2024 IEEE 63rd Conference on Decision and Control (CDC)
December 16-19, 2024. MiCo, Milan, Italy

Guaranteed Completion of Complex Tasks via Temporal Logic Trees and Hamilton-Jacobi Reachability

Frank J. Jiang[1], Kaj Munhoz Arfvidsson[1], Chong He[2], Mo Chen[2], Karl H. Johansson[1],

***Abstract*— In this paper, we present an approach for guaranteeing the completion of complex tasks with cyber-physical systems (CPS). Specifically, we leverage temporal logic trees constructed using Hamilton-Jacobi reachability analysis to (1) check for the existence of control policies that complete a specified task and (2) develop a computationally-efficient approach to synthesize the full set of control inputs the CPS can implement in real-time to ensure the task is completed. We show that, by checking the approximation directions of each state set in the temporal logic tree, we can check if the temporal logic tree suffers from the "leaking corner issue," where the intersection of reachable sets yields an incorrect approximation. By ensuring a temporal logic tree has no leaking corners, we know the temporal logic tree correctly verifies the existence of control policies that satisfy the specified task. After confirming the existence of control policies, we show that we can leverage the value functions obtained through Hamilton-Jacobi reachability analysis to efficiently compute the set of control inputs the CPS can implement throughout the deployment time horizon to guarantee the completion of the specified task. Finally, we use a newly released Python toolbox to evaluate the presented approach on a simulated driving task.**

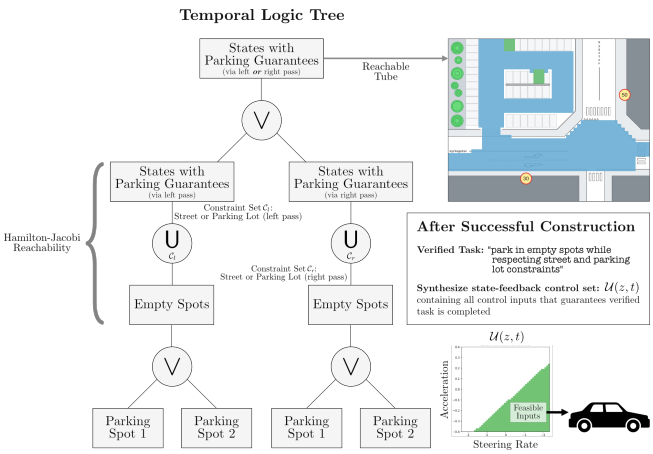

Fig. 1. We illustrate and annotate an example temporal logic tree that is used to guarantee the completion of a vehicle parking task.

I. Introduction

Over the past few decades, there has been a significant surge in interest towards the development of control techniques for CPS that offer formal safety and liveness guarantees. As CPS become more common in various applications, we need to ensure that these systems not only meet safety or task requirements, but are guaranteed to never violate them during deployment. This challenge has called for the proposal of rigorous methodologies around designing, validating, and implementing controllers for CPS in a way that ensures that safety and liveness is always fulfilled, even in varying or unpredictable environments.

Many of the developed approaches are based on safety/liveness filters or automata-based temporal logic approaches. For CPS, there is a large variety of safety/liveness filter-based control approaches [1], such as Hamilton-Jacobi reachability analysis-based approaches [2], [3], control barrier function-based approaches [4], [5], and zonotope-based approaches [6], [7]. Since safety/liveness filters are developed around the analysis of the dynamics propagation, the resultant controllers benefit from strong, low-level safety guarantees that take into account phenomenon such as the nonlinearity of the underlying dynamics or bounded disturbances. However, with many safety/liveness filter-based approaches, the tasks that are being solved are usually simple reach-avoid problems and the difficulty of the safety filter design can grow quickly as the complexity of the task grows. In contrast, automata-based temporal logic control approaches, such as [8]–[12], leverage the richness of temporal logic languages like linear temporal logic to specify, verify, and synthesize control policies for CPS. Although automata-based temporal logic approaches are powerful for working with complex tasks, the application of these approaches to nonlinear systems or disturbed systems can be impractical due to poor online scalability [13].

To combine the strengths of both approaches, there have been a number of proposals to combine safety/liveness filters with temporal logic over the recent years. In [14], authors explore the use of Hamilton-Jacobi reachability analysis to synthesize control sets for satisfying signal temporal logic specifications. In [15], authors explore the application of control barrier functions to efficiently synthesize control policies for a signal temporal logic fragment. More recently, authors introduce a tree-based computation model called

This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems, and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. It was also partially supported by the Swedish Research Council, Swedish Research Council Distinguished Professor Grant 2017-01078, the Knut and Alice Wallenberg Foundation Wallenberg Scholar Grant, and the Swedish Innovation agency (Vinnova), under grant 2021-02555 Future 5G Ride, within the Strategic Vehicle Research and Innovation program (FFI).

[1]F. J. Jiang, K. Munhoz Arfvidsson, and K. H. Johansson are with the Division of Decision and Control Systems, EECS, KTH Royal Institute of Technology, Malvinas väg 10, 10044 Stockholm, Sweden, email: {frankji, kajarf, kallej}@kth.se. They are also affiliated with the Integrated Transport Research Lab and Digital Futures.

[2]C. He and M. Chen are with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada, email: chong_he@sfu.ca, mochen@cs.sfu.ca.

temporal logic trees that directly utilizes backward reachability analysis to verify and synthesize control sets for linear temporal logic [13] and signal temporal logic [16]. While temporal logic trees have shown initial promise in CPS applications such as automated parking [17] and remote driving [18], there are still a number of challenges with the general application of temporal logic trees. Notably, one of the main challenges is explicitly synthesizing the set of control inputs that is guaranteed to satisfy the constructed temporal logic tree.

A. Contribution

The main contribution of this paper is an approach that efficiently synthesizes the least-restrictive set of control inputs that a CPS can implement to guarantee the completion of a specified task. To do this, we leverage the value functions resulting from Hamilton-Jacobi reachability analysis to efficiently compute least-restrictive control sets using a computation inspired by the work presented in [19]. These control sets are "least-restrictive," since they give the full set of inputs that the system can implement to stay within the computed satisfaction set. We start by detailing how to construct a temporal logic tree using Hamilton-Jacobi reachability analysis and show how we can check if there exist control policies that satisfy the constructed tree. Then, we develop a computationally efficient approach to synthesizing least-restrictive control sets from the value functions underlying the constructed temporal logic tree and evaluate the approach on a simulated driving task. Explicitly, the contributions of this paper can be summarized as follows:

1) we detail an algorithm to check for the existence of satisfying control policies for a constructed temporal logic tree,
2) we introduce a computationally-efficient approach for explicitly computing least-restrictive control sets from temporal logic trees,
3) we evaluate the methods presented in this paper on a simulated driving task using the newly open-sourced toolbox called "Python Specification and Control with Temporal Logic Trees" (pyspect)[1].

