IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

Online Modifications for Event-based Signal Temporal Logic Specifications

David Gundana and Hadas Kress-Gazit

***Abstract*—In this paper we present a grammar and control synthesis framework for online modification of Event-based Signal Temporal Logic (STL) specifications, during execution. These modifications allow a user to change the robots' task in response to potential future violations, changes to the environment, or user-defined task changes. In cases where a modification is not possible, we provide feedback to the user and suggest alternative modifications. We demonstrate our task modification process using a Hello Robot Stretch.**

I. INTRODUCTION

HIGH-LEVEL specifications can be used to describe complex robotic tasks, and there exist algorithms to synthesize control from them [1]–[7]. These tasks may include time critical objectives, response to environment events or disturbances, and complex sequencing of actions. To describe these high-level specifications, researchers have used temporal logics such as Linear Temporal Logic (LTL) [8] for discrete abstractions of systems and Signal Temporal Logic (STL) [9] for behaviors defined over continuous signals. A crucial aspect of using such formalisms in robotics in the ability to automatically synthesize robot control from specifications [5].

Approaches to control synthesis for STL specifications include Model Predictive Control (MPC), Control Barrier Functions (CBFs), or other Mixed-integer Linear Programs (MILP); MPC methods satisfy specifications by converting them, and the dynamics of the system, into a receding-horizon optimization problem [7], [10]–[12]. These methods are useful in that they provide guarantees on the satisfaction of a specification over the prediction horizon, but they do not scale to long time horizons or complex specifications, due to the increase in decision variables. Other MILP solutions to synthesizing control satisfying STL specifications [13], [14] attempt to improve the computational efficiency by decomposing the specification into subtasks or waypoints. Work from [15] provide methods to satisfy STL specifications using Mixed-integer Convex Programming while reducing the number of decision variables in the optimization problem. While these methods maintain guarantees on the satisfaction of the specification, they rely on solving optimization problems whose decision variables increase with the complexity of a specification. This complexity makes specifications difficult to satisfy in real time and in the presence of uncontrolled environment events. Work from [16]–[19] introduced the use of time-varying CBFs to satisfy STL specifications. Time-varying CBFs improve the computational efficiency of synthesizing control to satisfy specifications, as there are less decision variables in the optimization problem. These techniques are computationally efficient enough to create control for complex specifications in real time; however, they cannot provide guarantees, as in the MPC approach.

In prior work [20], [21], we have introduced Event-based STL to describe tasks that include reactivity to external environment events. To do this, we defined a grammar and a control synthesis approach that is computationally efficient enough to enable the system to respond in real-time to events, while also satisfying timing constraints. To control the system, we compose, using an automata-based framework, time-varying CBFs [16] such that the system either completes the task, or provides pre-failure warnings.

In this paper, we provide the ability to update the specification, i.e. the task, *while the robots are executing it*. During execution, changes in the environment may cause previously satisfiable specifications to become unsatisfiable. For example, a robot operating in a space with humans might require more time to reach a goal point while trying to avoid collisions. In our previous work, we provide a pre-failure warning for this case, but the specification may be violated as the time constraints from the Event-based STL specification are specified before the execution. There may also exist new tasks that the operator would like the robots to accomplish during execution, in addition to the original task. For example, while a robot is patrolling two rooms, the user may want to add a third room for the robot to patrol. In these cases, instead of stopping, synthesizing a new specification, and restarting the execution, the user will add the new requirements while continuing to satisfy the original task and its timing constraints.

We present a grammar for modifying Event-based STL specifications (Sec. III) and a framework for online control synthesis (Sec. V) that seamlessly incorporates the modified specifications into the robot control. The modified specification is satisfied, if possible given the robots bounded control input. When a satisfying controller is not found, we provide automatically generated feedback to the user, enabling them to refine the modifications. In modifying specifications, we allow changing the parameters of the currently executing specification, and adding new tasks to the existing specification.

There has been previous work modifying tasks in real time. In [22], [23], tasks are updated based on the addition of new information from sensors and knowledge base (real-time plan configuration). In [23], specifications are divided into hard and soft sub-specifications and plans to satisfy the specification

Manuscript received: June 10, 2023; Revised October 1, 2023; Accepted November 10, 2023. This paper was recommended for publication by Editor Lucia Pallottino upon evaluation of the Associate Editor and Reviewers' comments. This work is supported by the National GEM Consortium, Cornell Sloan Fellowship, and NSF IIS-1830471.

D. Gundana and H. Kress-Gazit are with Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY, 14853 USA. e-mail: {dog4,hadaskg}@cornell.edu.

Digital Object Identifier (DOI): see top of this page.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

are generated before execution begins. Hard sub-specifications, such as safety constraints, must always be satisfied while least-violating solutions are found which make progress towards the satisfaction of soft sub-specifications. During execution, the plan is updated and new solutions are found based on environment knowledge updates from the operating robots. In this paper, we allow the user to directly update tasks based on their preference or changes in goals without the need to find a least-violating solution.

Using existing methods such as MPC [7], [10]–[12] or MILP solutions [13], [14], one can update the specification; however, the new specification would initiate re-synthesis and, as stated above, be computationally inefficient. In most cases, we provide an efficient solution to modify tasks online without the need to stop or pause execution, and we discuss computational limitations that require executions to pause for large specification modifications.

Assumptions: We assume that all robots in the system have knowledge of the state of all other robots in the system, static obstacles, and dynamic obstacles. However, the robots do not know or try to reason about the behavior of other robots and moving obstacles. Additionally, we assume that the specification is not trivially violated at initialization.

Contributions: In this paper we present three main contributions: 1) a grammar for modifying Event-based STL specifications that formally defines how specifications can be modified during execution, 2) an automated control synthesis framework for satisfying modified Event-based STL specifications at runtime, 3) automatically generated feedback when a modification may not have the desired affect or when a specification may be violated in the future due to the bounded control of the robots. We demonstrate our framework in hardware using a Hello Robot Stretch.

II. PRELIMINARIES

A. System and Environment Model

The state and bounded control input of the system at time t are denoted by $\mathbf{x}_t \in \mathbb{R}^n$ and $\mathbf{u}_t \in \mathbf{U} \subset \mathbb{R}^m$, respectively. Eqn. (1) describes the discrete time dynamical system we control.

