Event-Based Signal Temporal Logic Tasks: Execution and Feedback in Complex Environments

David Gundana and Hadas Kress-Gazit

***Abstract*—In this work, we synthesize control for high-level, reactive robot tasks that include timing constraints and choices over goals and constraints. We enrich Event-based Signal Temporal Logic by adding disjunctions, and propose a framework for synthesizing controllers that satisfy such specifications. If there are multiple ways to satisfy a specification, we choose, at runtime, a controller that instantaneously maximizes robustness. During execution, we automatically generate feedback in the form of pre-failure warnings that give users insight as to why a specification may be violated in the future. We demonstrate our work through physical and simulated multi-robot systems operating in complex environments.**

***Index Terms*—Formal Methods in Robotics and Automation, Hybrid Logical/Dynamical Planning and Verification, Multi-Robot Systems**

I. Introduction

GETTING robots to autonomously achieve complex tasks, such as guiding people in an environment, reacting to emergencies, or patrolling large areas, requires both the ability to *specify* tasks and *automatically synthesize* control. In this paper, we focus on satisfying high-level specifications for single and multi-robot systems operating around static and dynamic obstacles. High-level specifications, that can be captured in formalisms such as as Linear Temporal Logic (LTL) [1] and Signal Temporal Logic (STL) [2], have been used to describe complex robotics tasks; there are several approaches for creating controllers that satisfy them [3]–[9].

STL naturally describes tasks that include timing constraints [10]; several researchers [11]–[13] propose methods for synthesizing control from STL specifications. In addition to the usual Boolean semantics of temporal logic formulas where a trace is either satisfying or violating a specification, STL has quantitative ("robust") semantics for describing the extent to which a specification is satisfied; this has been used for generating robust control [14].

Event-based STL [15] expands the expressivity of STL by adding the ability to specify reactions to uncontrolled, discrete events. This reactivity to events such as alarms or user inputs allows one to describe a larger set of tasks than was previously possible. While existing solutions to satisfying reactive STL specifications exist, these events typically have assumptions on their bounds and timing [16]. Our prior work [15] provides a framework for automatically satisfying Event-based STL specifications and gives feedback on the feasibility of the tasks; in this paper we enrich the expressivity of the logic and provide more detailed feedback.

The syntax and semantics of Event-based STL in [15] does not allow disjunction (the "or" Boolean operator) in the description of the system's required behavior (it was allowed in the description of the uncontrolled events). This restricts the expressivity of the formalism by limiting the types of tasks that can be described; specifically, one could not capture choice in behavior – e.g. "if you hear an alarm go to exit 1 **or** exit 2 within 10 seconds" – due to the underlying control synthesis approach [13]. Here, we expand the expressive capabilities of Event-based STL through the addition of disjunction in the robot requirements. During execution, our control synthesis algorithm finds an instantaneously robust solution that satisfies such a specification, if possible.

Providing feedback to the user on the feasibility of a high-level specification is one of the strengths of synthesis from temporal logic specifications; it can be used to repair specifications that are unsatisfiable (e.g. [17]). Work in [18] describes several methods for providing feedback for infeasible LTL specifications. Other work [19]–[21] propose methods for repairing infeasible specifications by strengthening assumptions about the environment or relaxing assumptions about the system. All these approaches provide feedback offline - before execution. In this work, we provide feedback to the user, *during execution*, regarding not only specification violation but also future possible violation, i.e. *pre-failure warnings*, essentially alerting the user to possible problems down the line. From these pre-failure warnings, we provide suggestions that would reduce the possibility of a specification becoming unsatisfiable in the future.

In this paper we consider complex workspaces – areas the robots operate in that include static and dynamic obstacles. Similar work [13], [15], [22], [23] on synthesis for STL and Event-based STL specifications using efficient control barrier functions focused on open workspaces that do not contain static obstacles. Others [12], [14] do consider static obstacles – they use mixed-integer linear programs to find a trajectory that avoids walls and obstacles to satisfy an STL task. However, the computational complexity associated with solving such programs makes it difficult to run in real-time in the presence of external disturbances and uncontrolled events. In this paper we provide a framework to satisfy an Event-based STL specification, in which the robot is reactive to external events, in a computationally efficient manner such that a trajectory can be found through complex environments.

Assumptions: *Environment:* We assume the map containing static obstacles such as walls is known. *Robots:* All

Manuscript received: February, 24, 2022; Revised June, 2, 2022; Accepted July, 1, 2022. 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.

controlled robots are holonomic with linear dynamics and have knowledge of the state of all other controlled robots, but not their task (goals, safety constraints). *Initial state:* We assume that the Event-based STL specification is not trivially violated by all trajectory starting at the given initial robots' and environment states.

Contributions: Expanding on [15], we increase the expressive capabilities of Event-based STL and allow for the satisfaction of tasks in complex environments. At runtime, we provide feedback on the feasibility and status of an execution. We present three main contributions: 1) an enriched definition of Event-based STL to describe tasks that include disjunction, 2) a control synthesis framework for Event-based STL specifications with disjunctions in a complex environment that leverages the robust semantics of STL to find a robust execution, and 3) pre-failure warnings and suggestions given to a user at runtime on the feasibility and status of the satisfaction of an Event-based STL specification. We demonstrate our control synthesis and pre-failure warnings through physical and simulated multi-robots systems.

II. Preliminaries

A. System Model

We consider a discrete time dynamical system representing the position of a robotic system at time t:

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

where $f : \mathbb{R}^n \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ are locally Lipschitz continuous functions, $\mathbf{x}_t \in \mathbb{R}^n$ is the state of the system, and $\mathbf{u}_t \in \mathbf{U} \subseteq \mathbb{R}^m$ is the bounded input.

B. Environment Representation

We consider satisfying specifications in a complex environment which include static and dynamic obstacles. Static obstacles such as walls are obstacles whose positions do not change during execution. Dynamic obstacles such as humans and other robots are obstacles whose positions are known and can change during execution. The positions of static obstacles (collection of line segments) M, and a roadmap $\mathcal{G} = (V, E)$ [24] of the workspace connecting all areas of the map are given a priori.

We define the positions of dynamic obstacles in the environment as $\mathbf{x}_{dyn} \in \mathbb{R}^k$ and denote $\mathbf{X}_t = (\mathbf{x}_t, \mathbf{x}_{dyn,t})$ for brevity in the following. We assume all robots know $\mathbf{x}_{dyn}$ at all times, but cannot control them. In addition, we capture task-relevant uncontrolled discrete environment events such as alarms or other external environment signals.

C. Control Barrier Functions for STL

Control Barrier Functions (CBFs) were first proposed in [25] to define safe-sets for a system and ensure that the set is forward invariant without having to find the entire reachable set. A set is forward invariant if a trajectory can be found which keeps the system inside the set if the system starts inside of the set and outside of the set if the system starts outside of the set [26], [27].