II. Preliminaries

In this section, we recall and introduce preliminary material that we use in the rest of the paper. Then, in the following section, we start using this material to clearly state the challenges and problems addressed in this work.

A. System Dynamics

In this work, we consider systems with the following control-affine dynamics

$$\dot{z} = f(z) + g(z)u, \tag{1}$$

where, $z \in \mathbb{R}^{n_x}$ and $u \in \mathcal{U} \subset \mathbb{R}^{n_u}$. f and g is uniformly continuous, bounded, and Lipschitz continuous in z. Given deployment time horizon T, we denote control functions as $u(\cdot) : [0, T] \to \mathcal{U}$, which we assume are measurable, and let $\mathbb{U}$ be the function space containing all $u(\cdot)$. Let $\zeta(t; z_0, t_0, u(\cdot)) \in \mathbb{R}^{n_x}$ be the state of system (1) at time t along a trajectory starting from initial state $\zeta(t_0; z_0, t_0, u(\cdot)) = z_0$ under $u(\cdot)$. For simplicity, we will sometimes write $\zeta(\cdot)$ to denote a trajectory of system (1).

B. Temporal Logic

In this section, we introduce the temporal logic we use to define complex tasks for system (1). In this work, we work with linear temporal logic (LTL). While LTL is a simpler logic compared to other popular logics like signal temporal logic, we choose to work with LTL in this work since we can express sufficiently complex tasks for our examples.

An LTL formula is defined over a finite set of atomic propositions $\mathcal{AP}$ with both logic and temporal operators. We can describe the syntax of LTL with:

$$\varphi ::= \textit{true} \mid p \in \mathcal{AP} \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \,\mathsf{U}\, \varphi_2,$$

where U denotes the "until" operators. By using the negation operator and the conjunction operator, we can define disjunction, $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$. Then, by employing the until operator, we can define: (1) eventually, $\Diamond\varphi = \textit{true}\,\mathsf{U}\,\varphi$ and (2) always, $\Box\varphi = \neg\Diamond\neg\varphi$. In this work, we omit the next operator $\bigcirc$, since we develop our approach using Hamilton-Jacobi reachability analysis for continuous time models like (1). Instead of working with LTL over infinite traces, we work with LTL over finite traces. The semantics for LTL over finite traces can be adapted from the semantics of the more common LTL over infinite traces by introducing a "last" time T and replacing ∞ with T [20]. This is particularly useful for working with general, nonlinear reachability analysis approaches as they typically do not compute or approximate infinite horizon reachable sets. In this work, we refer to T as the "deployment time horizon".

Definition 2.1: **(LTL semantics)** For an LTL formula φ, a trajectory $\zeta(\cdot)$, a deployment time horizon T, and a time instant check $0 \le t \le T$, the satisfaction relation $(\zeta(\cdot), t) \models \varphi$ is defined as

$$\begin{aligned}
&(\zeta(\cdot), t) \models p \in \mathcal{AP} \Leftrightarrow p \in l(\zeta(t)),\\
&(\zeta(\cdot), t) \models \neg\varphi \Leftrightarrow (\zeta(\cdot), t) \not\models \varphi,\\
&(\zeta(\cdot), t) \models \varphi_1 \wedge \varphi_2 \Leftrightarrow (\zeta(\cdot), t) \models \varphi_1 \wedge (\zeta(\cdot), t) \models \varphi_2,\\
&(\zeta(\cdot), t) \models \varphi_1 \vee \varphi_2 \Leftrightarrow (\zeta(\cdot), t) \models \varphi_1 \vee (\zeta(\cdot), t) \models \varphi_2,\\
&(\zeta(\cdot), t) \models \varphi_1 \,\mathsf{U}\, \varphi_2 \Leftrightarrow \exists t_1 \in [t, T] \text{ s.t.}\\
&\qquad\qquad \begin{cases} (\zeta(\cdot), t_1) \models \varphi_2,\\ \forall t_2 \in [t, t_1), (\zeta(\cdot), t_2) \models \varphi_1, \end{cases}\\
&(\zeta(\cdot), t) \models \Diamond\varphi \Leftrightarrow \exists t_1 \in [t, T], \text{ s.t. } (\zeta(\cdot), t_1) \models \varphi,\\
&(\zeta(\cdot), t) \models \Box\varphi \Leftrightarrow \forall t_1 \in [t, T], \text{ s.t. } (\zeta(\cdot), t_1) \models \varphi.
\end{aligned}$$

where p is an atomic proposition and $l(\cdot)$ is a labeling function defined as $l : \mathbb{R}^{n_x} \to 2^{\mathcal{AP}}$. For a proposition p, we define the following function for relating the proposition to a state set: $\mathcal{L}^{-1}(p) = \{z \in \mathbb{R}^{n_x} \mid p \in l(z)\}$. Using $l(\cdot)$ and $\mathcal{L}^{-1}(\cdot)$, we are able to associate sets in the state space of system (1) with atomic propositions.

[1] https://github.com/KTH-SML/pyspect

Definition 2.2: **(True Satisfaction Set)** For an LTL formula φ, we say Φ is φ's true satisfaction set when Φ is the largest set in $\mathbb{R}^{n_x}$ where $\forall z \in \Phi$, $\exists u(\cdot) \in \mathbb{U}$, $\forall t \in [0, T]$, $(\zeta(t; z, 0, u(\cdot)), t) \models \varphi$.

C. Temporal Logic Trees

Once we have specified an LTL specification for our system, we can use temporal logic trees (TLT) to check the feasibility and synthesize control sets for satisfying the specification. In Fig. 1, we illustrate an example of a TLT that is constructed for a parking task.

Definition 2.3: **(Temporal Logic Tree)** A temporal logic tree (TLT) is a tree for which each node is either a state set node corresponding to a subset of $\mathbb{R}^{n_z}$, or an operator node corresponding to one of the operators $\{\neg, \wedge, \vee, \mathsf{U}, \Box\}$; the root node and the leaf nodes are state set nodes; if a state set node is not a leaf node, its unique child is an operator node; the children of any operator node are state set nodes.