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t) + g(\mathbf{x}_t)\mathbf{u}_t, \tag{1}$$

where the functions $f : \mathbb{R}^n \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ are locally Lipschitz continuous.

The environment in which the system is operating contains static and dynamic obstacles; static obstacles, such as walls and other objects whose positions do not change during execution, are known a priori. The position of dynamic obstacles such as humans, robots, and other objects, $\mathbf{x}_{dyn,t}$ are known to the system during execution. An Event-based STL specification may require that the system avoid these dynamic obstacles (collision-avoidance) or operate with them. For example, a specification may require a robot to navigate to and grasp a moving object. For convenience we denote the state of the system and dynamic obstacles as $\mathbf{X}_t = (\mathbf{x}_t, \mathbf{x}_{dyn,t})$.

B. LTL and Büchi Automata

Linear Temporal Logic [8] has been used for synthesis in robotic systems [5]. An LTL formula is constructed from Boolean propositions $\pi \in AP$ where AP is a set of atomic propositions. The grammar is as follows:

$$\gamma ::= \pi | \ \neg\gamma \mid \gamma_1 \vee \gamma_2 \mid X\gamma \mid \gamma_1 U \gamma_2. \tag{2}$$

Where $\neg$ and $\vee$ are the Boolean operators "not" and "or" respectively and X and U are the temporal operators "next" and "until" respectively. The semantics of LTL are defined over an infinite sequence $\sigma = \sigma_1, \sigma_2, \ldots$ of truth assignments to the Boolean propositions $\pi \in AP$, where σ_i represents the set of AP that are $True$ at position i. The full semantics of LTL can be found in [8].

A Büchi automata is a tuple $B = (S, s_0, \Sigma, \delta, F)$. S is the set of states, where $s_0 \in S$ is the initial state and $F \subseteq S$ is a set of accepting states, Σ is a finite input alphabet, and $\delta : S \times \Sigma \to 2^S$ is a transition function. A transition occurs between states when the label on the transition between the two states is evaluated to $True$. A run of a Büchi automata, B, on an infinite word $w = w_1, w_2, w_3, \ldots, w_i \in 2^\Sigma$ is an infinite sequence of states $s_1, s_2, s_3, \ldots$, s.t. $\forall j \geq 1, (s_{j-1}, w_j, s_j) \in \delta$. A run of B is accepting if and only if $\inf(\omega) \cap F \neq \emptyset$, where $\inf(\omega)$ represents a set of states that are visited infinitely often on the input word w. Given an LTL formula γ, we can construct a Büchi automaton B_γ that accepts infinite words if and only if they satisfy γ. In the following we use [24] for this construction.

C. Büchi Intersection

Given two Büchi automata, $B_{\gamma_1} = (S^1, s_0^1, \Sigma, \delta^1, F^1)$ and $B_{\gamma_2} = (S^2, s_0^2, \Sigma, \delta^2, F^2)$, their intersection $B_{\gamma_1} \times B_{\gamma_2}$ is a Büchi automaton whose accepting traces satisfy $\gamma_1 \wedge \gamma_2$. An accepting run of this Büchi intersection must visit the accepting states of both Büchi automata infinitely often. From [8]: $B_\gamma = B_{\gamma_1} \times B_{\gamma_2} = \{S^1 \times S^2 \times \{0, 1, 2\}, s_0^1 \times s_0^2 \times \{0\}, \Sigma, \Delta, S^1 \times S^2 \times \{2\}\}$. A transition in the resultant Büchi automata is defined as $\Delta = ((g, q, i), \alpha, (g', q', j))$. A transition exists in Δ if the transition $(g, \alpha, g') \in \delta^1$ and $(q, \alpha, q') \in \delta^2$, where $\alpha \in \Sigma$. The accepting states of B_γ ensure that accepting states in B_{γ_1} and B_{γ_2} are visited infinitely often. Further details can be found in [8].

In this work we follow the Büchi intersection definition from [25] which does not assume both Büchi automata share the same finite alphabet Σ. Given two Büchi automata, B_{γ_1} and B_{γ_2} with input alphabets Σ_1 and Σ_2, we define the input alphabet of the resultant intersection to be $\Sigma = \Sigma_1 \cup \Sigma_2$ and use the projection on the original alphabets when determining transitions.

D. Event-based STL

Event-based STL [21] is defined over Boolean predicates μ representing the system state, and uncontrolled environment events which are represented as Boolean propositions π. The truth value of μ is determined by the evaluation of a predicate function $h(\mathbf{X}_t)$ as follows:

$$\mu ::= \begin{cases} False & \Rightarrow h(\mathbf{X}_t) < 0 \\ True & \Rightarrow h(\mathbf{X}_t) \geq 0. \end{cases} \tag{3}$$

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

An Event-based STL specification Ψ is constructed using the following syntax:

$$\varphi ::= \mu \mid \neg\mu \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2, \tag{4}$$

$$\alpha ::= \pi \mid \neg\alpha \mid \alpha_1 \wedge \alpha_2, \tag{5}$$

$$\begin{aligned}\Psi ::= & G_{[a,b]}\ \varphi \mid F_{[a,b]}\ \varphi \mid \varphi_1\ U_{[a,b]}\ \varphi_2 \mid \\ & G(\alpha \Rightarrow\ \Psi) \mid \Psi_1 \wedge \Psi_2 \mid \Psi_1 \vee \Psi_2,\end{aligned} \tag{6}$$

where φ is a formula over predicates μ, α is a Boolean formula over $\pi \in AP$, $\{a, b\} \in \mathbb{R}^+$ are timing bounds for a formula, F is "Eventually", G is "Always", U is "Until", and $\Rightarrow$ is implication. The operator G with no timing bound is assumed to have the timing bounds of $[0, \infty]$. The full definition of the semantics of Event-based STL is in [21] and we show an example in Section VI. Note that, unlike STL, Event-based STL only allows limited nesting of temporal operators. Specifically, temporal operators can only be nested on the right-hand side of the implication formula $G(\alpha \Rightarrow\ \Psi)$

The approach in [21] synthesizes control for Event-based STL formulas Ψ by first abstracting Ψ into an LTL formula, creating the corresponding Büchi automata and using it to sequentially compose Control Barrier Functions (CBFs) corresponding to the predicates in Ψ.