Work in [13] leveraged the properties of these forward invariant safe-sets to generate control for robots satisfying a subset of Signal Temporal Logic (STL) tasks; they use time-varying CBFs to ensure an STL specification is satisfied in the given time constraints. To satisfy a specification, [13] describes how to create a valid CBF cbf for each predicate μ and its corresponding predicate function $h(\mathbf{X}_t)$ (discussed further in Sec. III). Leveraging [13], and similar to [15], in this paper we create CBFs for individual predicates and for conjunctions of predicates, based on Lemma 2 in [13].

D. Linear Temporal Logic and Büchi Automata

An LTL formula γ [1] is constructed from Boolean propositions $\pi \in AP$ where AP is a set of atomic propositions. The syntax of LTL 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 $\neg$ ("not") is negation, $\vee$ ("or") is disjunction, X is the temporal operator "Next", and U is the temporal operator "Until". We define conjunction ("and") as $\gamma_1 \wedge \gamma_2 \equiv \neg(\neg\gamma_1 \vee \neg\gamma_2)$ and implication ("if") as $\gamma_1 \Rightarrow \gamma_2 \equiv \neg\gamma_1 \vee \gamma_2$. From U we can create "Eventually" ($F\gamma \equiv TrueU\gamma$) and "Always" ($G\gamma \equiv \neg F\neg\gamma$). The semantics of LTL are defined over an infinite sequence $\sigma = \sigma_1, \sigma_2, ...$ where σ_i is the set of atomic propositions that are $True$ at position i, $\sigma_i \subseteq AP$. The full semantics of LTL can be found in [1].

A non deterministic Büchi automata is a tuple $B = (S, s_0, \Sigma, \delta, F)$ where S is a finite set of states, $s_0 \in S$ is the initial state, Σ is a finite input alphabet, $\delta : S \times \Sigma \to 2^S$ is the transition function, and $F \subseteq S$ is a set of accepting states. An execution of B on an infinite input word $\omega = \omega_1, \omega_2, ..., \omega_j \in \Sigma$ is an infinite sequence of states $s_0, s_1, s_2, ...,$ s.t. $\forall j \geq 1, s_j \in \delta(s_{j-1}, \omega_j)$. A run of B is accepting if and only if $\inf(\omega) \cap F \neq \emptyset$ where $\inf(\omega)$ is the set of states that are visited infinitely often on the input word ω. Given an LTL formula γ over AP, we can create a Büchi automata, B_γ that accepts a run if and only if it satisfies γ [28]. The transitions of B_γ are typically labeled with subsets of AP but they can also be viewed as Boolean formulas $\boldsymbol{\sigma}_{s_i,s_j}$ over the set AP.

III. Event-based STL with Disjunction

Event-based STL [15] is defined over predicates $\mu \in \{True, False\}$ whose truth value are defined by the evaluation of predicate functions $h(\mathbf{X}_t)$.

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

In this paper, building on [15], we increase the expressivity of Event-based STL through the addition of disjunction in φ, thus enabling choice in the control decisions of the robots.

Syntax: The new definition of Event-based STL is:

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

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

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

where Ψ is an Event-based STL specification, φ is a Boolean, negation normal form formula over predicates, α represents Boolean formulas over uncontrolled external environment propositions $\pi \in AP$, G is shorthand for $G_{[0,\infty]}$, and $\{a, b\} \in \mathbb{R}^+$ are timing bounds of a formula. Though we consider the interval $[a, b]$ to be in continuous time, during

simulation and physical demonstrations we evaluate the system $\mathbf{X}_t$ and environment events α at a set sampling rate. To ensure environments events are captured, we assume that environment events last longer than the sampling rate.

This syntax allows for disjunction in predicates φ as well as in temporal operators. For example, the specification $G(\alpha_1 \rightarrow (G_{[a,b]}\varphi_1 \vee F_{[a,b]}(\varphi_2 \vee \varphi_3)))$ is now a valid Event-based STL specification using the new syntax.

Semantics: The semantics of Event-based STL, presented in Table I, are defined over $(\mathbf{X}_t, \sigma_t)$. $\mathbf{X}_t$ represents the state of the system $\mathbf{x}_t$ and dynamic obstacles $\mathbf{x}_{dyn,t}$ at time t, and σ_t represents the set of propositions in AP that are $True$ at time t. We use the semantics of $U_{[a,b]}$ from [15] where φ_1 must be $True$ $\forall t_1 \in [t+a, t_2]$. This differs from the definition of Until in [2] where φ_1 is $True$ $\forall t_1 \in [t, t_2]$. The Until operator from [2] is equivalent to the formula $\varphi_1 U_{[a,b]}\varphi_2 \wedge G_{[0,a]}\varphi_1$ in our formulation. While we allow $\neg\mu$, in practice we replace all such occurrences with a new predicate $\widehat{\mu}$ such that $\widehat{h}(\mathbf{X}_t) = -h(\mathbf{X}_t)$. In the following, we assume no negation on predicates.

$(\mathbf{X}_t, \sigma_t) \models \mu$	$\Leftrightarrow h(\mathbf{X}_t) \geq 0$
$(\mathbf{X}_t, \sigma_t) \models \neg\mu$	$\Leftrightarrow h(\mathbf{X}_t) < 0$
$(\mathbf{X}_t, \sigma_t) \models \varphi_1 \wedge \varphi_2$	$\Leftrightarrow (\mathbf{X}_t, \sigma_t) \models \varphi_1$ and $(\mathbf{X}_t, \sigma_t) \models \varphi_2$
$(\mathbf{X}_t, \sigma_t) \models \varphi_1 \vee \varphi_2$	$\Leftrightarrow (\mathbf{X}_t, \sigma_t) \models \varphi_1$ or $(\mathbf{X}_t, \sigma_t) \models \varphi_2$
$(\mathbf{X}_t, \sigma_t) \models \pi$	$\Leftrightarrow \pi \in \sigma_t$
$(\mathbf{X}_t, \sigma_t) \models \neg\alpha$	$\Leftrightarrow (\mathbf{X}_t, \sigma_t) \not\models \alpha$
$(\mathbf{X}_t, \sigma_t) \models \alpha_1 \wedge \alpha_2$	$\Leftrightarrow (\mathbf{X}_t, \sigma_t) \models \alpha_1$ and $(\mathbf{X}_t, \sigma_t) \models \alpha_2$
$(\mathbf{X}_t, \sigma_t) \models F_{[a,b]}\varphi$	$\Leftrightarrow \exists t_1 \in [t+a, t+b] s.t.\ (\mathbf{X}_{t_1}, \sigma_{t_1}) \models \varphi$
$(\mathbf{X}_t, \sigma_t) \models G_{[a,b]}\varphi$	$\Leftrightarrow \forall t_1 \in [t+a, t+b],\ (\mathbf{X}_{t_1}, \sigma_{t_1}) \models \varphi$
$(\mathbf{X}_t, \sigma_t) \models \varphi_1 U_{[a,b]}\varphi_2$	$\Leftrightarrow \exists t_2 \in [t+a, t+b] s.t.\ (\mathbf{X}_{t_2}, \sigma_{t_2}) \models \varphi_2$ and $\forall t_1 \in [t+a, t_2], (\mathbf{X}_{t_1}, \sigma_{t_1}) \models \varphi_1$
$(\mathbf{X}_t, \sigma_t) \models G(\alpha \Rightarrow \Psi)$	$\Leftrightarrow \forall t, (\mathbf{X}_t, \sigma_t) \not\models \alpha$ or $(\mathbf{X}_t, \sigma_t) \models \Psi$
$(\mathbf{X}_t, \sigma_t) \models \Psi_1 \wedge \Psi_2$	$\Leftrightarrow (\mathbf{X}_t, \sigma_t) \models \Psi_1$ and $(\mathbf{X}_t, \sigma_t) \models \Psi_2$
$(\mathbf{X}_t, \sigma_t) \models \Psi_1 \vee \Psi_2$	$\Leftrightarrow (\mathbf{X}_t, \sigma_t) \models \Psi_1$ or $(\mathbf{X}_t, \sigma_t) \models \Psi_2$