From a specified LTL formula, we can follow [13, Algorithm] to construct a temporal logic tree. Once the temporal logic tree is constructed, we can evaluate whether the LTL formula is satisfiable by checking whether the root of the temporal logic tree is an empty set or not [13, Theorem V.1].

1) Reachability Analysis: To construct a TLT, we need to compute the following types of reachable tubes.

Definition 2.4: **(Backward Reachable Tube)** Given system (1), a target set $\mathcal{T} \subseteq \mathbb{R}^{n_x}$, and a constraint set $\mathcal{C} \subseteq \mathbb{R}^{n_x}$, we define the backward reachable tube as

$$\begin{aligned}\mathcal{R}(\mathcal{T}; \mathcal{C}) = \{z \mid \exists u(\cdot) \in \mathbb{U}, \\ \exists \tau \in [0, T),\ \zeta(\tau; z, 0, u(\cdot)) \in \mathcal{T}, \\ \forall \tau' \in [0, \tau),\ \zeta(\tau'; z, 0, u(\cdot)) \in \mathcal{C}\},\end{aligned}$$

where $\mathcal{R}(\mathcal{T}; \mathcal{C})$ contains the set of states that are able to reach the target set $\mathcal{T}$ while staying within constraint set $\mathcal{C}$. For simplicity, we will denote this operation with $\mathcal{R}(\cdot)$.

Definition 2.5: **(Robust Control Invariant Set)** For system (1) and constraint set $\mathcal{C} \subseteq \mathbb{R}^{n_x}$, $\mathcal{RCI}(\mathcal{C}) \subseteq \mathbb{R}^{n_x}$ the largest robust control invariant set such that $\forall z \in \mathcal{RCI}(\mathcal{C})$ there $\exists u(\cdot) \in \mathbb{U}$ such that $\forall \tau \in [0, T]$, $\zeta(\tau; z, 0, u(\cdot)) \in \mathcal{C}$.

We denote the over- and under-approximation of the true backward reachable tube (or largest RCIS) with $\overline{\mathcal{R}}$ and $\underline{\mathcal{R}}$ (or $\overline{\mathcal{RCI}}$ and $\underline{\mathcal{RCI}}$), respectively.

D. Least-Restrictive Control Sets

After constructing TLTs, we can compute control sets that can be used to saturate the inputs to system (1) to guarantee that the verified task is completed. We define a least-restrictive control set as the following:

Definition 2.6: **(Least-Restrictive Control Set)** Given system (1), state $z \in \mathbb{R}^{n_x}$, time $t \in [0, T]$, a reachable tube $\mathcal{S}$, the least-restrictive control set is defined as

$$\begin{aligned}\mathcal{U}(z, t) = \{u(t) : u(\cdot) \in \mathbb{U}, \forall \tau \in [t, T], \\ \zeta(\tau; z, t, u(\cdot)) \in \mathcal{S}\}, \quad (2)\end{aligned}$$

where $\mathcal{U}(z, t)$ is a state and time feedback control set that contains all the control inputs system (1) is allowed to implement at state z and time t to stay within the reachable tube for the rest of the deployment time horizon.

III. Challenge: Completion of Complex Tasks

In this section, we clarify the specific problems we address in this work. To do this, we start by considering an example that will be evaluated at the end of this paper: vehicle parking. As is thoroughly discussed in [17], although vehicle parking may seem like a task that is simple, the complexity of the task easily grows in parking scenarios with stringent requirements. There may be varying speed requirements, hard-to-navigate driving spaces, multiple available spots, etc. As the complexity of the task grows or even changes in real-time (in the case where the parking environment changes), it's critically important we are able to guarantee the vehicle is able to safely and successfully complete it's parking task. One approach to this problem is constructing TLTs using Hamilton-Jacobi (HJ) reachability analysis [17], [18], [21]. A challenge faced by these works is that the least-restrictive control set is used implicitly and is not explicitly computed. Additionally, there is still no formal treatment for the challenge of checking whether there exists control policies that satisfy the constructed temporal logic tree. To address these two challenges, we solve the following problems.

Problem 3.1: Given system (1) and a TLT $\mathfrak{T}$ constructed using HJ reachability analysis for LTL task φ, guarantee that there exists control policies where system (1)'s resultant trajectory fully satisfies $\mathfrak{T}$.

Problem 3.2: Given that there exists control policies such that system (1) is able to satisfy the constructed $\mathfrak{T}$, efficiently utilize the value functions from the HJ reachability analysis to compute a least-restrictive control set $\hat{\mathcal{U}}$ that contains all of the control inputs system (1) can implement in real-time to guarantee it satisfies φ.

IV. Constructing TLT using HJ Reachability

In this section, we will detail the HJ reachability partial differential equations (PDE) that need to be solved for constructing the temporal logic tree. In particular, we introduce the computations involved with approximating the satisfaction sets of $\{\mathsf{U}, \Diamond, \Box\}$.

A. Key HJ PDEs

We define two key HJ PDEs that need to be solved to approximate solutions to $\mathcal{R}(\cdot)$ and $\mathcal{RCI}(\cdot)$ For the first PDE, with implicit target surface function $V_{\mathcal{T}}$ as our initial condition, we solve for solution $V_{\mathcal{R}}$ under implicit constraint surface function $V_{\mathcal{C}}$ in the constrained, reach HJ PDE

$$\begin{aligned}\max\{D_{t'} V_{\mathcal{R}}(z, t') + \min_{u \in \mathcal{U}} D_z V_{\mathcal{R}}(z, t') \cdot f(z, u), \\ -V_{\mathcal{C}}(z) - V_{\mathcal{R}}(z, t')\} = 0, \\ V_{\mathcal{R}}(z, 0) = V_{\mathcal{T}}(z),\end{aligned} \quad (3)$$

where $t' \leq 0$ is the time used to solve the PDE backwards in time. With $V_{\mathcal{C}}$ and $V_{\mathcal{T}}$ defined such that $\mathcal{C}$ and $\mathcal{T}$ are their respective zero sub-level sets (e.g. using a signed distance