E. Control Barrier Functions for Event-based STL

Control Barrier Functions [26]–[28] are used to create control for a system so that it is guaranteed to remain in a "safe-set" if the initial state of the system is in the set. Work in [16] created time-varying CBFs to generate control for robotic systems so that they satisfy a subset of STL specifications. In [20] we leveraged that work to synthesize control for Event-based STL specifications; each predicate μ and its associated timing bounds correspond to a time-varying CBF, which is used to generate the control whenever needed. The conditions for a time-varying CBF to be valid are defined in [16].

III. MODIFICATION GRAMMAR

The main contribution of this paper is to enable users to modify specification during execution. We specify the allowable modifications using the following grammar:

$$\begin{aligned}\Psi_{new} ::= & \Psi \mid \Psi_{new} \vee \Psi' \mid \Psi_{new} \wedge \Psi' \mid \\ & \Psi_{new,\mu\to\mu'} \mid \Psi_{new,[a,b]\to[a',b']}\end{aligned} \tag{7}$$

This grammar describes four different types of modifications that we allow:

1) $\Psi_{new} \vee \Psi'$ describes the addition of an Event-based STL specification using disjunction. Here, the system must satisfy either the original OR the new specification
2) $\Psi_{new} \wedge \Psi'$ describes the addition of an Event-based STL specification Ψ' using conjunction. Here, the original specification and new specification must both be satisfied
3) $\Psi_{new,\mu\to\mu'}$ modifies a predicate μ within a specification Ψ, i.e. replace μ with μ'
4) $\Psi_{new,[a,b]\to[a',b']}$ modifies timing bounds of an existing temporal operator in Ψ, i.e. replace $[a, b]$ with $[a', b']$. This modification changes the timing bounds of all predicates $\mu \in \varphi$ in a specific formula $F_{[a,b]}\varphi$, $G_{[a,b]}\varphi$, or $\varphi_1 U_{[a,b]}\varphi_2$.

IV. PROBLEM STATEMENT

Given an Event-based STL specification Ψ, user defined modifications Ψ_{new} given during execution, uncontrolled environment external events α, and the state of the system and environment $\mathbf{X}_t$, find a control strategy u to satisfy the modified Event-based STL specification during execution. If a control strategy is not found, provide feedback to the user. When a modification Ψ_{new} is made to a specification Ψ, the original specification is replaced for the remainder of the execution; we allow multiple modifications during runtime.

V. CONTROL SYNTHESIS FOR ONLINE MODIFICATIONS

A. Synthesis Overview

In this paper, we follow the control synthesis approach defined in [21] to automatically generate control and feedback for an Event-based STL specification. We use Algo. 1 to briefly outline this procedure and highlight the changes needed for online modification (lines highlighted in blue).

***prepSpec* (line 1 of Algo. 1):** First, we pre-process the specification by abstracting Ψ as an LTL formula γ and generating a Büchi automaton B with an initial state s_0. While abstracting the specification, we record the continuous semantics of each predicate μ and temporal operator in the Event-based STL specification Ψ through a set of abstracted propositions $\pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}} \in \Pi_\mu$. The subscripts of the abstracted propositions represent parameters of the formulas: the predicate (μ_i), the timing bounds ($[a, b]$), the environment events that the predicate must respond to (α_i), if any, and the dependence on other predicates in the specification (φ_{unt}). Further details on how to determine each subscript are in Algorithm 1 in [21]. To create the LTL formula γ, we replace the predicates with their abstracted proposition, and replace each temporal operator $F_{[a,b]}, G_{[a,b]}, U_{[a,b]}$ with a temporal operator without timing bounds F, G, U. From γ we create a Büchi automaton B using the tool Spot [24].

The propositions in Π_μ contain information on the timing bounds $[a, b]$ and predicate functions $h(\mathbf{x}_t) \in H$ of each predicate in the specification that they abstract. To satisfy an Event-based STL specifications, we create time-varying control barrier function templates CBF using the continuous values recorded in each proposition in Π_μ. Using work from [16], [20], these time-varying control barrier functions ensure an Event-based STL task is satisfied given the timing constraints for each predicate function $h(\mathbf{x}_t) \in H$.

***pickT* (line 10 of Algo. 1):** Following the pre-processing step and initialization, execution of the specification begins and, at each timestep, we choose an instantaneously-robust trace through B to an accepting state that satisfies γ given the set of propositions that are $True$ at time t and the state of the system $\mathbf{X}_t$. Instantaneous robustness is a measure that determines how robustly a specification is satisfied at the current time step, given the current state of the environment and system. We determine the instantaneous robustness by evaluating the first transition of all accepting traces of the system. Within the first transition, we assign a robustness score ρ to each proposition that appears in the label of a transition.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

We denote this set of propositions in a transition that must be $True$ for a transition to occur as $\pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}} \in \Pi_{\boldsymbol{\sigma},True}$. Each robustness score ρ measures how robustly the system can satisfy that proposition. The robustness score is determined based on the current truth value of the proposition given the state of the environment, using Eqn. (8).

$$\rho(\pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}}) = \begin{cases} h(\mathbf{X}_t) \text{ if } \pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}} = True \\ t_{rem} \quad \text{if } \pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}} = False \end{cases} \tag{8}$$

For propositions that are currently $True$ and must remain $True$ to satisfy the transition, we define robustness as the value of the predicate function of the proposition $h(\mathbf{X}_t)$. For propositions that are currently $False$ and must become $True$ for the transition to occur, we define the robustness of the proposition as t_{rem} or the estimated time remaining until the proposition becomes $True$. This is found using the current state of the system, the state of the environment, and the control bounds of the robot. Given the robustness score of each proposition, we create a robustness tuple $P(\Pi_{\boldsymbol{\sigma},True}) = (\rho_1, ..., \rho_k)$ for each transition of an accepting trace. We choose the transition (and trace) that contains the highest minimal robustness score of all possible transitions. Given this transition, we activate barrier functions associated with each proposition that is $True$ in the transition we label as $\Pi_{\mu_{act}}$. Further details on the procedure for satisfying an Event-based STL specification that maximizes instantaneous robustness are in [21].

