specifications subject to safety constraints. *IEEE Control Systems Letters* 6: 91–96.

Maler O and Nickovic D (2004) Monitoring temporal properties of continuous signals. In: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, pp. 152–166.

Mitchell IM and Templeton JA (2005) A toolbox of Hamilton-Jacobi solvers for analysis of nondeterministic continuous and hybrid systems. In: *Proceedings of International Workshop on Hybrid Systems: Computation and Control*. pp. 480–494.

Murgovski N and Sjöberg J (2015) Predictive cruise control with autonomous overtaking. In: *Proceedings of 54th IEEE Conference on Decision and Control*. pp. 644–649.

Raman V, Donzé A, Maasoumy M, Murray RM, Sangiovanni-Vincentelli A and Seshia SA (2014) Model predictive control with signal temporal logic specifications. In: *Proceedings of 53rd IEEE Conference on Decision and Control*. pp. 81–87.

Raman V, Donzé A, Sadigh D, Murray RM and Seshia SA (2015) Reactive synthesis from signal temporal logic specifications. In: *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. pp. 239–248.

Roehm H, Oehlerking J, Heinz T and Althoff M (2016) Stl model checking of continuous and hybrid systems. In: *Automated Technology for Verification and Analysis: 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings 14*. pp. 412–427.

Sadraddini S and Belta C (2015) Robust temporal logic model predictive control. In: *Proceedings of 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. pp. 772–779.

Scher G, Sadraddini S and Kress-Gazit H (2022) Robustness-based synthesis for stochastic systems under signal temporal logic tasks. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 1269–1275.

Singh NK and Saha I (2023) Stl-based synthesis of feedback controllers using reinforcement learning. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37. pp. 15118–15126.

Sun D, Chen J, Mitra S and Fan C (2022) Multi-agent motion planning from signal temporal logic specifications. *IEEE Robotics and Automation Letters* 7(2): 3451–3458.

Takayama Y, Hashimoto K and Ohtsuka T (2023) Signal temporal logic meets convex-concave programming: A structure-exploiting sqp algorithm for stl specifications. *arXiv preprint arXiv:2304.01475* .

van Huijgevoort BC, Verhoek C, Tóth R and Haesaert S (2023) Direct data-driven signal temporal logic control of linear systems. *arXiv preprint arXiv:2304.02297* .

Vasile CI and Belta C (2013) Sampling-based temporal logic path planning. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 4817–4822.

Vasile CI, Raman V and Karaman S (2017a) Sampling-based synthesis of maximally-satisfying controllers for temporal logic specifications. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 3840–3847.

Vasile CI, Raman V and Karaman S (2017b) Sampling-based synthesis of maximally-satisfying controllers for temporal logic specifications. In: *2017 IEEE/RSJ International*

Conference on Intelligent Robots and Systems (IROS). pp. 3840–3847.

Venkataraman H, Aksaray D and Seiler P (2020) Tractable reinforcement learning of signal temporal logic objectives. In: *Learning for Dynamics and Control*. PMLR, pp. 308–317.

Wolff EM and Murray RM (2016) Optimal control of nonlinear systems with temporal logic specifications. In: *Robotics Research*. Springer, pp. 21–37.

Yang G, Belta C and Tron R (2020) Continuous-time signal temporal logic planning with control barrier functions. In: *Proceedings of American Control Conference*. pp. 4612–4618.

Zhou Y, Maity D and Baras JS (2016) Timed automata approach for motion planning using metric interval temporal logic. In: *Proceedings of European Control Conference*. pp. 690–695.