To appear in the IEEE Transaction on Robotics, 2016.

Iterative Temporal Planning in Uncertain Environments with Partial Satisfaction Guarantees

Morteza Lahijanian, *Member, IEEE,* Matthew R. Maly, Dror Fried, Lydia E. Kavraki, *Fellow, IEEE,* Hadas Kress-Gazit, *Senior Member, IEEE,* and Moshe Y. Vardi, *Fellow, IEEE*

Abstract

This work introduces a motion-planning framework for a hybrid system with general continuous dynamics to satisfy a temporal logic specification consisting of co-safety and safety components in a partially unknown environment. The framework employs a multi-layered synergistic planner to generate trajectories that satisfy the specification and adopts an iterative replanning strategy to deal with unknown obstacles. When the discovery of an obstacle renders the specification unsatisfiable, a division between the constraints in the specification is considered. The co-safety component of the specification is treated as a *soft* constraint, whose partial satisfaction is allowed, while the safety component is viewed as a *hard* constraint, whose violation is forbidden. To partially satisfy the co-safety component, inspirations

This work was supported in part by NSF Expeditions 1139011, NSF NRI 1317849, NSF CCF 1018798, the U.S. Army Research Laboratory, and the U.S. Army Research office under grant number W911NF-09-1-0383. A subset of this paper was presented at the 16th International Conference on Hybrid Systems: Computation and Control, Philadelphia, PA, 2013.

M. Lahijanian was with the Department of Computer Science, Rice University, Houston, TX 77005 USA at the time this work was performed. He is now with the Department of Computer Science, University of Oxford, Oxford OX12JD UK (email: morteza.lahijanian@cs.ox.ac.uk).

M. R. Maly was with the Department of Computer Science, Rice University, Houston, TX 77005 USA at the time this work was performed. He is now with Google Inc., Mountain View, CA 94043 USA (email: matthew.r.maly@gmail.com).

D. Fried, L. E. Kavraki, and M. Y. Vardi are with the Department of Computer Science, Rice University, Houston, TX 77005 USA (email: {dror.fried,kavraki,vardi}@rice.edu).

H. Kress-Gazit is with the Sibley School of Mechanical and Aerospace Engineering at Cornell University (email: hadaskg@cornell.edu).

are taken from indoor-robotic scenarios, and three types of (unexpressed) restrictions on the ordering of sub-tasks in the specification are considered. For each type, a partial satisfaction method is introduced, which guarantees the generation of trajectories that do not violate the safety constraints while attending to partially satisfying the co-safety requirements with respect to the chosen restriction type. The efficacy of the framework is illustrated through case studies on a hybrid car-like robot in an office environment.

Index Terms

motion planning, formal methods, control synthesis, temporal logic, partial satisfaction, hybrid systems.

I. INTRODUCTION

In classical motion planning, a robotic system is asked to move from position A to position B while avoiding obstacles. Many works have been developed in the robotics community to solve this problem efficiently (e.g., [1]–[8]). Nevertheless, to one day have autonomous robots operating in unstructured environments, one must go beyond this foundational step of "A to B" motion planning and consider complex tasks and real robot dynamics. A dream would be to have a helper robot that can robustly execute commands such as "clean all the rooms on the second floor," or "prepare the conference room for a meeting." This paper takes a step towards realizing this dream by focusing on a motion-planning framework that encodes both rich tasks and complex dynamical systems and is capable of dealing with uncertainties in the environment.

In recent years, various computational frameworks for planning with temporal logic specifications have been developed to accommodate rich robot missions (e.g., [9]–[16]). Linear Temporal Logic (LTL) [17] is the most popular specification language in these works due to its expressiveness and well-studied model checking algorithms. This logic allows expressing *liveness* ("something good eventually happens") and *safety* ("something bad never happens") properties, which are natural in robotic tasks. Therefore, the employment of such a logic as a robotic specification language enables one to formally express complex missions, such as coverage of a set of goals ("visit A, B, and C in any order"), sequential goals ("visit A, B, and C in this order"), and temporal conditions on target visits ("first visit A or B and then eventually C. If D is ever reached, then always avoid B").

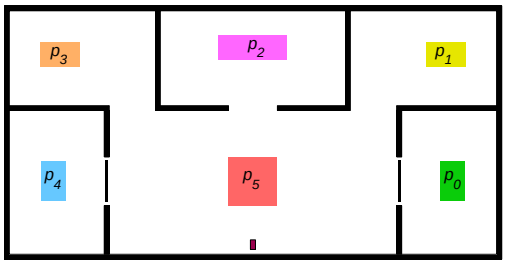


Fig. 1: A schematic representation of an office building consisting of a lobby and five rooms each with a door. In the instance captured in this figure, the doors of three rooms are open and two are closed. The properties of interest in this environment are represented by the rectangles labeled with $p_0, \ldots, p_5$. The robot is shown as a small rectangle directly below p_5 in the lobby.