***generateControl* (line 15 of Algo. 1):** At each time step we generate control to satisfy the propositions that must be $True$ in the chosen transition that maximizes robustness. In this paper we use the CBF template from [20], [21] for each proposition in $\Pi_{\mu_{act}}$. We generate control for each robot in the system using Eqn. (9):

$$\min_{\mathbf{u}_j \in \mathbf{U}_j} \parallel \mathbf{u}_j - \hat{\mathbf{u}}_j \parallel s.t. \\ \frac{\partial cbf_{\Psi_j}(\mathbf{X}_t)^T}{\partial \mathbf{x}}(f(\mathbf{x}_t) + g(\mathbf{x}_t)\mathbf{u}_j) + \frac{\partial cbf_{\Psi_j}(\mathbf{X}_t)}{\partial t} \geq \\ -\nu(cbf_{\Psi_j}(\mathbf{X}_t)), \tag{9}$$

where cbf_{Ψ_j} is an approximation of a the minimum CBF for the propositions that are activated in $\Pi_{\mu_{act}}$ that affect a robot j, $\mathbf{u}_j$ is a nominal controller, and $\nu : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a locally Lipschitz continuous class K function.

***preF* (line 16 of Algo. 1):** During execution we provide pre-failure warnings if needed [21]. Warnings are given when a specification may be violated for liveness tasks, safety tasks, or for potential violations that may occur if environment events happen. These warnings can give insight to modifications that may make the specification feasible. The warnings are determined by evaluating the state of the system, the robustness of the transition being executed, and the state of the uncontrolled environment events.

In the following sections we show how we build on this control synthesis framework to make online modifications to an executing Event-based STL specification. The main contributions of this paper, highlighted in blue in Algo. 1, are contained in the use of the newly introduced modification grammar to make modifications at runtime (lines 5-7 in Algo. 1) and how transitions in γ are chosen to satisfy the modified Event-based STL specification (line 14 in Algo. 1).

Completeness: Our approach to satisfying Event-based STL specifications is sound but not complete. Since we choose the next transition based on instantaneous robustness, we may choose a trace that is currently valid but that becomes infeasible due to environment behavior later.

Furthermore, we do not restrict the user-provided modifications, therefore they may be infeasible. In these cases, we automatically provide feedback to the user that the specification is unsatisfiable and provide suggestions on new modifications that may make the specification feasible.

Algorithm 1: Control Synthesis for Event-based STL

Input : Ψ, σ_t, $\mathbf{X}_t$, Ψ_{new}
Output: u, $preF$

1 $[\Pi_\mu, B, H, CBF, s_0] = prepSpec(\Psi)$;
2 $\mathcal{B} = [B]$;
3 $s_{\mathcal{B}} = [s_0]$;
4 **while** *True* **do**
5 **if** $\Psi \neq \Psi_{new}$ **then**
6 $[\mathcal{B}, s_{\mathcal{B}}, \Pi_\mu, CBF, H, \Psi] =$ $modify(\mathcal{B}, s_{\mathcal{B}}, \Pi_\mu, CBF, H, \Psi, \Psi_{new})$;
7 **end**
8 $P = [\ ]$;
9 **for** $i = 1$ *to* $|\mathcal{B}|$ **do**
10 $[s, \Pi_{\mu_{act}}, P(\Pi_{\boldsymbol{\sigma},True})] =$ $pickT(\sigma_t, \mathbf{X}_t, \mathcal{B}[i], H, s_{\mathcal{B}}[i])$;
11 $s_{\mathcal{B}}[i] = s$;
12 $P[i] = (\Pi_{\mu_{act}}, P(\Pi_{\boldsymbol{\sigma},True}))$;
13 **end**
14 $[\Pi_{\mu_{act}}, \mathcal{B}, s_{\mathcal{B}}] = chooseProps(P, \mathcal{B}, s_{\mathcal{B}})$;
15 $u = generateControl(\Pi_{\mu_{act}}, t, \mathbf{X}_t, CBF)$;
16 $preF = preFailure(\sigma_t, \mathbf{X}_t, \mathcal{B}, H, s_{\mathcal{B}})$;
17 **end**

Example: We use the following example to illustrate automatically generating control for modified specifications:

$$\Psi_{alarm} = G(alarm \Rightarrow F_{[0,7]}(\parallel \mathbf{x}_t - [3,4] \parallel < 1)) \tag{10}$$

For a robot with state $\mathbf{x}_t = [x_t, y_t]$, this specification states that if the event $alarm$ is sensed, then the distance from the robot and point $[3, 4]$ must eventually, within 7 seconds, be less than 1.

B. Online Modifications of Event-based STL Specifications

During execution an operator may make modifications to a specification online, according to the grammar defined in Sec. III. If the modified specification $\Psi_{new} \neq \Psi$ (lines 5-7 in Algo. 1), the task execution is modified following Algo. 2:

***findMods* (line 1)**: Given the original specification Ψ and the modified specification Ψ_{new}, we first parse the modifications. The list $\boldsymbol{\Psi}' = [\Psi'_1, \Psi'_2, \ldots, \Psi'_i]$ contains the new formulas Ψ'_i that are added to the original specification Ψ; the list $\mathbf{op} = [op_1, op_2, \ldots, op_i]$ captures which Boolean operator is used, $op_i \in \{\vee, \wedge\}$. Modifications to the timing bounds and predicate of a specification are captured by $\boldsymbol{\pi}_{\mu,[a,b]\to\mu',[a',b']} =$

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

Algorithm 2: $modify$ (Update Control)

Input : $\mathcal{B}$, $s_{\mathcal{B}}$, Π_μ, CBF, H, Ψ, Ψ_{new}
Output: $\mathcal{B}$, $s_{\mathcal{B}}$, Π_μ, CBF, H, Ψ

1 $[\boldsymbol{\Psi}', \mathbf{op}, \boldsymbol{\pi}_{\mu,[a,b]\rightarrow\mu',[a',b']}] = findMods(\Psi, \Psi_{new})$;
2 **for** $i = 1$ *to* $|\boldsymbol{\Psi}'|$ **do**
3 $\quad [\Pi_\mu^{'}, B^{'}, H^{'}, CBF^{'}, s_0^{'}] = prepSpec(\boldsymbol{\Psi}'[i])$;
4 $\quad \Pi_\mu \leftarrow \Pi_\mu \cup \{\Pi_\mu^{'}\}$;
5 $\quad H \leftarrow H \cup \{H^{'}\}$;
6 $\quad CBF \leftarrow CBF \cup \{CBF^{'}\}$;
7 $\quad$ **if** $\boldsymbol{op}[i] = \wedge$ **then**
8 $\quad\quad [\mathcal{B}, s_{\mathcal{B}}] = intersectB(\mathcal{B}, s_{\mathcal{B}}, B^{'}, s_0^{'})$;
9 $\quad$ **else if** $op[i] = \vee$ **then**
10 $\quad\quad \mathcal{B} = [\mathcal{B}, B^{'}]$;
11 $\quad\quad s_{\mathcal{B}} = [s_{\mathcal{B}}, s_0^{'}]$;
12 $\quad$ **end**
13 **end**
14 **for** $i = 1$ *to* $|\boldsymbol{\pi}_{\mu,[a,b]\rightarrow\mu',[a',b']}|$ **do**
15 $\quad [\Pi_\mu, CBF, H, \Psi] =$
$\quad\quad modifyProp(\Pi_\mu, CBF, H, \Psi, \boldsymbol{\pi}_{\mu,[a,b]\rightarrow\mu',[a',b']}[i])$
16 **end**
17 $\Psi = \Psi_{new}$;