TABLE I: Semantics of Event-based STL

Example 1. *Consider a restaurant-like environment, shown in Fig 1. The task is for a team composed of three different types of robots to assist in running a restaurant. The "Host" robot ($\boldsymbol{x}_1$), upon sensing $lead$, is to go to a customer within 25 seconds and remain close to them until they eventually reach the dining area entrance within 60 seconds. The external environment event $lead$ signifies that a customer is sensed as waiting in the waiting area. The "Cleaning" robots ($\boldsymbol{x}_2$,$\boldsymbol{x}_3$) and "Server" robots ($\boldsymbol{x}_4$,$\boldsymbol{x}_5$) respond to $request_k$ of customers seated in the restaurant. When $request$ is sensed, one Cleaning and one Server robot must go to the location of the customer within 20 seconds. The position of the customers are known to all robots and denoted as $\boldsymbol{x}_{cstmr} = [x_{cstmr}, y_{cstmr}]$. In simulation demonstrations, we vary the number of customers and $requests$ the robots must respond to. During execution all robots must avoid collision with each other and all walls in the environment at all time.*

We encode this task as an Event-based STL specification $\Psi = \Psi_{host} \wedge \Psi_{request_k} \wedge \Psi_{collision_{ij}} \wedge \Psi_{wallAvoid_i}$:

- $\Psi_{host} = G(lead \Rightarrow (F_{[0,25]}(\| \boldsymbol{x}_{1,t} - \boldsymbol{x}_{cstmr,1,t} \| < 1) \wedge (\| \boldsymbol{x}_{1,t} - \boldsymbol{x}_{cstmr,1,t} \| < 1) U_{[25,60]}(\| \boldsymbol{x}_{1,t} - [1.75, -1] \| < 1)))$
- $\Psi_{request_k} = G(request_k \Rightarrow F_{[0,20]}((((\| \boldsymbol{x}_{2,t} - \boldsymbol{x}_{cstmr,k,t} \| < 1) \vee (\| \boldsymbol{x}_{3,t} - \boldsymbol{x}_{cstmr,k,t} \| < 1)) \wedge ((\| \boldsymbol{x}_{4,t} - \boldsymbol{x}_{cstmr,k,t} \| < 1) \vee (\| \boldsymbol{x}_{5,t} - \boldsymbol{x}_{cstmr,k,t} \| < 1))))$
- $\Psi_{collision} = G_{[0,\infty]}(\| \boldsymbol{x}_{i,t} - \boldsymbol{x}_{j,t} \| > 0.05)\ , \forall i \neq j$
- $\Psi_{wallAvoid_i} = G_{[0,\infty]}(min(\| \boldsymbol{x}_{i,t} - M \|) > 0.1)\ , i = (1, 2, \ldots, 5)$

where $\boldsymbol{x}_{i,t} = [x_{i,t}, y_{i,t}]$ for robot i, the dynamics of each robot is described by (1), and point $[1.75, -1]$ represents the entrance to the dining area.

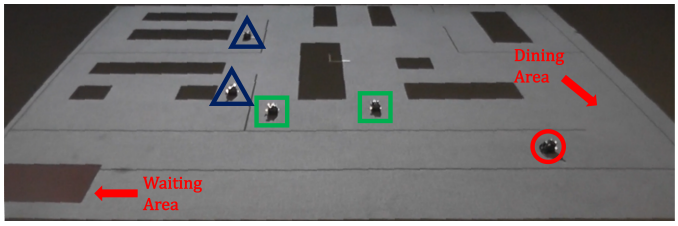

Fig. 1: Workspace for Example 1. The black lines are walls, black rectangles are tables. The robots: $Host$ (red circle), $Cleaning$ (blue triangles), and $Server$ (green squares).

IV. Problem Formulation

Given an Event-based STL specification Ψ, a system in the form of (1), discrete environment events $\pi \in AP$, continuous environment state $\mathbf{x}_{dyn}$, and a complex environment M with a roadmap $\mathcal{G}$, we find a control strategy $\mathbf{u}$ such that the system satisfies Ψ while taking into account robustness. Here, we do not use the STL robustness from [14], but rather define an instantaneous robustness metric, which measures how robustly a specification is satisfied at a given time step.

The specification may require the system to react to uncontrolled environment events and signals $(AP, \mathbf{x}_{dyn})$, thus the system may not be able to complete its task. In such cases, a prioritization scheme to schedule the satisfaction of tasks based on the robustness of the Event-based STL formula is needed. If a task cannot be completed as required, feedback to the user may give insight to potential problems. In the following sections we provide a prioritization scheme and methods of notifying users of potential future violations during an execution before a specification is violated.

V. Control Synthesis

Control synthesis from an Event-based STL specification is described in Fig. 2. The contributions of this paper are Sections V-A, V-B, V-D, and VI. Given Ψ_{STL}, we first abstract it as an LTL formula γ and create a Büchi Automaton B_γ (Sec. V-A). Based on the transitions in B_γ, we choose a trace that leads to an accepting state that can be visited infinitely often (Sec. V-B). Given this transition, we activate CBFs associated with the labels on the chosen transition using CBF templates. We use the activated CBFs to find a controller for each robot (Sec. V-C). During execution, we provide pre-failure warnings (Sec. VI).