Most of the existing works in motion planning with temporal logic specifications are based on the assumptions that the environment is static, and its full map is available. These assumptions, however, do not usually hold in real-world scenarios. For instance, consider a mobile robot in an office building that is to perform a set of janitorial tasks daily. In this example, it is reasonable to assume that some basic features of the environment are static (e.g., the floor plan of the office building), but the states of the office doors may vary from time to time. In such a scenario, the motion-planning framework needs to have the capability of dealing with unforeseen obstacles in the environment.

Recently, several works have been developed to address cases where the environment is partially unknown (e.g., [18]–[21]). The methods used in most of these studies are based on reactive synthesis. The objective in the framework of reactive synthesis is to construct a control strategy that accounts for every possible variation in the environment. In addition to the strong requirement of the knowledge of all the possible environmental changes, the reactive synthesis method suffers from a high computational cost [22]. Hence, when the number of the environmental uncertainties becomes large, the reactive synthesis problem can become too complex.

Another method to deal with partially unknown environments is iterative planning (e.g., [23], [24]). In this approach, a new plan is computed on-the-fly every time an unexpected environmental feature is observed. There are two major challenges in employing iterative approaches in planning from complex tasks. One challenge is to compute satisfying plans fast. The other

challenge is to deal with the question of what to do when the specification becomes unsatisfiable due to the discovery of an unexpected obstacle. Consider, for instance, the janitor robot example above in the office building whose schematic representation is shown in Fig. 1. The office environment consists of a lobby and five rooms, each with a door. The properties of interest in this office are shown as regions with the labels of p_i, where $i \in \{0, \ldots, 5\}$. They could represent office objects such as plants, a coffee machine, a blackboard, a desk, and a supply cabinet in the rooms and a demo area in the lobby. An example of a motion specification for this robot is as follows.

Specification 1: Do the following tasks in any order: water the plants and pick up the supply cabinet key in region p_0, turn off the coffee machine in p_1, and clean the blackboard in p_2. Then, go to region p_4 to pick up a duster from the supply cabinet before visiting p_3 to dust the desk. For safety reasons, always avoid the demo region p_5 in the lobby and never enter the meeting room with the blackboard (p_2) if the previously executed task was watering the plants (p_0).

In this example, the robot initially has no knowledge of the state of the office doors. It discovers their status as it moves in the environment. Note that (for the instance captured in this figure) the robot is not able to visit p_0 and p_4 since the doors of the corresponding rooms are closed. Thus, there are parts of Specification 1 that cannot be satisfied. Nevertheless, in such tasks, it is desired for the robot to continue with the mission even when the discovered obstacle prevents the robot from meeting some objectives of the specification as long as the safety requirements are not jeopardized.

In this paper, we consider such realistic scenarios of planning in a partially unknown workspace for robots with complex nonlinear dynamics. We focus on hybrid systems as they reveal the generality of our approach. Moreover, we study the meaning of partial satisfaction of a temporal logic specification with both liveness and safety requirements. Thus, the following problem emerges as a key challenge:

"*Given a mission expressed as a temporal logic specification consisting of liveness and safety properties for a robot in a partially unknown workspace, find a plan for the robot to satisfy the liveness objectives as closely as possible while always respecting the safety constraints*."

In previous work [23], we solved a simpler version of this problem where specifications with solely liveness objectives were considered. In that work, an iterative planning approach is used in

conjunction with a temporal logic planning framework with synergistic combination of discrete and continuous planners developed in [14]–[16]. The online planning is made possible in [23] by employing a hybrid state space abstraction method that utilizes a workspace decomposition [15] and by treating the automaton that represents the specification as a monitor at the top level of discrete planner. At the low level, a sampling-based planner is used (e.g., [4], [6]). Upon the discovery of an unknown obstacle, the framework generates a new abstraction and uses a metric over the automaton to compute a new plan that maximizes the satisfaction of the specification. The partial satisfaction method proposed in [23] guarantees that the robot meets the maximum number of the mission's objectives for only certain types of specifications such as coverage.

In this work, we extend the efforts in [23] and present a comprehensive planning framework for robots in partially unknown environments that accommodates safety specifications in addition to liveness. In this framework, the safety conditions are treated as *hard constraints*, whose full satisfaction is required, while the liveness requirements are viewed as *soft constraints* whose partial satisfaction is allowed. We present three methods of partial satisfaction for the liveness (soft) constraints to appropriately attend to user's (unexpressed) intentions in different types of specifications (e.g., sequential and temporal conditions on goal visits in addition to coverage) while still satisfying the safety (hard) constraints. Furthermore, we introduce an improved abstraction method that allows local updates to the discrete abstraction model instead of rebuilding it from scratch every time a new obstacle is observed. This modification enables faster replanning than the one presented in [23].