$[\pi_{\mu,[a,b]\rightarrow\mu',[a',b']_1}, \pi_{\mu,[a,b]\rightarrow\mu',[a',b']_2}, \ldots, \pi_{\mu,[a,b]\rightarrow\mu',[a',b']_i}]$, where each element in the list represents a modification of the predicate or timing bounds of an existing proposition π_μ to a proposition with a new predicate or timing bound $\pi_{\mu'}$. In our example, during execution, we modify Ψ_{alarm} to be:

$$\begin{aligned}\Psi_{new} = (&G(alarm \Rightarrow F_{[0,15]}(\| \mathbf{x}_t - [3,5] \| < 0.5)) \\ &\wedge G_{[0,50]}(\| \mathbf{x}_t - [1,1] \| > 0.5)) \\ &\vee G(alarm \Rightarrow (F_{[0,15]} \| \mathbf{x}_t - [0,4] \| < 1))\end{aligned} \tag{11}$$

For Ψ_{new} in our example:

- $\Psi_1' = G_{[0,50]}(\| \mathbf{x}_t - [1,1] > 0.5 \|)$
- $\Psi_2' = G(alarm \Rightarrow (F_{[0,15]} \| \mathbf{x}_t - [0,4] \| < 1))$
- $\mathbf{op} = [\wedge, \vee]$
- $\pi_{\|\mathbf{x}_t - [3,4]\| < 1,[0,7]} \rightarrow \pi_{\|\mathbf{x}_t - [3,5]\| < 0.5,[0,15]}$

When abstracting Ψ_{new}, we maintain the order of the predicates as they appear in the specification to match the index of the existing abstracted predicates in Ψ. This is so that we can identify the predicates that exist in the original specification that are modified, and those that are added via conjunction or disjunction.

prepSpec **(lines 3-6):** We prepare the specification $\boldsymbol{\Psi}'[i]$ in the same manner as the original specification Ψ. This includes abstracting $\boldsymbol{\Psi}'[i]$ as an LTL formula and generating a Büchi automaton (line 3 of Algo. 2). From abstracting $\boldsymbol{\Psi}'[i]$, the abstracted propositions Π_μ', the set of predicate functions H', and the set of Control Barrier Functions CBF' are added to their respective sets. Depending on $\mathbf{op}[i]$, we update the current set of Büchi automata $\mathcal{B}$.

intersectB **(line 8):** If the specification is added via conjunction, $\mathbf{op}[i] = \wedge$, we find the Büchi intersection of all $B \in \mathcal{B}$ and the Büchi automaton for the added specification $B^{'}$ following the procedure in Sec. II-C and [8]. When making this modification, the size of the set $\mathcal{B}$ does not change. The intersection of B and B' results in a single Büchi automaton in which an accepting run satisfies both Ψ and $\Psi'[i]$.

(lines 10 - 11): If the specification is added via disjunction, $\mathbf{op}[i] = \vee$, then the Büchi automaton B' and its initial state s_0' are added to the current lists $\mathcal{B}$ and $s_{\mathcal{B}}$. During execution the system will determine which $B \in \mathcal{B}$ should be followed to maximize instantaneous robustness.

We follow the order of operation (parentheses first, followed by conjunction, and then disjunction) in which new specifications $\boldsymbol{\Psi}'[i]$ are added to the original specification Ψ as written in Ψ_{new}. For example, in our motivating example, a specification is added via conjunction followed by disjunction. This results in the list of Büchi automata being $\mathcal{B} = [B \times B_1, B_2]$, where B is the Büchi automaton for the original specification Ψ, B_1 is the Büchi automaton for the specification added via conjunction Ψ_1', and B_2 is the Büchi automaton for the specification added via disjunction Ψ_2'.

modifyProp **(line 15):** For all modifications of the predicate or timing bounds that are made to Ψ, we first determine if the proposition that is being modified exists in a transition to the forward reachable states of all $B \in \mathcal{B}$. That is, if the proposition will ever be required to be $True$ in any transition that is reachable from the current states of all $B \in \mathcal{B}$. If the proposition is in the forward reachable states, this means that the proposition may be activated in the future and we modify the predicate or timing bounds of the abstracted proposition. If the proposition does not exist in a transition to the forward reachable states, this means that the system has already satisfied the proposition and it will not be satisfied again. In this case we notify the user that the modification is no longer relevant for the task.

For example, consider a specification where a robot is tasked with eventually being in a room between times 5 and 15. After the robot has successfully visited the room within the time bounds, the system will not require the proposition that abstracts the task to be $True$ again during the execution. The user may make a modification to the specification which changes the location of the room that the robot is to eventually visit. If the user modifies the specification after the robot has already visited the original room location, the robot will not visit the new room location. In this case, the system recommend that the user add the new task via conjunction.

After all modifications are made, the Event-based STL specification Ψ is replaced with the new specification Ψ_{new}. This is so that, in subsequent timesteps, additional new modifications may be added.