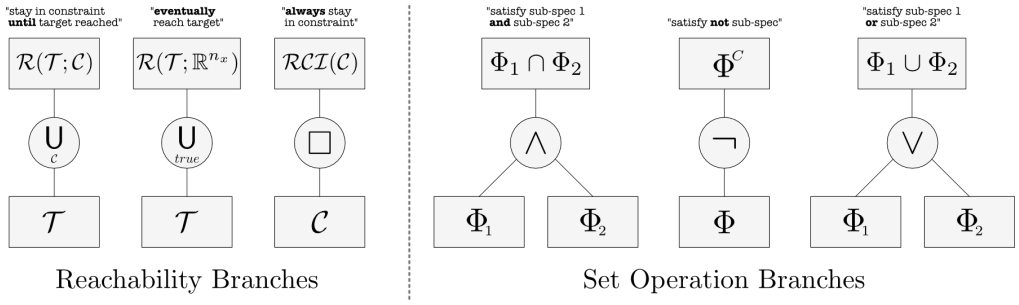

Fig. 2. We illustrate and describe all the different branches that can occur in a TLT and their corresponding state set and operator nodes.

function), we compute the backward reachable tube defined in Definition 2.4 as the zero sub-level set of the solution of (3) at time $t' = -T$, which we write explicitly as

$$\mathcal{R}(\mathcal{T};\mathcal{C}) = \{z \mid V_{\mathcal{R}}(z,-T) \leq 0\}. \tag{4}$$

Then, for the second PDE, it will be useful to introduce the concept of avoid tubes, although they are not always necessary for temporal logic trees.

Definition 4.1: **(Avoid Tube)** Given the system (1) and a set the system should stay outside of $\mathcal{O} \subset \mathbb{R}^{n_x}$, we define an avoid tube as

$$\mathcal{A}(\mathcal{O}) = \{z \mid \forall u(\cdot) \in \mathbb{U}, \exists \tau \in [0,T),\ \zeta(\tau; z, 0, u(\cdot)) \in \mathcal{A}\},$$

We then introduce the second HJ PDE that we solve to find avoid tubes. With the implicit avoid surface function $V_{\mathcal{O}}$ as an initial condition, we solve for $V_{\mathcal{A}}$ in the following avoid HJ PDE,

$$\begin{aligned} D_{t'}V_{\mathcal{A}}(z,t') + \max_{u\in\mathcal{U}} D_z V_{\mathcal{A}}(z,t')\cdot f(z,u) = 0, \\ V_{\mathcal{A}}(z,0) = V_{\mathcal{O}}(z), \end{aligned} \tag{5}$$

We can compute the avoid tube defined in Definition 4.1 as the zero sub-level set of the solution of (5) at time $t' = -T$, which we write explicitly as

$$\mathcal{A}(\mathcal{O}) = \{z \mid V_{\mathcal{A}}(z,-T) \leq 0\}. \tag{6}$$

Next, we show how we utilize (4) and (6) to approximate the satisfaction sets for $\{\,\mathsf{U}\,, \Diamond, \Box\}$ and verify that there exists control policies that satisfy each operator.

B. Approximating U (Until) and $\Diamond$ (Eventually)

Let φ_1 and φ_2 be two LTL sub-formulae. In the left-most branch in Fig. 2, we can see how $\varphi_1 \,\mathsf{U}\, \varphi_2$ looks as a TLT branch when $\mathcal{C}$ and $\mathcal{T}$ correspond to the satisfaction sets of φ_1 and φ_2, respectively. Now, let Φ_1, Φ_2, V_{φ_1}, V_{φ_2} be the satisfaction state sets and surface functions corresponding to LTL formulae φ_1 and φ_2. Then, we approximate $\varphi_1 \,\mathsf{U}\, \varphi_2$ based on (4) by finding

$$\mathcal{R}(\Phi_2;\Phi_1) = \{z \mid V_{\mathcal{R}}(z,-T) \leq 0\}. \tag{7}$$

In other words, the until operator corresponds to solving a constrained backward reachability problem with the target as the satisfaction set of φ_2 and with the constraint as the satisfaction set of φ_1. Since $\Diamond\varphi_2 = true \,\mathsf{U}\, \varphi_2$, we perform a similar computation to (7) to find the satisfaction set for the eventually operator. However, since Φ_1 becomes the satisfaction set of $true$, we end up solving an unconstrained backward reachability problem, as is illustrated in the middle reachability branch in Fig. 2. For most solvers, this is equivalent to dropping $V_{\mathcal{C}}(z)$ in (3). Depending on how many ancestor $\neg$ operator nodes the U node has, we may need to change whether we under- ($\underline{\mathcal{R}}(\cdot)$) or over-approximate ($\overline{\mathcal{R}}(\cdot)$) solutions to (4). This can be done numerically in level-set methods for solving (4) [2].

C. Approximating $\Box$ (Always)

Let φ and $\Phi \subset \mathbb{R}^{n_x}$ be an LTL sub-formula and the corresponding satisfaction set. In Definitions 2.5, $\mathcal{RCI}(\cdot)$ is defined around the existence of a control policy $u(\cdot)$ that keeps system (1) within a set for all time. Since this is consistent with the satisfaction requirement of $\Box\varphi$, to approximate the satisfaction set of $\Box\varphi$ we can find an approximate solution to $\mathcal{RCI}(\Phi)$. In Fig. 2, we can see how $\Box\varphi$ looks as a TLT branch when $\mathcal{C} = \Phi$. For approximating $\mathcal{RCI}(\Phi)$, we utilize the avoid tube defined in Definition 4.1. Instead of directly approximating the largest set of states where there exists control policies that keep the system in Φ, it is more common to approximate the set of states where for all control policies the system is forced outside of Φ and taking the complement of this set. In other words, we compute $\mathcal{RCI}(\cdot)$ as the following:

$$\mathcal{RCI}(\Phi) = \mathcal{A}^C(\Phi^C). \tag{8}$$

Based on (6) and with $\mathcal{O} = \Phi^C$, we approximate $\Box\varphi$ with

$$\mathcal{RCI}(\Phi) = \{z \mid V_{\mathcal{A}}(z,-T) > 0\}. \tag{9}$$

Similar to the U operator, depending on how many ancestor $\neg$ operator nodes the $\Box$ node has, we may need to change whether we under- ($\underline{\mathcal{RCI}}(\cdot)$) or over-approximate ($\overline{\mathcal{RCI}}(\cdot)$) solutions, which is done by numerically over- or under-approximating (6), respectively.

V. Computing Least-Restrictive Control Sets

In this section, we present our approach for efficiently computing least-restrictive control sets from a TLT constructed using HJ reachability analysis. We start by introducing an algorithm for quickly checking if control policies still exist for satisfying the TLT. Then, we introduce the explicit computation of the least-restrictive control sets that enable a system to follow the existing control policies.