The novel contributions of this work are fourfold: (1) an iterative technique for replanning in the presence of unforeseen obstacles in the environment, (2) an integration of safety requirements into the planning framework for complex dynamical systems with temporal goals, (3) three methods of partial satisfaction to attend to user's intentions in the specification without violating safety, and (4) a method to identify and patch only the components of the abstraction that are affected by a newly discovered obstacle. Therefore, the proposed framework enables a complex robot modeled as a hybrid system with nonlinear continuous dynamics to modify its plan on-the-fly upon discovery of unknown obstacles to satisfy a temporal logic specification without having to return to its base and replan from scratch.

We illustrate this framework through case studies on a three-gear car-like robot with nonlinear continuous dynamics in an office environment. In the case studies, the robot was given LTL

specifications with both liveness and safety requirements and a partial map of the environment. We considered different types of specifications and varied the accuracy of the robot's initial map to demonstrate the efficacy of the framework.

II. PRELIMINARIES

A. Robot Hybrid Model

In this paper, we consider a general mobile robot whose dynamics are subject to restrictions in the regions of a partially unknown environment. We describe its motion in such an environment by the hybrid system $H = (S, s_0, \text{INV}, \text{SENSE}, E, \text{GUARD}, \text{JUMP}, U, \text{FLOW}, \Pi, L)$ [15], [23], [25], [26], where

- $S = Q \times X$ is the hybrid state space that is a product of a set of discrete modes, $Q = \{q_1, q_2, \ldots, q_m\}$ for some finite $m \in \mathbb{N}$, by a set of continuous state spaces $X = \{X_q \subseteq \mathbb{R}^{n_q} : q \in Q\}$;
- $s_0 \in S$ is the initial state;
- $\text{INV} = \{\text{INV}_q : q \in Q\}$, is the set of invariants, where $\text{INV}_q : X_q \to \{\top, \bot\}$ with $\top$ and $\bot$ corresponding to *true* and *false*, respectively;
- $\text{SENSE} : X_q \to \{\top, \bot\}$, is the sensing function that returns *true* if an unknown obstacle is detected;
- $E \subseteq Q \times Q$ describes discrete transitions between modes.
- $\text{GUARD} = \{\text{GUARD}_{q_i,q_j} : (q_i, q_j) \in E\}$, where $\text{GUARD}_{q_i,q_j} : X_{q_i} \times \{\top, \bot\} \to \{\top, \bot\}$ is a guard function that enables transitions between different modes given the continuous state of the robot and the unknown-obstacle detector readings (i.e., output of SENSE);
- $\text{JUMP} = \{\text{JUMP}_{q_i,q_j} : (q_i, q_j) \in E\}$, where $\text{JUMP}_{q_i,q_j} : X_{q_i} \to X_{q_j}$ is the jump function.
- $U = \{U_q \subset \mathbb{R}^{m_q} : q \in Q\}$ is the set of input spaces;
- $\text{FLOW} = \{\text{FLOW}_q : q \in Q\}$, where $\text{FLOW}_q : X_q \times U_q \times \mathbb{R}^{\geq 0} \to X_q$ is the flow function that describes the continuous dynamics of the system through a set of differential equations;
- Π is a set of atomic propositions;
- $L : S \to 2^{\Pi}$ is a labeling function assigning to each hybrid state possibly several elements of Π.

A pair $s = (q, x) \in S$ denotes a hybrid state of the system. $\text{FLOW}_q(x, u, t)$ gives the continuous state of the system when the input u is applied for t time units starting from state x. The evolution

of the robot represented by the system H is as follows. The robot starts evolving from its initial state $s(0) = s_0 = (q, x_0)$ according to the flow function $\text{FLOW}_q(x, u, t)$ with control $u \in U_q$. Let $t_{qq'}$ denote the time that the robot first hits the guard $\text{GUARD}_{qq'}$. Then, the system makes a transition to mode q'. The robot dynamics change to $\text{FLOW}_{q'}(x, u, t)$ at the continuous state $x(t_{qq'}) = \text{JUMP}(q, q')$, i.e., $s(t_{qq'}) = (q', \text{JUMP}(q, q'))$. Thus, the robot now moves according to $\text{FLOW}_{q'}$ for control $u \in U_{q'}$. This process goes on as long as the invariant function remains true. As soon as the invariant becomes false, the system terminates, and the robot stops moving. See Sec. III-B and VII for modeling of a robot with multiple modes of operation as a hybrid system.