C. Determining a Specification to Satisfy (Algo. 1)

Before execution, we initialize the lists $\mathcal{B}$ and $s_{\mathcal{B}}$ (lines 2-3) to store the Büchi automaton and its current state for the original specification Ψ; we add the Büchi automata and initial states for any specifications Ψ' added via disjunction through online modifications (line 6) .

chooseProps **(line 14):** When a specification is modified using disjunction, the system must choose which $B \in \mathcal{B}$ it should follow to satisfy the specification. We determine which Büchi automaton and specification the system should follow by evaluating the instantaneous robustness of all Büchi

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

automata. By doing this, we choose to follow and satisfy the specification which maximizes robustness for the system. Using the procedure from [21], at each time step we calculate the instantaneous robustness of the shortest path to an accepting state in each $B \in \mathcal{B}$. We present two methods to determine which $B \in \mathcal{B}$ should be executed:

1) We choose the $B \in \mathcal{B}$ that has the maximum instantaneous robustness and remove all other B from $\mathcal{B}$. This results in $|\mathcal{B}| = 1$ and essentially means that the system commits to one of the formulas in the disjunction until another modification is made.
2) For the current timestep we choose to execute the $B \in \mathcal{B}$ that maximizes robustness but we do not remove any other B. Instead, at each timestep we re-evaluate each $B \in \mathcal{B}$ and always choose to execute the B with maximum instantaneous robustness. If a $B \in \mathcal{B}$ is violated at a timestep, it is removed from the set because it can no longer be satisfied.

The trade-off between these methods is that in the first method, we save computation time by immediately committing to one Büchi automaton, essentially satisfying one clause of the disjunction, at the risk of failing to fulfil the full specification (if, for example, the environment changes and that part of the specification can no longer be completed in time). The second method, on the other hand, may be more robust to environment changes, as it recalculates the instantaneous robustness at every time step; however, this requires more computation time and might include chattering, where the system switches between different behaviors. For both methods, if the environment is adversarial, we cannot guarantee task completion. In this paper we follow the second method. We leave further analysis regarding the best choice of the Büchi automaton to future work.

VI. PHYSICAL DEMONSTRATION

We demonstrate modifying a specification at runtime through scenarios where a mobile manipulator picks up objects from known locations and moves them to known depots.

A. Robot and Environment Model

We use a Hello Robot Stretch (Fig. 1) with state:

$$\mathbf{x}_t = [x_t, y_t, \theta_t, d_t, z_t, \beta_t]. \tag{12}$$

Where x_t, y_t, and θ_t are the position and orientation of the robot base in 2D, d_t is the distance from the robot origin to the tip of the gripper, z_t is the height of the tip of the gripper from the ground, and β_t represents the distance between the fingers of the gripper.

We model the base of the robot as a holonomic robot and use feedback linearization to generate wheel velocity commands for the differential drive base. This makes the model of the system linear w.r.t. the control. The control $\mathbf{u}_t$ is:

$$\mathbf{u}_t = [v_{x,t}, v_{y,t}, \omega_t, v_{d,t}, v_{z,t}, v_{\beta,t}], \tag{13}$$

where $v_{x,t}$ and $v_{y,t}$ are the velocities of the base in the x and y direction, ω_t is the rotational velocity of the base, $v_{d,t}$ is the velocity of the arm extension, $v_{z,t}$ is the velocity of the height of the arm, and $v_{\beta,t}$ is the velocity of the gripper.

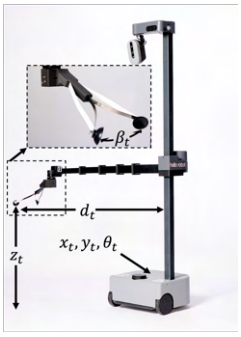

Fig. 1. Hello Robot Stretch [29] and its state $\mathbf{x}_t = [x_t, y_t, \theta_t, d_t, z_t, \beta_t]$.

We assume that the position of an object of interest $\mathbf{x}_{obj,t} = [x_{obj,t}, y_{obj,t}, z_{obj,t}]$ and the depot where the object is deposited $\mathbf{x}_{dep} = [x_{dep,t}, y_{dep,t}, z_{dep,t}]$ are known at all times; in practice, we use an Optitrack motion capture system to track the robot, object, and depot.

B. Original Task

The pick-and-place task is as follows: *Consider a Stretch robot operating in a lab environment where it must pick up items and deliver them to depot areas. The Event-based STL specification is* $\Psi_{collect} = \Psi_{pick} \wedge \Psi_{align} \wedge \Psi_{extend} \wedge \Psi_{retract} \wedge \Psi_{deposit}$ where:

- $\Psi_{pick} = G(pick \Rightarrow (F_{[0,30]}(d_{obj} < 1)))$
  - Given an external event $pick$, the robot has 30 seconds to get to the location of the object.
- $\Psi_{align} = G(d_{obj} < 1 \Rightarrow (F_{[0,15]}(\theta_{obj} < 0.1)))$
  - Whenever the robot is within 1 unit of the object, it must align its body and arm with the object within 15 seconds.
- $\Psi_{extend} = G((d_{obj} < 1 \wedge \theta_{obj} < 0.1) \Rightarrow (F_{[0,10]}(|z_t - z_{obj,t}| < 0.05 \wedge |d_t - d_{obj,t}| < 0.05)) \wedge F_{[10,15]}(\beta_t < 1)))$
  - If the robot is within 1 unit and aligned with the object, it must first extend its arm to reach the location of the object within 10 seconds. After 10 seconds but before 15 seconds, the distance between the gripper's fingers must be less than 1 (the gripper is closed).
- $\Psi_{retract} = G((\beta_t < 1) \Rightarrow ((\beta_t < 1) U_{[0,25]}(d_{dep} < 1 \wedge d_t < 0.2 \wedge |z_t - z_{dep}| < 0.1)))$
  - If the distance between the gripper fingers is less than 1 (the robot has grasped the object), the robot must maintain this gripper distance until it eventually within 25 seconds reaches the location of the depot. Additionally, the robot arm must eventually be retracted to be close to the body of the robot and the height of the gripper must eventually be within 0.1 units of the height of the depot.
- $\Psi_{deposit} = G((d_{dep} < 1) \Rightarrow (F_{[0,20]}(\theta_{dep} < 0.1 \wedge |d_t - d_{dep}| < 0.05) \wedge F_{[20,25]}(\beta_t > 3)))$
  - Whenever the robot is within 1 unit of the depot, the robot must align with the depot and extend its arm so that the gripper reaches the depot within 20 seconds. After 20 seconds but before 25 seconds of being within 1 unit of the depot, the distance between the grippers must be greater than 3, which releases the object.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

In the specification $d_{obj} = \| [x_t, y_t] - [x_{obj,t}, y_{obj,t}] \|$, $d_{dep} = \| [x_t, y_t] - [x_{dep,t}, y_{dep,t}] \|$, $\theta_{obj} = |\theta_t - tan^{-1}(\frac{y_{obj,t}-y_t}{x_{obj,t}-x_t})+\frac{\pi}{2}|$, and $\theta_{dep} = |\theta_t - tan^{-1}(\frac{y_{dep,t}-y_t}{x_{dep,t}-x_t})+\frac{\pi}{2}|$. The angles θ_{obj} and θ_{dep} align the robot perpendicular to the object/depot such that the arm, which extends outward and does not rotate, is aligned with the object/depot.