A. Abstracting Ψ_{STL} to γ

Given Ψ_{STL}, we abstract it as an LTL formula γ (Algo. 1) over $AP \cup \Pi_\mu$, where AP is the same as for Ψ_{STL} and Π_μ is a set of propositions corresponding to abstractions

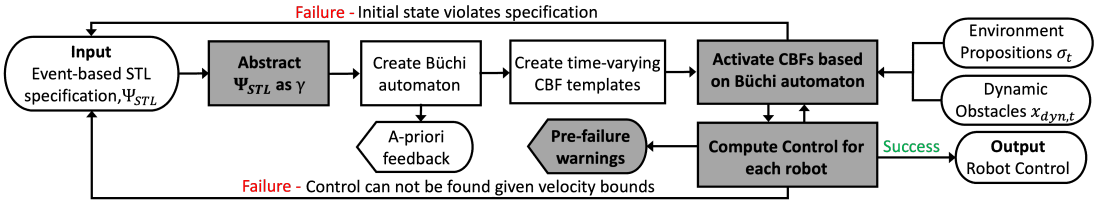

Fig. 2: Synthesis framework for Event-Based STL specifications. This paper's contributions are the shaded boxes.

of predicates in Ψ_{STL}, as defined below. The propositions $\pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}} \in \Pi_\mu$ contain information about the predicate that they abstract, its timing constraints, and dependence on other propositions, to be used in the construction of the CBFs (Sec. V-C). To create the proposition, we build a parse tree from Ψ_{STL}, with the leaves being predicates μ_i and the parents being the temporal operators and external environment events, and use its structure to abstract μ_i.

Algorithm 1: Abstracting predicates μ_i

Input : Ψ_{STL}
Output: Π_μ
1 **for** $\mu_i \in \Psi_{STL}$ **do**
2 $\quad \varphi_{unt} = searchUntil(\mu_i, \Psi_{STL})$;
3 $\quad$ **if** $\varphi_{unt} \neq \emptyset$ **then**
4 $\quad\quad [a,b] = timingFromRight(\mu_i, \Psi_{STL})$;
5 $\quad$ **else**
6 $\quad\quad [a,b] = timingFromLeft(\mu_i, \Psi_{STL})$;
7 $\quad$ **end**
8 $\quad \alpha_i = findEvents(\mu_i, \Psi_{STL})$;
9 $\quad \pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}} = makeProp(\mu_i, [a,b], \alpha_i, \varphi_{unt})$;
10 $\quad \Pi_\mu = \pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}} \cup \Pi_\mu$
11 **end**

***searchUntil* (Line 2)**: The Until operator, $\varphi_1 U_{[a,b]} \varphi_2$, is special in that the timing constraints of φ_1 depend on another formula φ_2, therefore to ensure correct execution of φ_1, we must capture this dependence. Here, for each predicate $\mu_i \in \Psi_{STL}$, we first determine if the parent of μ_i is the temporal operator U and if it precedes the U in Ψ_{STL}, that is, if μ_i exists in φ_1. From the semantics of Event-based STL, $\varphi_1 U_{[a,b]} \varphi_2$ is satisfied if φ_1 is $True$ $\forall t_1 \in [t+a, t_2]$, where t_2 is the time φ_2 becomes $True$. To capture the dependence of φ_1 on φ_2, we denote $\varphi_{unt} = \varphi_2$ for all propositions abstracting a predicate $\mu_i \in \varphi_1$. During execution, we use the truth value of φ_{unt} to update the timing bounds of μ_i, $[t+a, t_2]$. If μ_i is in φ or φ_2 in formulas of the form $G_{[a,b]}\varphi$, $F_{[a,b]}\varphi$, or $\varphi_1 U_{[a,b]} \varphi_2$, then φ_{unt} is empty.

***timingFromRight* (Line 4)**: If φ_{unt} is not empty, the timing bounds $[a,b]$ is from the first U immediately to the right of μ_i in Ψ_{STL}. For example, in Ψ_{host}, when abstracting $\| \mathbf{x}_{1,t} - \mathbf{x}_{cstmr,1,t} \| < 1$ in the Until, $[a,b] = [25, 60]$.

***timingFromLeft* (Line 6)**: If φ_{unt} is empty, the timing bounds $[a,b]$ are from the first temporal operator immediately to the left of μ_i. For example, in Ψ_{host}, when abstracting $F_{[0,25]}(\| \mathbf{x}_{1,t} - \mathbf{x}_{cstmr,1,t} \| < 1)$, $[a,b] = [0, 25]$.

***findEvents* (Line 8)**: The timing bounds of a predicate may depend on the timing of external environment events. While abstracting μ_i we capture the label of these events so that during execution their timings can be used to construct the appropriate time bounds of the CBF. The variable α_i represents the external environment events associated with a predicate μ_i. Due to the formula structure, one can abstract a single formula over the truth values of all external events in a subformula $G(\alpha_i \Rightarrow \Psi)$. This formula is a conjunction of all α that appear in the (possibly nested) subformula.

For example, in $G(\mathbf{A} \Rightarrow G(\mathbf{B} \Rightarrow (\varphi_1 \vee \varphi_2)))$, we choose $\alpha_i = \mathbf{A} \wedge \mathbf{B}$ for all predicates μ_i in φ_1 and φ_2.

***makeProp* (Line 9)**: Given the timing bounds of μ_i and other information relevant to the timing bounds such as α_i and φ_{unt}, we create the abstracted proposition $\pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}}$ and add it to the set Π_μ.

To create γ we replace all instances of μ_i with their abstracted proposition $\pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}}$. We then replace the temporal operators $F_{[a,b]}$ with F, $G_{[a,b]}$ with G and $U_{[a,b]}$ with U. The result is an abstraction of Ψ_{STL} as an LTL formula γ with propositions that capture the required timing and triggers. In Example 1, Ψ_{host} is abstracted as:

$$\begin{aligned}\gamma_{host} \; &= G(lead \Rightarrow \pi_{\mu_1,[0,25],lead} \wedge \\ &\quad \pi_{\mu_2,[25,60],lead,\varphi_{\mu_3}} U \pi_{\mu_3,[25,60],lead}).\end{aligned} \tag{7}$$

Where $\mu_1 = \mu_2 = (\| \mathbf{x}_{1,t} - \mathbf{x}_{cstmr,1,t} \| < 1)$ and $\mu_3 = (\| \mathbf{x}_{1,t} - [1.75, -1] \| < 1)$.