In this work, we employ sampling-based techniques to plan for the hybrid system H. These methods generate trajectories of H by sampling control u and its associated time duration t. A finite trajectory obtained by such techniques can be represented as

$$\Xi = s_0 \xrightarrow{\xi(s_0,u_1,t_1)} s_1 \xrightarrow{\xi(s_1,u_2,t_2)} s_2 \cdots \xrightarrow{\xi(s_{n_\xi-1},u_{n_\xi},t_{n_\xi})} s_{n_\xi},$$

where $\xi(s, u, t)$ denotes a trajectory segment of the hybrid system starting from s under control u for the duration of t and $n_\xi \in \mathbb{N}$. Note that $\xi(s, u, t)$ is given by FLOW, JUMP, or a combination of FLOW and JUMP. In addition to being collision-free, we require ξ between two states s_i and s_{i+1} to contain at most one label change. Formally,

- if $L(s_i) = L(s_{i+1})$, then $L(s) = L(s_i)$ for every $s \in \xi(s_i, u_{i+1}, t_{i+1})$;
- if $L(s_i) \neq L(s_{i+1})$, then $\exists t' \in (0, t_{i+1}]$ such that $L(s \in \xi(s_i, u_{i+1}, t)) = L(s_i)$ for every $t \in [0, t')$, and $L(s \in \xi(s_i, u_{i+1}, t)) = L(s_{i+1})$ for every $t \in [t', t_{i+1}]$.

Note that the validity of each ξ can be checked through a simple modification to the existing collision checking functions of sampling-based motion planners (e.g., [5]).

The (observation) trace of trajectory Ξ is, hence, given by $L(s_0) \dots L(s_{n_\xi})$. To capture the number of consecutive repetitions of the hybrid state labels, we rewrite this trace as

$$\{L(s_0)\}^{i_0}\{L(s_{i_0})\}^{i_1}\{L(s_{i_0+i_1})\}^{i_2} \dots \{L(s_{i_0+\dots+i_{l-1}})\}^{i_l},$$

where $\{L(s)\}^{i_j}$ indicates i_j consecutive repetitions of $L(s)$ with $i_j \in \mathbb{N}$ for $j \in \{0, \dots, l\}$, $l \in \mathbb{N}$, and $l \leq n_\xi$. We define the *event-driven trace* of trajectory Ξ as

$$\bar{\sigma} = L(s_0)L(s_{i_0})L(s_{i_0+i_1}) \dots L(s_{i_0+\dots+i_{l-1}}).$$

Note that the event-driven trace of a trajectory captures only the change in the labels of the hybrid states and ignores the consecutive repetitions of the same label, i.e., $L(s_{I_j}) \neq L(s_{I_{j-1}})$,

where $I_j = \sum_{k=0}^{j} i_k$. Intuitively, this trace is the discrete observation of the trajectory Ξ and can be thought of as the task that the robot performs by executing Ξ. For the remainder of the paper, we refer to the event-driven trace of the hybrid system trajectory as the trace or word obtained by the robot.

B. Syntactically Co-safe and Safe LTL

We use syntactically co-safe and syntactically safe LTL to write the specifications of robotic tasks. Co-safe LTL is used to encode tasks for the robot to achieve (liveness) [14]–[16], [23], and safe LTL is used to encode behaviors for the robot to avoid. Their syntax and semantics are defined below.

Definition 1 (Co-safe Syntax): Let $\Pi = \{p_1, \ldots, p_k\}$ be a set of boolean atomic propositions. A *syntactically co-safe* LTL formula over Π is inductively defined as follows:

$$\varphi := p \,|\, \neg p \,|\, \varphi \vee \varphi \,|\, \varphi \wedge \varphi \,|\, \mathcal{X}\varphi \,|\, \varphi \mathcal{U} \varphi \,|\, \mathcal{F}\varphi$$

where $p \in \Pi$, $\neg$ (negation), $\vee$ (disjunction), and $\wedge$ (conjunction) are boolean operators, and $\mathcal{X}$ ("next"), $\mathcal{U}$ ("until"), and $\mathcal{F}$ ("eventually") are temporal operators.

Definition 2 (Safe Syntax): A *syntactically safe* LTL formula over Π is inductively defined as follows:

$$\varphi := p \,|\, \neg p \,|\, \varphi \vee \varphi \,|\, \varphi \wedge \varphi \,|\, \mathcal{X}\varphi \,|\, \mathcal{G}\varphi$$

where $p \in \Pi$, $\neg$ (negation), $\vee$ (disjunction), and $\wedge$ (conjunction) are boolean operators, and $\mathcal{X}$ ("next") and $\mathcal{G}$ ("always") are temporal operators.

Definition 3 (Semantics): The semantics of syntactically co-safe and safe LTL formulas are defined over infinite words over 2^{Π}. Let $\sigma = \{\tau_i\}_{i=0}^{\infty}$ be an infinite word with letters $\tau_i \in 2^{\Pi}$, and define $\sigma_i = \tau_0, \tau_1, \ldots, \tau_{i-1}$ and $\sigma^i = \tau_i, \tau_{i+1}, \ldots$. Then σ_i is a prefix of the word σ, and σ^i is a suffix of σ. The notation $\sigma \models \varphi$ indicates that σ satisfies formula φ and is inductively defined as follows.