Algorithm 1 `ctrlExists`

Input: Root node $\mathfrak{R}$ of a constructed TLT
Output: E, O, U, or I

```
if isLeaf(𝔑) then
    return approxDirection(𝔑) # E, O, U
end if
c = child(𝔑)
G = grandChildren(𝔑)
if length(G) == 1 then
    g = G
    a = ctrlExists(g)
    if a == I then return I
    # check operator nodes with single child
    if c == U or c == □ then
        a_o = approxDirection(c)
        if a_0 != a then return I else return a
    else if c == ¬ then
        return -1 * a
    end if
else
    g_1, g_2 = G
    a_1, a_2 = ctrlExists(g_1), ctrlExists(g_2)
    if a_1 == I or a_2 == I then return I
    # check operator nodes with two children
    if c == ∧ then
        if a_1==U or a_2==U then return I else return O
    else if c == ∨ then
        if a_1 == a_2 then return a_1
        else if a_1 == E then return a_2
        else if a_2 == E then return a_1
        else return Invalid
        end if
    end if
end if
```

A. Checking for existing control policies

From [13], we know that the constructed TLT for an LTL specification φ should under-approximate the true satisfaction set of φ. However, as is emphasized in [13, Theorem V.1] the under-approximation is tied to the existence of control policies that satisfy the constructed TLT. In some applications or tasks, this can be obvious to the designer. However, in many applications or tasks, this should be automatically checked. One of the key challenges with automatically checking and ensuring that there exists control policies that satisfy the constructed TLT is the so-called "leaking corner problem." The leaking corner problem occurs when two satisfaction sets are intersected. This problem also arises in the context of system decomposition [19], [22] and classic reach-avoid problems [23]. In the construction of TLTs, this problem is further complicated by the fact that in an LTL formula, there may be many nested $\{\neg, \wedge, \vee\}$, which all have requirements and effects around approximation directions. To address this, we start by characterizing the effect of the set operations underlying $\{\neg, \wedge, \vee\}$.

Lemma 5.1: **(Effect of $\neg$)** Let φ be an LTL subformula, $\Phi \subset \mathbb{R}^{n_x}$ be φ's true satisfaction set. Similarly, let Φ' be the true satisfaction set of $\neg\varphi$. Also, let $\hat{\Phi}$ be either an over- or under-approximation of Φ.

- Let $\hat{\Phi}$ be an over-approximation of Φ ($\hat{\Phi} \supset \Phi$), then $\hat{\Phi}^C$ is an under-approximation of Φ' ($\hat{\Phi}^C \subset \Phi'$).
- Let $\hat{\Phi}$ be an under-approximation of Φ ($\hat{\Phi} \subset \Phi$), then $\hat{\Phi}^C$ is an over-approximation of Φ' ($\hat{\Phi}^C \supset \Phi'$).

Algorithm 2 `leastRestrictiveCtrl`

Input: state $z \in \mathbb{R}^{n_x}$, time t, and constructed TLT $\mathfrak{T}$
Output: least-restrictive control set $\mathcal{U}_{z,t} \subseteq \mathcal{U}$ for z and t

```
C = controlTree(z, t, 𝔗) # [13, Algorithm 4]
Ĉ = compressTree(C) # [13, Algorithm 2]
U_{z,t} = setBacktrack(Ĉ) # [13, Algorithm 5]
return U_{z,t}
```

Proof: Proof in [24, Appendix I]. ■

In other words, the $\neg$ operator reverses the approximation direction.

Lemma 5.2: **(Effect of $\wedge$)** Let $\varphi = \varphi_1 \wedge \varphi_2$, and Φ, Φ_1, Φ_2 be their respective, true satisfaction sets. Then,

$$\Phi \subseteq \Phi_1 \cap \Phi_2 \tag{10}$$

Proof: Proof in [24, Appendix I]. ■

In other words, the set intersection underlying $\wedge$ can lead to over-approximation of the true satisfaction set. This is the key leaking corner issue, as we want the constructed TLT to under-approximate the true satisfaction set. However, this issue can be remedied if the $\wedge$ operator node has an $\neg$ operator node as an ancestor.

Corollary 5.1: **(Effect of $\neg$ on $\wedge$)** Let $\varphi = \varphi_1 \wedge \varphi_2$, and Φ, Φ_1, Φ_2 be their respective, true satisfaction sets. Then, we get the following under-approximation of $\neg\varphi$

$$\Phi^C \supseteq (\Phi_1 \cap \Phi_2)^C \tag{11}$$

Proof: Seen through Lemma 5.1 and Lemma 5.2. ■

In other words, if an $\neg$ is applied to an $\wedge$, the result is an under-approximation. This means that the resulting set only contains states where there exist control policies that satisfy the corresponding sub-formula.

Remark 5.1: Within the scope of this work, we will propose an algorithm for just checking if the leaking corner issue affects the constructed TLT. However, there are a couple other approaches to treat this problem: (1) additional assumptions are made to ensure $\wedge$ results in an exact result, and (2) post-processing of the $\wedge$ operator's underlying set intersection to ensure result is exact or an under-approximation. In some cases, the first approach is viable and an assumption can be made (i.e. one child is the subset of the other child) where leaking corner problems is avoided. However, for cases when the leaking corner problem cannot be avoided, an important future work will be to develop and incorporate new, computationally-efficient algorithms for the second approach.

Lemma 5.3: **(Effect of $\vee$)** Let $\varphi = \varphi_1 \vee \varphi_2$, and Φ, Φ_1, Φ_2 be their respective, true satisfaction sets. Then,

$$\Phi \equiv \Phi_1 \cup \Phi_2 \tag{12}$$

Proof: Proof in [24, Appendix I]. ■

This means that there is no contribution of additional approximation from $\vee$ operators when taking the union of the underlying satisfaction sets. That said, there are requirements on the children of $\vee$, which are outlined in Algorithm 1.

Now that we have characterized the individual effects of the $\{\neg, \wedge, \vee\}$ operators on the existence of control policies, we present Algorithm 1. In Algorithm 1, we recurse through

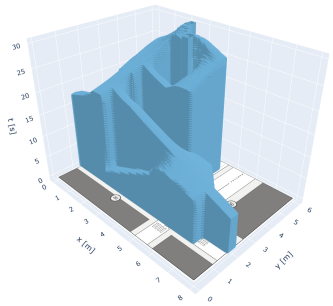

Fig. 3. The full, under-approximating satisfaction set for the parking task computed by constructing the temporal logic tree using HJ reachability analysis

the TLT and consider the numerical approximations underlying $\{\mathsf{U}, \Box\}$ and the approximation effects of $\{\neg, \wedge, \vee\}$ to decide whether control policies exist the satisfy the full TLT. For the output of the algorithm, we enumerate the variables $\mathtt{E} = 0$, $\mathtt{O} = +1$, $\mathtt{U} = -1$, and $\mathtt{I} = \mathtt{NaN}$ corresponding to exact, over-, under-approximation, and invalid, respectively.