B. Choosing a Transition in the Büchi Automaton

From γ we create a Büchi automata B_γ (Sec. II-D) using Spot [28]. We represent the label of a transition in B_γ from state s_i to s_j, $\boldsymbol{\sigma}_{s_i,s_j}$, as a Boolean formula in disjunctive normal form over $\Pi_\mu \cup AP$. We separate $\boldsymbol{\sigma}_{s_i,s_j}$ by the disjunction operators resulting in Boolean formulas $\boldsymbol{\sigma}_{sep,s_i,s_j}$ s.t. $\bigvee \boldsymbol{\sigma}_{sep,s_i,s_j} = \boldsymbol{\sigma}_{s_i,s_j}$. If any formula $\boldsymbol{\sigma}_{sep,s_i,s_j}$ is $True$, the transition from s_i to s_j is enabled.

Example 2. *Consider a simplified version of $\Psi_{request_k}$ from Example 1 where (8) is the Event-based STL formula and (9) is the abstracted LTL formula. Fig. 3 shows the Büchi automaton B_γ corresponding to (9).*

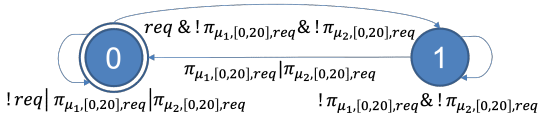

Fig. 3: Büchi automaton for (9). State 0 is an accepting state.

$$\begin{aligned}\Psi_{req} = G(req \Rightarrow F_{[0,20]}((&(\| \boldsymbol{x}_{1,t} - [1,1] \| < 1) \\ &\vee (\| \boldsymbol{x}_{2,t} - [1,1] \| < 1)))),\end{aligned} \tag{8}$$

$$\gamma_{req} = G(req \Rightarrow F(\pi_{\mu_1,[0,20],req} \vee \pi_{\mu_2,[0,20],req})). \tag{9}$$

When executing, at each time step we choose a trace in B_γ to an accepting state in F. There may exist multiple traces to an accepting state and the synthesis algorithm must decide

Algorithm 2: Choosing a transition in a B_γ

Input : $\sigma_t, \mathbf{X}_t, B_\gamma, h_i(\mathbf{X}_t), prevS$
Output: $\Pi_{\mu_{act}}, currS$
1 $possCurrS = updateS(prevS, h_i(\mathbf{X}_t), \mathbf{X}_t, \sigma_t, B_\gamma)$;
2 $F_{reach} = reachableAccepting(B_\gamma, \sigma_t, possCurrS)$;
3 **for** $s \in possCurrS$ **do**
4 $\quad paths = findAllPaths(s, F_{reach}, \sigma_t)$;
5 $\quad allPaths = paths \cup allPaths$;
6 **end**
7 $\mathbf{\Sigma}_{firstT} = findPotTransitions(allPaths)$;
8 $(\Pi_{\mu_{act}}, currS) = selectTrans(\mathbf{\Sigma}_{firstT}, h_i(\mathbf{X}_t))$;

which transition to choose and which predicates to activate. This process is described in Algo. 2.

***updateS* (line 1):** At each time step, given the previous state in B_γ, $prevS$, the state of the controlled and uncontrolled parts of the system, $\mathbf{X}_t$ and σ_t, and the transition labels $\boldsymbol{\sigma}_{sep,s_i,s_j}$, we determine the set of possible states the system is in, $possCurrS \subseteq S$ by evaluating which formulas leading out of $prevS$ are currently $True$. In Example 2, from state 0, if req is $True$ while $\pi_{\mu_1,[0,20],req}$ and $\pi_{\mu_2,[0,20],req}$ are $False$ (neither robot 1 or 2 are within 1 unit of [1,1]), then $possCurrS = \{1\}$. This means that the system must be in state 1 as it cannot remain in state 0.

***reachableAccepting* (line 2):** Next, we find the set of accepting states,$F_{reach} \subseteq F$, that can be reached and visited infinitely often from $s \in possCurrS$, assuming σ_t does not change. That is, there exists a path in B_γ from $s \in possCurrS$ to F_{reach} that is labeled with σ_t. In Example 2, $F_{reach} = \{0\}$ as state 0 is the only state that can be reached from state 1 and is an accepting state.

***findAllPaths* (line 4-5):** We then find the set of paths, $allPaths$, from states in $possCurrS$ to states in F_{reach} (line 4-5). The set $allPaths$ represents all traces through B_γ that lead to $F \in F_{reach}$. To reduce the number of paths evaluated, we only consider simple paths. That is, paths where each state is visited no more than once. From state 1 in Fig. 3, $allPaths = \{\{1, 0\}\}$ as there is only one path to the accepting state 0 ignoring repeating states.

***findPotTransitions* (line 7):** We determine the set of all possible first transitions $\mathbf{\Sigma}_{firstT}$ for each $path \in allPaths$ (line 7). For Example 2, there is only one $path \in allPaths$ which contains two formulas that lead to a transition. Therefore $\mathbf{\Sigma}_{firstT} = \{\pi_{\mu_1,[0,20],req}, \pi_{\mu_2,[0,20],req}\}$.

***selectTrans* (line 8):** For each possible transition formula $\boldsymbol{\sigma}_{sep,firstT} \in \mathbf{\Sigma}_{firstT}$, we create the set $\Pi_{\boldsymbol{\sigma},True} \subseteq \Pi_\mu$ which is the set of propositions in Π_μ that must be $True$ for the transition $\boldsymbol{\sigma}_{sep,firstT}$ to occur. These propositions capture which CBFs need to be activated, as described next. In addition to the propositions labeling the transition, we examine propositions in $\Pi_{\boldsymbol{\sigma},True}$ that appear on the right hand side of an Until operator in Ψ_{STL}. The LTL semantics of $\varphi_1 U \varphi_2$ require φ_1 to be $True$ until φ_2 becomes $True$ but does not require both to be $True$ at the same time. Since the semantics of the Büchi automaton assume instantaneous transitions, we need to take special care when executing transitions where φ_2 is $True$ – since in our framework transitions are not instantaneous (they require activation of CBFs), we need to ensure φ_1 remains $True$ until the transition is complete. We do that by adding to $\Pi_{\boldsymbol{\sigma},True}$ all propositions $\pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}}$ where $\varphi_{unt} \in \Pi_{\boldsymbol{\sigma},True}$. This ensures that φ_1 remains $True$ for specifications of the form $\varphi_1 U \varphi_2$. For example, for the until subformula in Ψ_{host}, if a transition has $\pi_{\mu_3,[25,60],lead} \in \Pi_{\boldsymbol{\sigma},True}$, then $\pi_{\mu_2,[25,60],lead,\varphi_{\mu_3}}$ is added to $\Pi_{\boldsymbol{\sigma},True}$.