- $\sigma \models p$ if $p \in \tau_0$;
- $\sigma \models \neg p$ if $p \notin \tau_0$;
- $\sigma \models \varphi_1 \vee \varphi_2$ if $\sigma \models \varphi_1$ or $\sigma \models \varphi_2$;
- $\sigma \models \varphi_1 \wedge \varphi_2$ if $\sigma \models \varphi_1$ and $\sigma \models \varphi_2$;

- $\sigma \models \mathcal{X}\varphi$ if $\sigma^1 \models \varphi$;
- $\sigma \models \varphi_1 \mathcal{U} \varphi_2$ if $\exists k \geq 0$, s.t. $\sigma^k \models \varphi_2$, and $\forall i \in [0, k)$, $\sigma^i \models \varphi_1$;
- $\sigma \models \mathcal{F}\varphi$ if $\exists k \geq 0$, s.t. $\sigma^k \models \varphi$.
- $\sigma \models \mathcal{G}\varphi$ if $\forall k \geq 0$, $\sigma^k \models \varphi$.

An important property of syntactically co-safe LTL formulas is that, even though they have infinite-time semantics, finite words are sufficient to satisfy them. Similarly, finite words are sufficient to violate syntactically safe LTL formulas. Hence, we can capture desired robot behavior as a pair of co-safe and safe LTL specifications, where the co-safe component describes tasks for the robot to complete in finite time, and the safe component describes behaviors to avoid. We then say that a robot trajectory *satisfies* the pair of specifications if its corresponding word (event-driven trace) satisfies the co-safe component and does not violate the safe component. This combination of co-safe and safe LTL formulas allows us to describe many rich types of robotic tasks which can be realized in a finite time horizon in a safe manner (e.g., Specification 1 in Sec. I).

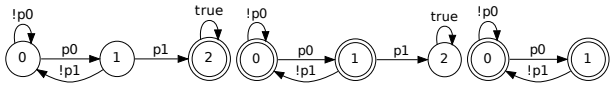
To evaluate robotic trajectories against LTL formulas, we use deterministic finite automata (DFA) [27]. A DFA is given by a tuple $(Z, \Sigma, \delta, z_0, F)$, where

- Z is a finite set of states,
- $\Sigma = 2^{\Pi}$ is the input alphabet, where each input letter is a truth assignment to the propositions in Π,
- $\delta : Z \times \Sigma \to Z$ is the transition function,
- $z_0 \in Z$ is the initial state, and
- $F \subseteq Z$ is the set of accepting states.

A *run* of a DFA $\mathcal{A}$ is a sequence of states $\gamma = \gamma_0 \gamma_1 \ldots \gamma_n$, where $\gamma_0 = z_0$ and $\gamma_i \in \mathcal{A}.Z$ for $i = 1, \ldots, n$. A run γ is called an *accepting run* if $\gamma_n \in \mathcal{A}.F$.

From a co-safe LTL formula φ_{cosafe}, a DFA $\mathcal{A}_{\varphi_{\text{cosafe}}}$ that accepts precisely all of the formula's satisfying finite words can be constructed [28]. Each input letter to $\mathcal{A}_{\varphi_{\text{cosafe}}}$ is a set $\tau \in 2^{\Pi}$ of propositions that are currently true in the system. To simplify notations, we refer to $\mathcal{A}_{\varphi_{\text{cosafe}}}$ as $\mathcal{A}_{\text{cosafe}}$. We assume $\mathcal{A}_{\text{cosafe}}$ is minimized, i.e., has the minimal number of states [29].

Similarly, from a safe LTL formula φ_{safe}, a DFA $\mathcal{A}_{\neg\varphi_{\text{safe}}}$ that accepts precisely all of the formula's violating finite words can be constructed [28]. To accept precisely all of the finite words that do not violate φ_{safe}, we flip the acceptance condition of this DFA to obtain $\neg\mathcal{A}_{\neg\varphi_{\text{safe}}}$,



(a) $\mathcal{A}_{\neg\varphi^1_{\text{safe}}}$ (b) $\neg\mathcal{A}_{\neg\varphi^1_{\text{safe}}}$ (c) $\mathcal{A}^1_{\text{safe}}$

Fig. 2: The conversion from the DFA corresponding to φ^1_{safe} to the minimal DFA $\mathcal{A}^1_{\text{safe}}$. The start state is 0.