C. Online Modifications

During execution, the user can make online modifications to the specification in response to pre-failure warnings or in order to change the robot's behavior. We demonstrate such modifications below and in our accompanying video. Figure 2 shows the setup of the demonstration and one potential initial configuration of the robot, object, and depot.

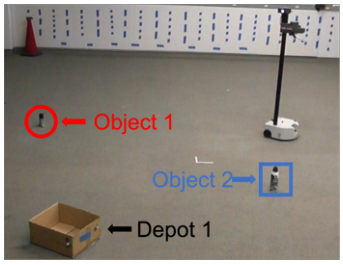

Fig. 2. Physical demonstration setup. Object 1 is shown in the red circle, Object 2 (used for modification 2) is in the blue square, the depot is in the bottom left corner, and the robot is in the top right corner.

1) During execution there may not be enough time for the robot to reach the object. This may be due to disturbances while navigating or an initial robot position that is too far away from the object. In the demonstration, the system generates a pre-failure warning indicating that there is not enough time to reach the location of the object. Here, the user is able to modify the specification to add more time for the robot to reach the object and allowing the remainder of the specification to be satisfied. We modify Ψ_{pick} to be the following:

- $\Psi_{pick} = G(pick \Rightarrow (F_{[0,45]}(d_{obj} < 1)))$

During execution, this gives the robot 15 more seconds to reach the object.

2) To demonstrate the ability to change predicate functions, we modify d_{obj} such that a different object that is placed in the environment is picked up by the robot. During execution we change each occurrence of d_{obj} in $\Psi_{collect}$ to be $d_{obj,2}$, where $d_{obj,2} = \| [x_t, y_t] - [x_{obj,2,t}, y_{obj,2,t}] \|$. Additionally, we modify the predicate $d_t < 0.2$ in $\Psi_{retract}$ to be $d_t < 0.05$. This results in the arm being closer to the robot's body while navigating. This aids in ensuring the object does not fall from the gripper as it navigates to the depot.

3) In the third demonstration, we add additional specifications via conjunction. The modified specification is $\Psi_{new} = \Psi_{collect} \wedge \Psi_{avoid}$, where $\Psi_{avoid} = G_{[0,100]}(\| [x_t, y_t] - [cone_{x,1}, cone_{y,1}] \| > 0.3 \wedge \| [x_t, y_t] - [cone_{x,2}, cone_{y,2}] \| > 0.3)$. This newly modified specification requires the robot to avoid two cones that are placed in the environment during execution. The location of the cones are known to the robot through the Optitrack system.

4) The final demonstration modifies the specification to allow the object to be deposited in either depot 1 or depot 2. To do this, we modify the specification to be $\Psi_{new} = \Psi_{collect} \vee \Psi_{collect'}$. Where $\Psi_{collect'} = \Psi_{pick} \wedge \Psi_{align} \wedge \Psi_{extend} \wedge \Psi_{retract'} \wedge \Psi_{deposit'}$. All instances of the position d_{dep} in $\Psi_{retract} \wedge \Psi_{deposit}$ are replaced with $d_{dep,2}$, which represents the location of a second depot $d_{dep,2} = \| [x_t, y_t] - [x_{dep,t,2}, y_{dep,t,2}] \|$. By adding $\Psi_{collect'}$ via disjunction, the system chooses to navigate and deposit the item in the depot such that robustness is maximized at each timestep. In the demonstration the locations of the depots are changed in real time and the system chooses which depot the object should be placed based on the time remaining in the specification and the location of the depots.

D. Demonstration Discussion

In the accompanying video we show a demonstration of each modification. Here, we discuss the computation limitations and results of the modifications. In the first demonstration the robot is able to complete the task as it is written. Following work from [21] and line 1 of Algo. 1 the specification is pre-processed. The time it takes to prepare a specification, the prep time, is the time required to abstract each predicate, create a Büchi automaton from the abstracted LTL formula using Spot [24], and to parse the output from Spot into a readable format. Prep time is a one-time preprocessing time for a given specification. The total prep time for our demonstration was 3.74 seconds. During execution of the first demonstration there are no pre-failure warnings given and the average computation time to generate a control for the robot at each time step was 0.14 seconds. All demonstrations are run on a 2.3 GHz Quad-Core CPU with 8 GB of RAM.

In modification 1, the initial position of the robot was changed so that the robot did not have enough time to reach the object in the required time. During execution a pre-failure warning was given that the specification may be violated in the future and Ψ_{pick} was modified to increase the time from 30 to 45 seconds. The average computation time for the execution of the specification was 0.14 seconds and the total modification time took 0.005 seconds to complete. This includes the time it takes to identify the modified timing bound and change all appropriate abstracted predicates to represent the modification.

The total modification time to find the predicate change in the specification and modify the abstracted predicates and CBFs for modification 2 was 0.009 seconds. In these cases both the modifications for timing bound and predicate change modifications do not require the execution to be stopped or paused because no pre-failure warning was given after the modification was made. The modification time only occurs for one time step and computation times for future time steps are not impacted by the modification.

For the third modification a new specification was added via conjunction. The total modification time for this modification includes the time to identify the new specification, abstract it as an LTL specification, create a Büchi automaton, and find a Büchi intersection with all existing Büchi automata in the system. The most computationally expensive operations are the Büchi automata generation with Spot [24] and finding a Büchi intersection with all Büchi in $\mathcal{B}$. For this demonstration, there was only one existing Büchi automaton. Büchi automata

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

are added to the system when specifications are added via disjunction. The complexity of the specification being added via conjunction greatly increases the modification time and, if the modification time is large, it might pause the execution. In our demonstration the total modification time of adding the collision avoidance specifications took 0.23 seconds. In this case, this is fast enough to not require the execution to pause.