Instantaneous robustness: We choose the next transition the system should take, based on $\boldsymbol{\sigma}_{sep,firstT} \in \mathbf{\Sigma}_{firstT}$ and an instantaneous robustness score $\rho : \Pi_\mu \to \mathbb{R}$, indicating how robustly the system can currently satisfy a proposition in $\Pi_{\boldsymbol{\sigma},True}$. This is different from the robust semantics of STL [14] which is defined over a complete execution, wrt the entire formula.

We compute ρ only for propositions in $\Pi_{\boldsymbol{\sigma},True}$, i.e. propositions that must be $True$ for the transition to occur; since we replace predicates μ that appears in negation in Ψ_{STL} with $\widehat{\mu}$ such that $\widehat{h}(\mathbf{X}_t) = -h(\mathbf{X}_t)$, the STL formula does not contain predicates that must be $False$. This means that propositions in $\boldsymbol{\sigma}_{sep,firstT}$ that are $False$ are not required to remain $False$ to satisfy the specification; therefore their robustness will not influence task success and we do not calculate a robustness score for those propositions.

We compute ρ for each proposition $\pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}} \in \Pi_{\boldsymbol{\sigma},True}$. Essentially, ρ captures how close we are to satisfying or violating μ_i:

$$\rho(\pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}}) = \begin{cases} h(\mathbf{X}_t) \text{if } \pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}} = True \\ t_{rem} \text{ if } \pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}} = False \end{cases} \tag{10}$$

For propositions that are currently $True$, we define ρ as the distance to violating the predicate. For propositions that are currently $False$, but need to become $True$ before the systems completes the transition, we define ρ as the minimum time remaining t_{rem} for the predicate to become $True$ given the control bounds $\mathbf{U}$. We estimate this time using the distance D to the safe-set. We assume that the robot will travel at its maximum velocity $\mathbf{u}$. Here, t_e is the time at which α_i becomes $True$. If no α_i exists, $t_e = 0$.

$$t_{rem} = (t_e + b - t) - D/\parallel \mathbf{u} \parallel . \tag{11}$$

Given ρ for each proposition $\pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}} \in \Pi_{\boldsymbol{\sigma},True}$, we create a tuple $P(\Pi_{\boldsymbol{\sigma},True}) = (\rho_1, ..., \rho_k)$ for each $\boldsymbol{\sigma}_{sep,firstT} \in \mathbf{\Sigma}_{firstT}$, where $k = |\Pi_{\boldsymbol{\sigma},True}|$, is the number of predicates that must be $True$ for the transition to occur. When choosing the transition in B_γ to execute next, we first consider transitions with minimal k. By doing this we prioritize choosing transitions with the least number of CBFs that need to be activated. If multiple transitions have the same k, we compare the smallest ρ in each P (the least robust predicate) and choose the transition that has the maximum value, i.e. we maximize the minimal robustness of the transition. If there are multiple transitions with the same maximal minimum ρ, we look at the next smallest ρ until a single transition is chosen (line 8). If all robustness scores are equal for multiple transitions, we arbitrarily choose the first transition in the set.

Given the chosen transition, we define $\Pi_{\mu_{act}} = \Pi_{\boldsymbol{\sigma},True}$. This represents the propositions that are $True$ in the transition that maximizes robustness, and indicates which

CBFs should be activated. The state from which the chosen transition originates, $s_i \in possCurrS$ is labeled as $currS$ and used as an input to Algo. 2 in the next time step (labeled $prevS$). In Example 2, when in state 1, $\mathbf{\Sigma}_{firstT} = \{\pi_{\mu_1,[0,20],req}, \pi_{\mu_2,[0,20],req}\} = \{\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2\}$, $\Pi_{\boldsymbol{\sigma}_1,True} = \{\pi_{\mu_1,[0,20],req}\}, \Pi_{\boldsymbol{\sigma}_2,True} = \{\pi_{\mu_2,[0,20],req}\}$. If robot 1 is at $[0, 0]$ and robot 2 is at $[3, 3]$, assuming no static obstacles, $\parallel \mathbf{u} \parallel = 1$, $t_e = 0$, and $t = 5$, then $P_1 = \{13.59\}$ and $P_2 = \{11.17\}$ and we activate the CBF for robot 1.

C. Generating Control

Based on [13], we create controllers to satisfy propositions that must become $True$ $\pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}} \in \Pi_{\mu_{act}}$. Similar to [15], and included here for completeness, for each proposition, we create a CBF template, cbf_{μ_i}:

$$cbf_{\mu_i}(\mathbf{X}_t) = \frac{(t - t_e - a)h_i(\mathbf{X}_{t_e})}{b - a} - h_i(\mathbf{X}_{t_e}) + h_i(\mathbf{X}_t). \quad (12)$$

If φ_{unt} is not empty in $\pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}}$, b is equal to the time at which φ_{unt} becomes $True$. If it is empty, b is equal to the timing bound from $[a, b]$ in $\pi_{\mu_i,[a,b],\alpha_i,\varphi_{unt}}$.

Our CBF template (12) is a valid CBF for our system and predicates; it may not be valid for other systems or predicates. For (12), at $t = t_e + a$, $cbf_{\mu_i}(\mathbf{X}_t) = 0$. At $t = t_e + b$, $cbf_{\mu_i}(\mathbf{X}_t) = h_i(\mathbf{X}_t)$. The CBF value changes linearly with time such that, at the end of the time bound, the safe-set associated with the CBF is equal to the safe-set associated with the predicate μ_i. We find a control strategy for each robot using (13).

$$\min_{\mathbf{u}_i \in \mathbf{U}_i} \parallel \mathbf{u}_i - \hat{\mathbf{u}}_i \parallel s.t.$$
$$\frac{\partial cbf_{\Psi_i}(\mathbf{X}_t)^T}{\partial \mathbf{x}}(f(\mathbf{x}_t) + g(\mathbf{x}_t)\mathbf{u}_i) + \frac{\partial cbf_{\Psi_i}(\mathbf{X}_t)}{\partial t} \geq \quad (13)$$
$$-\nu(cbf_{\Psi_i}(\mathbf{X}_t)),$$

where $\nu : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ is a locally Lipschitz continuous class K function, i is the index of the robot, $\hat{\mathbf{u}}_i$ is a nominal controller, and cbf_{Ψ_i} is an approximation of the minimum cbf_j for all propositions π that are activated for the current transition. Eqn. (13) finds at each timestep, and for each robot, the closest controller to the nominal controller that ensures the specification is satisfied.

D. Choosing a Nominal Controller

We design our nominal controller such that it is more likely to avoid deadlocks; a deadlock can occur when static or dynamic obstacles are present on the path of the robot to its goal. For each robot we choose a nominal controller at each time step based on the set of its activated propositions and whether they may result in a conflict.