which we minimize [27] and refer to simply as $\mathcal{A}_{\text{safe}}$. Specifically, given $\mathcal{A}_{\neg\varphi_{\text{safe}}} = (Z, \Sigma, \delta, z_0, F)$, we define

$$\mathcal{A}_{\text{safe}} = (Z, \Sigma, \delta, z_0, Z \setminus F)$$

and then minimize $\mathcal{A}_{\text{safe}}$. Note that all the accepting states of $\mathcal{A}_{\neg\varphi_{\text{safe}}}$ have only self-transitions [28]. Therefore, $\mathcal{A}_{\text{safe}}$ includes only accepting states after the minimization step. The DFA $\mathcal{A}_{\text{safe}}$ accepts a finite word $\bar{\sigma}$ if and only if there exists an infinite word extension to $\bar{\sigma}$ that satisfies φ_{safe}; that is, the language of $\mathcal{A}_{\text{safe}}$ is given by

$$\mathcal{L}\left(\mathcal{A}_{\text{safe}}\right) = \left\{\bar{\sigma} \in \Sigma^* \mid \exists \underline{\sigma} \in \Sigma^\omega \text{ such that } \bar{\sigma}.\underline{\sigma} \models \varphi_{\text{safe}}\right\}.$$

Here Σ^ω denotes the set of all infinite words over the alphabet Σ. Throughout this paper, we loosely say that a finite word $\bar{\sigma}$ "satisfies the safety formula φ_{safe}" to mean that $\bar{\sigma}$ has an infinite word extension that satisfies φ_{safe}, or equivalently, that $\bar{\sigma}$ does not violate φ_{safe}.

For example, consider the safety formula

$$\varphi^1_{\text{safe}} = \mathcal{G}(p_0 \rightarrow \mathcal{X}\neg p_1).$$

The words that includes the sequence of p_0p_1 violates φ^1_{safe}. The DFA that accepts all of the violating finite words of φ^1_{safe} is $\mathcal{A}_{\neg\varphi^1_{\text{safe}}}$ which is shown in Fig. 2a. By flipping the accepting condition, we obtain $\neg\mathcal{A}_{\neg\varphi^1_{\text{safe}}}$ in Fig. 2b which accepts all the finite words that do not include p_0p_1. In other words, $\neg\mathcal{A}_{\neg\varphi^1_{\text{safe}}}$ accepts all of the prefixes of the infinite words that satisfy φ^1_{safe}. We attain $\mathcal{A}^1_{\text{safe}}$ shown in Fig. 2c by minimizing $\neg\mathcal{A}_{\neg\varphi^1_{\text{safe}}}$. Note that all of the states of $\mathcal{A}^1_{\text{safe}}$ are accepting. This DFA generates only the words that do not include the sequence of p_0p_1, i.e., non-violating words.

III. Problem Description and Overall Approach

A. Problem Description

In this work, we consider a mobile robot with complex and possibly nonlinear hybrid dynamics moving in an environment to accomplish a high-level task. We assume that the environment consists of polytopic regions of interests, each of which holds a set of properties (propositions). Let Π denote the set of all environmental propositions. The robot task is expressed as a temporal logic statement φ defined over Π in the form of a conjunction of syntactically co-safe and safe LTL formulas:

$$\varphi = \varphi_{\text{cosafe}} \wedge \varphi_{\text{safe}}. \tag{1}$$

An example of such a task is Specification 1 in Sec. I. Setting $\Pi = \{p_0, \ldots, p_5\}$, the co-safe component of this specification becomes "first visit p_0, p_1, and p_2 in any order and then visit p_4 and p_3 in that order," and the safe component becomes "always avoid p_5 and never go to p_2 if the previously visited propositional region is p_0." These statements can be translated to the following LTL formulas:

$$\begin{aligned} \varphi_{\text{cosafe}} &= (\neg(p_3 \vee p_4)\,\mathcal{U} p_0) \wedge (\neg(p_3 \vee p_4)\,\mathcal{U} p_1) \wedge \\ &\quad (\neg(p_3 \vee p_4)\,\mathcal{U} p_2) \wedge (\neg p_3 \,\mathcal{U}(p_4 \wedge \mathcal{X}\mathcal{F} p_3)) \qquad (2) \\ \varphi_{\text{safe}} &= \mathcal{G}\neg p_5 \wedge \mathcal{G}(p_0 \rightarrow \mathcal{X}\mathcal{X}\neg p_2). \qquad (3) \end{aligned}$$

We are interested in finding a motion plan for the robot to achieve φ.

We assume that the motion of the robot in the environment is described by hybrid system H defined in Sec. II-A. An example of such a system is a robot that is capable of switching between different modes of operation (e.g., vacuuming and mopping) in an office environment. Let the first $n_e \in \mathbb{N}^+$ components of the continuous states of the hybrid system in mode q refer to the position of the robot in a n_e-dimensional environment. Also, let $\textsc{Proj}(A)$ denote projection of a Euclidean set A onto $\mathbb{R}^{n_e}$. We define the workspace of the robot as $W = \{(q, W_q) : W_q = \textsc{Proj}(X_q), q \in Q\}$. In words, the first n_e elements of the continuous component of a hybrid state in each mode correspond to a point in the workspace of the robot (in this study we assume $n_e = 2$). We establish the relationship between the hybrid state of the robot $s = (q, x)$ and its workspace by function $h_H : S \rightarrow W$. Similarly, given $w \in W$, we define $h_H^{-1}(w) = \{s \in S : h_H(s) = w\}$.