Theorem 5.1: Let φ be an LTL formula and $\mathfrak{R}$ be the non-empty root node of a constructed TLT $\mathfrak{T}$ for verifying φ. If $\mathtt{ctrlExists}(\mathfrak{R})$ returns $\mathtt{U}$, then there exists control policies that satisfy $\mathfrak{T}$.

Proof: Proof in [24, Appendix I]. ■

B. Synthesizing Least-Restrictive Control Sets

Now, to synthesize the least-restrictive control set for full TLT, we follow [13, Algorithm 3]. For clarity, we include an adapted version in Algorithm 2. We start by computing the individual least-restrictive control sets for U and $\Box$ operator nodes (Line 11 and 23 in [13, Algorithm 4]). This is done by computing the following control set using the value function V_Φ from performing HJ reachability analysis:

$$\mathcal{U}_\Phi(z,t) = \{u \in \mathcal{U} \mid D_t V_\Phi(z, -T+t) + D_z V_\Phi(z, -T+t)^\top (f(z) + g(z)u) \leq 0\}. \quad (13)$$

For $0 \leq t \leq T$, let the computational time step be δt, and $s = -T + t$. The value function can be modified with first-order Taylor expansion:

$$V_\Phi(\zeta(s+\delta t; z, s), s+\delta t) = V_\Phi(z,s) + D_t V_\Phi(z,s)\delta t + D_z V_\Phi(z,s)^\top (f(z) + g(z)u)\delta t \leq 0. \quad (14)$$

The above equation implies that the satisfaction of Φ induces the least restrictive control set as the half-space:

$$a + b^\top u \leq 0 \quad (15)$$

with

$$a = V_\Phi(z,s) + D_t V_\Phi(z,s)\delta t + D_z V_\Phi(z,s)^\top f(z)\delta t$$
$$b^\top = D_z V_\Phi(z,s)^\top g(z)\delta t$$

This results in a least-restrictive control set that is an additional control constraint on top of any original control constraints, such as control bounds. Readers can find a code snippet for computing this control set using the odp toolbox [25] in [24, Appendix II].

Proposition 5.1: Satisfaction of Φ is guaranteed if $V_\Phi(z,s) \leq 0$. In this case, the state z at time s has a nonempty feasible control set.

Proof: This is a well-known result (for example, see [26]), and in the "reach" case follows immediately from Eq. (3). The first argument of the max operator, $D_{t'} V_\mathcal{R}(z,t') + \min_{u\in\mathcal{U}} D_z V_\mathcal{R}(z,t') \cdot f(z,u)$, is the total time derivative of $V_\mathcal{R}$ when the optimal control is applied. Thus, Eq. (3) implies that $V_\mathcal{R}$ is non-increasing along optimal trajectories. It is also known that $V_\mathcal{R}$ exists and is unique [27], [28], so by construction, if $V_\mathcal{R}(z,s) \leq 0$, there must exist a control to keep the value of $V_\mathcal{R}$ non-positive. This can also be shown in a similar argument involving $V_\mathcal{A}$ in the "avoid case", following Eq. (5). We conclude the proof by noting that $V_\Phi(z,s)$ is equal to either $V_\mathcal{R}$ or $V_\mathcal{A}$ depending on Φ. ■

Finally, we can show that the output of Algorithm 2 will result in the guaranteed completion of the specified task.

Theorem 5.2: Let φ be an LTL formula and $\mathfrak{T}$ be the constructed TLT that passes $\mathtt{ctrlExists}$. If at $t = 0$, $z(0)$ is in the root node's state set and $\forall t \in [0, T]$, system (1) implements $u(t) \in \mathcal{U}_{z(t),t}$, where $\mathcal{U}_{z(t),t}$ is the output of $\mathtt{leastRestrictiveCtrl}(z(t), t, \mathfrak{T})$, then system (1) is guaranteed to satisfy φ.

Proof: Since the system starts at time $t = 0$ with $z(0)$ in $\mathfrak{T}$'s root node's state set, then we know the system initially satisfies φ since the root node's state set under-approximates the true satisfaction set of φ. In Algorithm 2, when the control tree is synthesized, each state set node in $\mathfrak{T}$ is replaced by least-restrictive control sets (13) or the union/intersection of control sets (13). We know from Proposition 5.1 that the individual least-restrictive control sets that are the parents of U and $\Box$ operator nodes in the control tree are nonempty feasible control sets and ensure the system satisfies the corresponding LTL sub-formula. Since we know $\mathfrak{T}$ passes $\mathtt{ctrlExists}$, we know that $\mathfrak{T}$ does not have leaking corners and the combined least-restrictive control sets will also be nonempty feasible control sets. Since $\mathtt{compressTree}$ and $\mathtt{setBacktrack}$ only apply unions to the least-restrictive control sets up $\mathfrak{T}$, the final output $\mathcal{U}_{z,t}$ of $\mathtt{setBacktrack}$ is also the nonempty feasible control set containing all of the control inputs the system can implement to satisfy φ at state z and time t. ■

Now that we are able to check the existence of satisfying control policies and synthesize least-restrictive control sets for a specified task, we illustrate and evaluate our approach on a simulated driving task in the next section.