1) Non-conflicting propositions: A set of activated propositions for a single robot is non-conflicting if their safe-sets intersect and a solution to the control optimization problem can be found. To determine the nominal controller $\hat{\mathbf{u}}_i$ when activated propositions are non-conflicting, we first determine a goal p within the intersection of the safe-set using the predicate functions $h_i(\mathbf{X}_t)$. For simplicity, we choose goal p to be the closest point to the current state of the robot. For example, given the state of the system $\mathbf{x}_t = [0, 0, 0]$ and the predicates $y_t \geq 2, x_t \geq 3, \theta_t \geq \frac{\pi}{2}$ we choose a goal point $p = [3, 2, \frac{\pi}{2}]$. Here, p is the closest point at time t that satisfies all predicates. If static obstacles exist between p and $\mathbf{x}_t$ we use the roadmap $\mathcal{G}$ to find a path to p. We choose the nominal controller to be in the direction of the path to p with magnitude equal to the maximum control input.

2) Conflicting propositions: We consider propositions to be conflicting if the safe-sets of their predicate $h_j(\mathbf{X}_t)$ are non-intersecting but their time bounds $t_{bound_j} = [a_j + t_e, b_j + t_e]$ intersect. If activated propositions conflict for a single robot, the specification may be violated. In such cases, we reduce the chance of failure by choosing nominal controllers that prioritize a robust ordering of propositions that are currently $False$ but must become $True$. Propositions that are $True$ and must remain $True$, for example safety requirements, are automatically accounted for in the optimization (due to the forward invariance of the CBF).

Robustness prioritization: For each transition that has conflicting propositions that are $False$, we compute the robustness scores for all orderings of satisfying these propositions. For example, if a robot is to visit points a and b during the same time bounds, we compute two robustness tuples P. P_1 represents the robustness of visiting a and then b and P_2 represents the robustness of visiting b and then a. Essentially, we change D to include the full path (current location to a to b for P_1) when calculating ρ. We then choose the nominal controller based on the maximal robustness.

VI. Pre-Failure Warnings and Completeness

During an execution, the timing of uncontrolled external events, both discrete and continuous, may make the specification more difficult to satisfy. This can be caused by conflicting safe-sets for individual tasks or changes in the state of dynamic obstacles. One of the strengths of our approach is that, at runtime, we provide **pre-failure warnings** to users that indicate that a previously satisfiable specification may become unsatisfiable. We distinguish between three types of pre-failure warnings: warnings for tasks that must eventually be $True$ (liveness tasks), warnings for tasks that must always be $True$ (safety tasks), and potential violations of a specification if environment events change.

Liveness Tasks: For goals that must eventually be reached we examine all robustness scores ρ for the activated transition. If any robustness score is negative, this means that there is not enough time to satisfy all activated propositions and we give a pre-failure warning to the user. We consider this a pre-failure warning because the specification will not be violated until an individual CBF is violated.

Safety tasks: For constraints that must always be $True$ we evaluate the value of the CBFs at each time step; for satisfying the safety, all individual activated CBFs must be satisfied. We give a pre-failure warning when any activated CBF falls below a pre-determined threshold. This alerts the user when a robot may not be able to remain within the safe-set because it is near the boundary. If the safe-set moves or an obstacle disrupts the robot, the robot may not be able to remain within the safe-set and a violation could occur.

Environment changes: In addition to warnings due to the current transition being executed, we evaluate the worst-case

scenario of a change in ρ based on changes in the discrete environment events σ_t. At each time step we evaluate all the transitions out of the current state; if there exists a transition with conflicting CBFs such that the robot cannot reach the CBF safe-sets within the time bounds, from its current state, we provide a pre-failure warning. If any ρ is negative this means that there is not enough time for the specification to be satisfied if the environment propositions change and this transition is taken. In these cases we suggest changes in the timing bounds of the potentially violating task that will result in a positive ρ for the transition. In example 1 if $lead$ is sensed at $t = 0$ and at $t = 35$ the $Host$ robot has met the customer and is guiding them to the Dining area. If $lead$ is sensed again at $t = 35$ the $Host$ robot will have to guide the current customer to the dining area and meet the new customer within the same 25 seconds. Given the control bounds of the robot, it may not be possible for it to accomplish both tasks. In this case we give the pre-failure warning that if $lead$ is sensed, the task may not be satisfied, and we suggest how much extra time would be required for the specification to be satisfied. However, if the $Host$ robot has made enough progress towards the Dining area and there is enough time to respond to the a new customer, no pre-failure warning is given.

Completeness: Our framework attempts to find a control strategy that satisfies an Event-based STL specification with disjunctions, in complex environments, under control bounds. Our approach is sound, but not complete; there may be cases where we can no longer satisfy a specification when a solution may have existed had the controller made a different choice in the prioritization scheme.

VII. Simulation Demonstration

We simulate Example 1 with varying parameters (number of robots, number of customers the robots must respond to) to explore the scalability of our framework. We increase the number of Cleaning robots (2-4), Server robots (2-4), and customers (2-4). All simulations are run on a 2.3 GHz Quad-Core CPU with 8 GB of RAM. Table II captures our results including the number and types of robots, the number of predicates in the specification, the time to prepare a specification, the size of the Büchi automaton, and the time it takes to compute the control for the robots in a transition.

Prep Time is the time it takes to prepare the specification for execution; it is a one-time preprocessing step for a given specification. It includes running Algo. 1 for each predicate, creating a Büchi automaton from the LTL specification using Spot [28], and parsing the Büchi automaton. Algo.1 typically takes 0.7ms per predicate. Spot [28], which we use as an off-the-shelf tool, took 28 seconds to generate the Büchi automaton for our most complex specification. We parse the Büchi Automaton so that its transitions can be evaluated as $True$ or $False$ during execution. To reduce computation time during execution, we parse all possible transitions in B_γ even though they may not be evaluated during execution.

Computation time is the time it takes to choose a transition (Algo. 2) and generate control inputs for all robots (13). Table II captures the minimum and maximum computation times for one time step in an execution. A large increase in computation time can be seen when generating control with pre-failure warnings for more complex specifications. This is due to increase in possible combinations of uncontrolled external events and transitions. For each combination of external events, we calculate a worst-case scenario robustness which increases computation time. Computation also increases with the complexity of the Büchi automaton. We measure the complexity of a Büchi automaton by the number of conjunctive formulas that result in a transition when $True$.