The final modification adds a new specification $\Psi_{collect'}$ via disjunction. This specification has the same complexity as the original specification $\Psi_{collect}$. Because only predicates are changed between the two specifications, the abstracted LTL formulas are identical, and they result in the same Büchi automaton. The total modification time to add a specification via conjunction includes the time to identify the new specification being added, abstract it as an LTL formula, and generate a Büchi automaton. In this example, the modification time of $\sim$3.75 seconds was similar to the prep time for the original specification $\Psi_{collect}$ because of the identical Büchi automata. In some cases, this longer computation time may require an execution pause in order to satisfy the specification. However, because it is a one time computation, this may be done a priori if possible modifications are known. When adding specifications via disjunction, the system is given the choice of which specification to satisfy and it chooses the one that maximizes instantaneous robustness. As a result of searching more Büchi automata, the average computation time rises from 0.14 seconds to 0.16 seconds.

VII. Conclusions

In this paper we formally define a modification grammar for Event-based STL specifications. We present an automated control synthesis framework to satisfy, online, modifications of Event-based STL specifications, and provide feedback for modifications that may no longer be relevant based on the timing. Through our demonstrations we show that online modifications of predicates and timing bounds are computationally efficient and do not require a pause in execution.

Currently, the computation time of adding specifications via conjunctions or disjunction are dominated by the time it takes to generate a Büchi automaton, which increases with the complexity of the specification. In future work we will determine the feasibility of reducing this computation time by pre-computing complex Büchi automata for known modifications or exploring other potential methods to reduce the complexity of transforming an LTL formula to an automaton. Additionally, we will explore methods to efficiently compute the robustness of a full trace to an accepting state in the Büchi automaton rather than a single transition, thus creating a robust solution for the entire behavior of the robot.

References

[1] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multi-agent motion tasks based on LTL specifications," *Proceedings of the IEEE Conference on Decision and Control*, vol. 1, pp. 153–158, 2004.

[2] I. Filippidis, D. V. Dimarogonas, and K. J. Kyriakopoulos, "Decentralized multi-agent control from local LTL specifications," *Proceedings of the IEEE Conference on Decision and Control*, pp. 6235–6240, 2012.

[3] M. Kloetzer and C. Belta, "Temporal logic planning and control of robotic swarms by hierarchical abstractions," *IEEE Transactions on Robotics*, vol. 23, no. 2, pp. 320–330, 2007.

[4] J. Chen, S. Moarref, and H. Kress-Gazit, "Verifiable control of robotic swarm from high-level specifications robotics track," *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, vol. 1, no. 4, pp. 568–576, 2018.

[5] H. Kress-Gazit, M. Lahijanian, and V. Raman, "Synthesis for Robots: Guarantees and Feedback for Robot Behavior," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 211–236, 2018.

[6] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE transactions on robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[7] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *53rd IEEE Conference on Decision and Control*, pp. 81–87, IEEE, 2014.

[8] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model checking*. MIT press, 2018.

[9] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," *Lecture Notes in Computer Science*, pp. 152–166, 2004.

[10] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," in *Proceedings of the 18th international conference on hybrid systems: Computation and control*, pp. 239–248, 2015.

[11] S. S. Farahani, V. Raman, and R. M. Murray, "Robust model predictive control for signal temporal logic synthesis," *IFAC-PapersOnLine*, vol. 48, no. 27, pp. 323–328, 2015.

[12] M. Charitidou and D. V. Dimarogonas, "Barrier function-based model predictive control under signal temporal logic specifications," in *2021 European Control Conference (ECC)*, pp. 734–739, IEEE, 2021.

[13] A. T. Buyukkocak, D. Aksaray, and Y. Yazıcıoğlu, "Planning of heterogeneous multi-agent systems under signal temporal logic specifications with integral predicates," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1375–1382, 2021.

[14] D. Sun, J. Chen, S. Mitra, and C. Fan, "Multi-agent motion planning from signal temporal logic specifications," *arXiv preprint arXiv:2201.05247*, 2022.

[15] V. Kurtz and H. Lin, "Mixed-integer programming for signal temporal logic with fewer binary variables," *IEEE Control Systems Letters*, 2022.

[16] L. Lindemann and D. V. Dimarogonas, "Control barrier functions for signal temporal logic tasks," *IEEE control systems letters*, 2018.

[17] L. Lindemann and D. V. Dimarogonas, "Robust control for signal temporal logic specifications using discrete average space robustness," *Automatica*, vol. 101, pp. 377–387, 2019.

[18] L. Lindemann and D. V. Dimarogonas, "Decentralized control barrier functions for coupled multi-agent systems under signal temporal logic tasks," in *2019 18th European Control Conference*, pp. 89–94, 2019.

[19] A. T. Buyukkocak, D. Aksaray, and Y. Yazıcıoğlu, "Control barrier functions with actuation constraints under signal temporal logic specifications," *arXiv preprint arXiv:2204.03631*, 2022.

[20] D. Gundana and H. Kress-Gazit, "Event-based signal temporal logic synthesis for single and multi-robot tasks," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3687–3694, 2021.

[21] D. Gundana and H. Kress-Gazit, "Event-based signal temporal logic tasks: Execution and feedback in complex environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10001–10008, 2022.

[22] Y. Ding and Y. Zhang, "A logic approach for ltl system modification," in *International Symposium on Methodologies for Intelligent Systems*, pp. 435–444, Springer, 2005.

[23] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local ltl specifications," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, 2015.

[24] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, É. Renault, and L. Xu, "Spot 2.0 — a framework for LTL and ω-automata manipulation," *Lecture Notes in Computer Science*, pp. 122–129, 2016.

[25] A. Fang and H. Kress-Gazit, "Automated task updates of temporal logic specifications for heterogeneous robots," in *2022 International Conference on Robotics and Automation*, pp. 4363–4369, IEEE, 2022.

[26] P. Wieland and F. Allgöwer, "Constructive safety using control barrier functions," *IFAC Proceedings Volumes*, vol. 7, pp. 462–467, 2007.

[27] L. Wang, A. D. Ames, and M. Egerstedt, "Safety barrier certificates for collisions-free multirobot systems," *IEEE Transactions on Robotics*, vol. 33, no. 3, pp. 661–674, 2017.

[28] 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*, pp. 3861–3876, 2017.

[29] Hello Robot, "Stretch RE1," 2022. https://hello-robot.com/product.

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.