VI. Simulated Driving Example

In this section, we apply the presented approach to a simulated driving task where a vehicle is tasked to park into a parking lot on a road network inspired by the Kyrkogatan-Nygatan intersection in Eskilstuna, Sweden. For this task, the vehicle starts from somewhere in the road network nearby the parking lot and while it's entering the parking lot, it's task will be changed due to an unplanned blocking of one part of

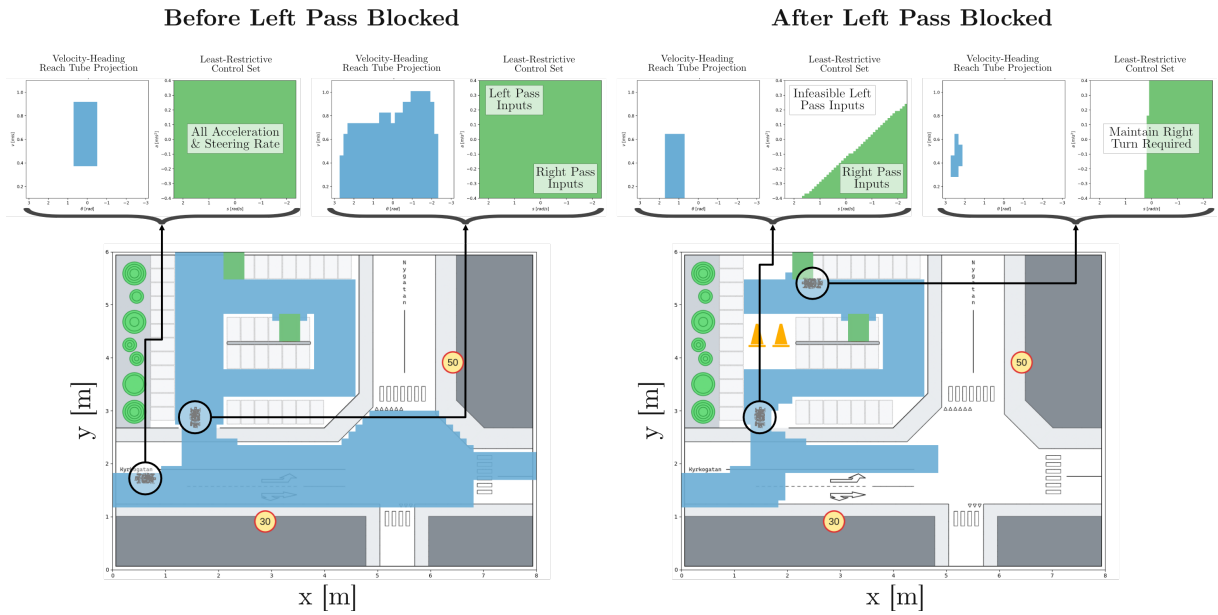

Fig. 4. We illustrate the deployment of the simulated vehicle for the specified parking task. In the two larger plots, we show the full environment and an xy projection (in blue) of the 5D satisfaction state set of the constructed TLT. In the smaller plots, we show the velocity-heading projection of the constructed TLT's satisfaction set (in blue) and the computed least-restrictive control set (in green) for different states and times throughout the task completion.

the parking lot. With this example, we illustrate the resultant least-restrictive control sets that are computed during the deployment to show case how the sets correctly constrain the vehicle's acceleration and steering rates to guarantee the parking task is completed. For this example, we utilize the newly released pyspect toolbox, where the code for the example itself can also be found.

A. Nonlinear, 5-state Vehicle Model

For the vehicle, let $z = [x, y, \theta, \delta, v]^\top$ be the state, where x, y, θ, δ, and v are the vehicle's x-position, y-position, heading angle, steering angle, and velocity, respectively. Then, let $u = [s, a]^\top$ be the input, where s and a are the steering rate and acceleration inputs into the vehicle, respectively. Explicitly, we write the dynamics as the following:

$$f(z) = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \frac{v\tan\delta}{L} \\ 0 \\ 0 \end{bmatrix}, \quad g(z) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix},$$

where L is the wheel-base length of the vehicle.

B. Task Specification

When constructing the task specification, we start by defining atomic propositions that describe this environment in terms of road geometries and other state constraints. For convenience, we denote state inequalities using propositions like $x_{<c}$ where $L'(x_{<c}) \equiv \{z \in \mathbb{R}^{n_x} \mid x < c\}$. Then, we define sub-formulae such as *Kyrkog. Geometry*, *P-Lot Geometry*, etc. that describe the drivable space. We also embed any speed limits and heading constraints into these formulae, for instance,

$$\begin{aligned} \textit{Kyrkogatan} = \textit{Kyrkog. Geometry} \\ &\wedge (y_{<3} \to v_{>0.4} \wedge v_{<1.0}) \\ &\wedge (y_{\geq 3} \to v_{>0.3} \wedge v_{<0.6}) \\ &\wedge (y_{<1.8} \to \theta_{>\frac{+\pi}{5}} \wedge \theta_{<\frac{-\pi}{5}}) \\ &\wedge (y_{\geq 1.8} \to \theta_{>\frac{+4\pi}{5}} \vee \theta_{<\frac{-4\pi}{5}}). \end{aligned}$$

For the exact definitions and values, we refer readers to the implementation available in pyspect.

The task is for the vehicle to park in one of the two empty parking spots. Namely, the target is *Empty Spots* = *Parking Spot 1* $\vee$ *Parking Spot 2*. In practice, picking between these can relate to which is closest or is most accessible. Furthermore, the sub-formulae

$$\varphi_L = (\textit{Kyrkogatan} \vee \textit{P-Lot Left}) \cup \textit{Empty Spots} \quad \text{and}$$
$$\varphi_R = (\textit{Kyrkogatan} \vee \textit{P-Lot Right}) \cup \textit{Empty Spots}$$

allow the vehicle to go either left or right inside parking lot. With these, the final task specification is $\varphi = \varphi_L \vee \varphi_R$ and the corresponding TLT is shown in Fig. 1.

C. Results

The TLT is evaluated with a time horizon $T = 30$ seconds. In Fig. 3, we visualize the time evolution of the satisfaction set along the xy-plane. Since φ allows the vehicle to reach the empty spots via either left or right pass, we can see the tube filling the entirety of the parking lot's inside. The vehicle can enter the parking lot from both sides of Kyrkogatan and, as such, we can see the tube's ridge near the entrance splitting in both directions. Consider now that the left pass is blocked, see Fig. 4, which forces the vehicle to go via the right pass. The unplanned block results in the pruning of the

TLT branch (shown in Fig. 1) corresponding to φ_L. Then, the only remaining branch corresponds to φ_R, forcing the vehicle to switch to the right pass for the parking maneuver. Consequently, the backward reachable tube becomes much smaller and the system has stricter constraints on control inputs. Without needing to reconstruct the TLT, it is possible to react to the unplanned block by following the constraints imposed by computing least-restrictive control sets based on φ_R. Constructing the TLT in this simulation, with a 30 second time horizon, took 65.01 seconds on a system with an AMD Ryzen Threadripper 3970X and an NVIDIA GeForce RTX 2080 Ti while computing the least-restrictive control set took 18 milliseconds on average, resulting in a control rate of around 50 Hz.