VIII. Physical Demonstration

In the accompanying video we demonstrate our work using Anki Vector robots. The physical demonstration is of Example 1 with one $Host$ robot, two $Server$ robots, two $Cleaning$ robots, one customer in the waiting area, and two customers that can $requests$ robots. We use a teleoperated robot to represent the customer. We assume all robots are holonomic when generating control and use feedback linearization to control the nonlinear robots. The average computation time was 0.199 seconds with pre-failure warnings and 0.123 seconds without pre-failure warnings. When $lead$ is sensed, the $Host$ robot moves to the waiting area and then leads the customer to the dining area. Because the $Host$ robot is required to remain close to the customer before it eventually reaches the dining area, if the customer does not follow, as seen in the video, the $Host$ robot will have to wait. This can potentially lead to a specification violation. During execution, we provide pre-failure warnings, for example, while the $Host$ robot is leading the customer to the dining area, a pre-failure warning is given that if another customer is sensed in the waiting area the robot may not have enough time to satisfy the specification.

IX. Discussion

Prep Time: The prep time increases with the number of possible transitions in the Büchi automaton and predicates in a specification. Note that this step is only needed for each unique Büchi automaton, meaning that if the structure of the specification does not change, prep time will not be repeated. Changing parameters such as the initial positions of the robots, the timing of external environment events, and $h(\mathbf{X}_t)$ of predicates do not require repeating this step.

Computation Time: Allowing disjunction in the specification increases not only the number of accepting traces in B_γ, but also the number of combinations of predicates that, when $True$, satisfy the transition label. This can be seen in Table II where the computation time for the same number of costumers (external events) increases with the number of robots (disjunction in $\Psi_{request_k}$). It is possible mitigate this increase in computation time through decentralization of the control process after the set of activated proposition are determined so that each robot generates control on their own given the propositions that are active.

STL synthesis: There are two main approaches to control synthesis from STL specifications: (i) using model predictive control with either mixed integer linear programs (e.g. [18], [29]) or convex quadratic programs [22] and (ii) using CBFs (e.g. [13], [15]). The former approach can provide guarantees, assuming bounded uncontrolled signals and a finite horizon, while the later is computationally more efficient.

No. of Customers	Host Robots	Cleaning Robots	Server Robots	Total Robots	No. of Predicates	Prep Time	Büchi States	No. of σ_{split}	Min/Max Comp. Time w/o Pre-F	Min/Max Comp. Time w/ Pre-F
2	1	2	2	5	22	5.378s	17	2434	0.024s/0.059s	0.039s/0.084s
		3	3	7	36	5.424s	17	4422	0.037s/0.075s	0.067s/0.141s
		4	4	9	53	5.776s	17	7090	0.057s/0.107s	0.119s/0.244s
3	1	2	2	5	26	7.926s	41	45,958	0.042s/.086s	0.102s/0.439s
		3	3	7	42	15.61s	41	150,718	0.116s/0.322s	0.348s/1.898s
		4	4	9	61	54.73s	41	398,950	0.321s/0.706s	0.859s/5.902s
4	1	2	2	5	30	48.25s	97	1,155,274	0.140s/0.398s	0.741s/6.256s
		3	3	7	48	1,266s	97	7,361,046	1.024s/3.102s	5.386s/81.172s

TABLE II: Computational results for the simulated scenario.

In our work, we use CBFs ([13]) due to their computational efficiency and ability to handle long time horizon specifications. For example, if we were to use the BluSTL [29] to encode our 5 robot, 2 costumer example, we would need to solve a mixed integer linear program with 7,150 constraints. Choosing CBFs though, we lose the guarantees that [18] can provide. We mitigate this by providing pre-failure warnings and considering transition robustness.

Our work builds on and extends that of [13] by adding reaction to external events, to static obstacles and adding disjunction. Tasks expressible in the grammar of [13] are also expressible in Event-based STL.

X. Conclusion

In this paper we enrich the definition of Event-based STL to include disjunctions. By doing so we capture a larger sets of tasks than before; specifically choice in robot behavior. To satisfy tasks with disjunction, we propose a control synthesis framework to find controllers that satisfy Event-based STL specifications in a way that maximizes transition robustness. In addition, we provide pre-failure warnings during execution to notify a user of potential failures before they occur. This feedback would allow the user to adjust the task if needed. In future work we will provide methods for using pre-failure warnings to modify a specification at run-time. We will also create methods for allowing users to add new tasks to a specification during execution. Our code is available at https://github.com/davidgundana/Event-based-STL.

References

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

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

[3] 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.

[4] 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*, no. 0, pp. 6235–6240, 2012.

[5] V. Raman and H. Kress-Gazit, "Synthesis for multi-robot controllers with interleaved motion," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4316–4321, 2014.

[6] 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.

[7] 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.

[8] 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*, pp. 211–236, 2018.

[9] 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.

[10] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Lecture Notes in Computer Science*, vol. 6246 LNCS, pp. 92–106, 2010.

[11] A. M. Jones, K. Leahy, C. Vasile, S. Sadraddini, Z. Serlin, R. Tron, and C. Belta, "ScRATCHS : Scalable and Robust Algorithms for Task-based Coordination from High-level Specifications," *International Symposium of Robotics Research*, pp. 1–16, 2019.

[12] Z. Liu, J. Dai, B. Wu, and H. Lin, "Communication-aware motion planning for multi-agent systems from signal temporal logic specifications," *Proceedings of the American Control Conference*, pp. 2516–2521, 2017.

[13] 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, 2019.

[14] V. Raman, M. Maasoumy, A. Donze, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," *Proceedings of the IEEE Conference on Decision and Control*, pp. 81–87, 2014.

[15] 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.

[16] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC 2015*, pp. 239–248, 2015.

[17] A. Pacheck, S. Moarref, and H. Kress-Gazit, "Finding missing skills for high-level behaviors," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10335–10341, IEEE, 2020.

[18] V. Raman and H. Kress-Gazit, "Automated feedback for unachievable high-level robot behaviors," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 5156–5162, 2012.

[19] K. Chatterjee, T. A. Henzinger, and B. Jobstmann, "Environment assumptions for synthesis," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5201 LNCS, pp. 147–161, 2008.

[20] A. Cimatti, M. Roveri, V. Schuppan, and A. Tchaltsev, "Diagnostic information for realizability," in *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pp. 52–67, 2008.

[21] R. Alur, S. Moarref, and U. Topcu, "Pattern-based refinement of assume-guarantee specifications in reactive synthesis," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 501–516, Springer, 2015.

[22] 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.

[23] 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.

[24] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[25] P. Wieland and F. Allgöwer, "Constructive safety using control barrier functions," *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 7, no. PART 1, pp. 462–467, 2007.

[26] 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.

[27] 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.

[28] 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.

[29] A. Donzé, V. Raman, G. Frehse, and M. Althoff, "Blustl: Controller synthesis from signal temporal logic specifications.," *ARCH@ CPSWeek*, vol. 34, pp. 160–8, 2015.