Thus, all the environmental propositions Π can be assigned to the states of the hybrid system according to their location in the workspace. This means that achieving φ in the environment is equivalent to satisfying φ in the robot's hybrid state space. Therefore, the challenge is to design a motion plan for H that satisfies φ.

Furthermore, we assume that while the robot has full information of the propositional regions and their locations in the environment, it has only partial *a priori* knowledge of the obstacles of the environment. Thus, for each mode q of the hybrid system, $W_{q,obs} = W^k_{q,obs} \cup W^u_{q,obs}$ where $W^k_{q,obs}$ and $W^u_{q,obs}$ refer to the sets of known and unknown obstacles, respectively. We also assume that the robot can detect an unknown obstacle when it comes within some proximity of it. This is represented by SENSE in the hybrid model H in Sec. II-A. In practice, SENSE can be viewed as a range sensor with a fixed radius ρ, as is commonly seen in related work [30].

Lastly, we assume that specification φ is satisfiable in the initial map. Due to possible unknown obstacles in the environment, however, the satisfaction of φ cannot be guaranteed. At the same time, we do not want the robot to abort the mission if it realizes that fragments of the specification cannot be met. Instead, we require the robot to satisfy the co-safe component (φ_{cosafe}) of the specification as closely as possible while never violating safety (φ_{safe}). We envision many scenarios where this can be an advantageous approach (e.g., the janitor robot example). Note that formulas in the form of (1) constitute a fragment of general LTL, and the decomposition of the specification into safety and co-safety may not be unique. It is up to the user to define which requirements to be safety (φ_{safe} - hard constraints) and which ones to be co-safety (φ_{cosafe}-soft constraints). We define and discuss the definition of satisfying a specification as closely as possible below and in Sec. V. We now focus on the following problem.

PROBLEM: Given a robot hybrid model H with a partially unknown workspace W and a task specification φ expressed as a conjunction of a syntactically co-safe LTL formula φ_{cosafe} and a syntactically safe LTL formula φ_{safe} over Π, find a motion plan that does not violate φ_{safe} and satisfies φ_{cosafe} as *closely* as possible.

B. Overall Approach

We employ a multi-layered synergistic framework [14]–[16], [25], [26] to solve the above motion planning problem by using the initial knowledge of the workspace. The framework

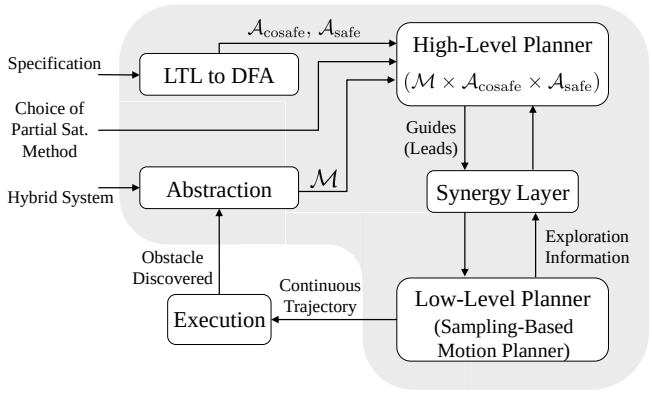


Fig. 3: Multi-layered synergistic motion-planning framework.

consists of three main layers: a high-level search layer, a low-level search layer, and a synergy layer that facilitates the interaction between the high-level and the low-level search layers (see Fig. 3). The high-level planner uses an abstraction of the hybrid system [15] and the specification φ to suggest high level plans. The low-level planner uses the dynamics of the hybrid system and the suggested high-level plans to explore the state space for feasible solutions. In our work, the low-level layer is a sampling-based planner and does not assume the existence of a controller [8].

To satisfy a specification in a partially undiscovered environment, an iterative high-level planner is employed. Every time an unknown obstacle is encountered, the high-level planner modifies the coarse high-level plan online by accounting for the geometry of the discovered obstacle, the path traveled to that point, and the remaining segment of the specification that is yet to be satisfied. This replanning is achieved in three steps. (1) A "braking" operation is applied to prevent the robot from colliding with the newly discovered obstacle. We assume that the robot's sensing radius is sufficiently large to ensure that the robot does not collide with the obstacle (the braking operation can be avoided if a contingency maneuver is used - see below). (2) The abstraction model is "patched" to reflect the new changes to the environment. All feasibility estimates and data collected by the synergistic framework are preserved for the unaffected portion of the abstraction. (3) The path traveled by the robot so far is mapped onto the modified abstraction model, and a new satisfying plan is generated as a continuation of the explored portion of the old plan. Thus, the robot does not need to return to its starting point

every time it encounters an unknown environmental feature. Moreover, the robot's progress in satisfying the specification is preserved. A high-level diagram of this iterative motion-planning framework appears in Fig. 3.