VII. Conclusion

In this paper, we present an approach for guaranteeing a CPS completes an LTL task by synthesizing least-restrictive control sets from TLTs constructed from HJ reachability analysis. We are able to take advantage of the richness of LTL task specification together with the strong, low-level guarantees provided by HJ reachability analysis. We detail the key HJ reachability computations required to construct a TLT and provide an algorithm that verifies the existence of control policies that satisfy the constructed TLT. Then, if there exists control policies that satisfy the constructed TLT, we can efficiently compute least-restrictive control sets that guarantee the CPS completes the specified task. To implement and evaluate the approach, we develop pyspect for working with TLT. Using this toolbox, we showcase the efficacy of the approach on a simulated driving example where we visualize the evolution of the least-restrictive control sets and find the computation to be efficient. Our future work includes both the development of computationally-efficient approaches to directly resolving leaking corner issues and deploying this method on real hardware.

References

[1] K. P. Wabersich, A. J. Taylor, J. J. Choi, K. Sreenath, C. J. Tomlin, A. D. Ames, and M. N. Zeilinger, "Data-Driven Safety Filters: Hamilton-Jacobi Reachability, Control Barrier Functions, and Predictive Methods for Uncertain Systems," *IEEE Control Systems Magazine*, vol. 43, no. 5, pp. 137–177, 2023.

[2] I. M. Mitchell, *Application of level set methods to control and reachability problems in continuous and hybrid systems*. stanford university, 2002.

[3] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, "Hamilton-Jacobi Reachability: A Brief Overview and Recent Advances," no. arXiv:1709.07523, sep 2017.

[4] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2016.

[5] J. J. Choi, D. Lee, K. Sreenath, C. J. Tomlin, and S. L. Herbert, "Robust control barrier–value functions for safety-critical control," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 6814–6821.

[6] M. Althoff, "An introduction to CORA 2015," in *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.

[7] N. Kochdumper, F. Gruber, B. Schürmann, V. Gaßmann, M. Klischat, and M. Althoff, "AROC: A Toolbox for Automated Reachset Optimal Controller Synthesis," in *Proc. of the 24th International Conference on Hybrid Systems: Computation and Control*, 2021.

[8] B. Yordanov, J. Tumova, I. Cerna, J. Barnat, and C. Belta, "Temporal logic control of discrete-time piecewise affine systems," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1491–1504, 2011.

[9] S. Karaman, R. G. Sanfelice, and E. Frazzoli, "Optimal control of mixed logical dynamical systems with linear temporal logic specifications," in *2008 47th IEEE Conference on Decision and Control*. IEEE, 2008, pp. 2117–2122.

[10] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding Horizon Temporal Logic Planning," *IEEE Transactions on Automatic Control*, vol. 57, no. 11, pp. 2817–2830, 2012.

[11] A. Ulusoy and C. Belta, "Receding horizon temporal logic control in dynamic environments," *The International Journal of Robotics Research*, vol. 33, no. 12, pp. 1593–1607, 2014.

[12] C. Belta, B. Yordanov, and E. A. Gol, *Formal Methods for Discrete-Time Dynamical Systems*. Springer, 2017.

[13] Y. Gao, A. Abate, F. J. Jiang, M. Giacobbe, L. Xie, and K. H. Johansson, "Temporal Logic Trees for Model Checking and Control Synthesis of Uncertain Discrete-time Systems," *IEEE Transactions on Automatic Control*, p. 1, 2021.

[14] M. Chen, Q. Tam, S. C. Livingston, and M. Pavone, "Signal temporal logic meets reachability: Connections and applications," in *International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2018, pp. 581–601.

[15] L. Lindemann and D. V. Dimarogonas, "Control barrier functions for signal temporal logic tasks," *IEEE control systems letters*, vol. 3, no. 1, pp. 96–101, 2018.

[16] P. Yu, Y. Gao, F. J. Jiang, K. H. Johansson, and D. V. Dimarogonas, "Online Control Synthesis for Uncertain Systems under Signal Temporal Logic Specifications," *arXiv e-prints*, p. arXiv:2103.09091, mar 2023.

[17] F. J. Jiang, Y. Gao, L. Xie, and K. H. Johansson, "Ensuring safety for vehicle parking tasks using Hamilton-Jacobi reachability analysis," in *2020 59th IEEE Conference on Decision and Control (CDC)*, 2020, pp. 1416–1421.

[18] ——, "Human-Centered Design for Safe Teleoperation of Connected Vehicles," in *IFAC Conference on Cyber-Physical Human-Systems*, Shanghai, China, 2020.

[19] C. He, Z. Gong, M. Chen, and S. Herbert, "Efficient and Guaranteed Hamilton-Jacobi Reachability via Self-Contained Subsystem Decomposition and Admissible Control Sets," *IEEE Control Systems Letters*, 2023.

[20] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*. Association for Computing Machinery, 2013, pp. 854–860.

[21] P. Yu, Y. Gao, F. J. Jiang, K. H. Johansson, and D. V. Dimarogonas, "Online control synthesis for uncertain systems under signal temporal logic specifications," *The International Journal of Robotics Research*, 2023.

[22] M. Chen, S. L. Herbert, M. S. Vashishtha, S. Bansal, and C. J. Tomlin, "Decomposition of Reachable Sets and Tubes for a Class of Nonlinear Systems," *IEEE Transactions on Automatic Control*, vol. 63, no. 11, pp. 3675–3688, 2018.

[23] D. Lee, M. Chen, and C. J. Tomlin, "Removing Leaking Corners to Reduce Dimensionality in Hamilton-Jacobi Reachability," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 9320–9326.

[24] F. J. Jiang, K. M. Arfvidsson, C. He, M. Chen, and K. H. Johansson, "Guaranteed completion of complex tasks via temporal logic trees and hamilton-jacobi reachability (extended version)," *arXiv preprint arXiv:2404.08334*, 2024.

[25] M. Bui, G. Giovanis, M. Chen, and A. Shriraman, "OptimizedDP: An Efficient, User-friendly Library For Optimal Control and Dynamic Programming," no. arXiv:2204.05520, apr 2022.

[26] J. F. Fisac, M. Chen, C. J. Tomlin, and S. S. Sastry, "Reach-avoid problems with time-varying dynamics, targets and constraints," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. Seattle Washington: ACM, apr 2015, pp. 11–20.

[27] E. N. Barron and H. Ishii, "The Bellman equation for minimizing the maximum cost," *Nonlinear Analysis: Theory, Methods & Applications*, vol. 13.

[28] E. N. Barron, "Differential games maximum cost," *Nonlinear Analysis: Theory, Methods & Applications*, vol. 14, jun.