We note that, in general, a contingency maneuver can be used instead of a "braking" operation when an unknown obstacle is discovered. For the simplicity of the presentation, we assume that the robot is capable of stopping, but by no means our framework is limited to this maneuver. The exploration for the "best" contingency plan is not the focus of this paper, and we refer the reader to [30], [31] for the description of general contingency plans.

To design a satisfying high-level plan, we use the automata representing the specification. Recall that from φ in (1), a pair of DFAs, $\mathcal{A}_{\text{cosafe}}$ and $\mathcal{A}_{\text{safe}}$, can be constructed that accept all of the satisfying finite words for φ_{cosafe} and φ_{safe}, respectively (Sec. II-B) [25], [28]. In cases that φ_{cosafe} is found unsatisfiable, we consider the intended restrictions on the specification to define a suitable partial satisfaction metric. For instance, consider the co-safe component of Specification 1 above and Specifications 2 and 3 below for the janitor robot example shown in Fig. 1.

Specification 2: Water the plans and pick the supply cabinet key in region p_0, turn off the coffee machine at p_1, and clean the blackboard in p_2 in any order. Then, go to p_4 to pick up a duster from the supply cabinet before visiting p_3 to dust and organize the desk.

Specification 3: Pick up trash from the bins in regions p_0, p_1, and p_2 in any order, and then take the collected trash to the chute in p_4. Next, organize the desk in p_3.

Note that all three specifications have the same requirements with respect to Π (see Sec. III-A) and translate to the same co-safe LTL formula (φ_{cosafe} in (2)), but the importance of the regional visits clearly varies when it comes to partial satisfaction. Given that regions p_0 and p_4 are unaccessible due to the closed doors, it is still desired for the robot to turn off the coffee machine in p_1 and clean the blackboard in p_2 in Specification 1. However, visiting p_3, even though accessible, is pointless since the robot cannot dust the desk without having picked up the duster from p_4. In Specification 2, though, it is desired for the robot to visit p_3 in addition to p_1 and p_2 since organizing the desk can be performed without the duster. On the contrary, only organizing the desk in p_3 is the obvious thing to do in Specification 3 since there is no need to collect trash when the trash chute is blocked.

To address each of the above scenarios, we introduce three methods of partial satisfaction.

These methods are developed for specifications with the form in (1) and utilize $\mathcal{A}_{\text{cosafe}}$ to measure the "distance-to-satisfaction" of the specification. These measures are used to produce a high-level plan that completely satisfies φ_{safe} (i.e., does not violate φ_{safe}) and satisfies φ_{cosafe} as closely as possible. The definition of these metrics are described in Sec. V.

It is important to note that our method of generating a new high-level plan is fast. This is for the following two reasons: (1) we are not recomputing the two DFAs, which do not need to change since the specification does not change following the discovery of an obstacle, and (2) we generate the abstraction of the hybrid system by decomposing the workspace through triangulation, which has been shown to be computationally inexpensive [16] (in fact we verify in our experimental results in Sec. VII). Our method to *locally patch* the changed region of the workspace essentially boils down to a retriangulation of the smaller portion. For instance, the computation time for regenerating the abstraction for the janitor robot example is on the order of a hundredth of a second on a modern PC.

IV. Planning Framework

In this section, we describe our iterative planning framework. This planner is detailed in Alg. IV.1. It builds on top of the motion-planning framework developed in [14]–[16], [23]. Sec. IV-A, IV-B, and IV-C present modifications to the previous works to allow for hybrid dynamics and to enable planning with safety in addition to co-safety specifications. Sec. IV-D and V detail the replanning and partial satisfaction methods that are designed to handle environment uncertainties.

A. Abstraction

To produce a high-level plan, we first abstract the hybrid system H to a discrete model $\mathcal{M} = (D, d_0, \rightarrow_D, \Pi, L_D)$, where D is a set of discrete states, $d_0 \in D$ is the initial state, $\rightarrow_D \subseteq D \times D$ is the transition relation, and $L_D : D \rightarrow 2^{\Pi}$ is a labeling function. We refer to the model $\mathcal{M}$ as the *abstraction* of the hybrid system. Our method of constructing $\mathcal{M}$ is adapted from the ones detailed in [15], [16] and modified for hybrid systems.

To construct $\mathcal{M}$, we first partition each workspace W_q (for each discrete mode q) into a set of *regions* (i.e., $W_q = \bigcup_{i=1}^{N_r^q} r_i^q$). Specifically, we use a geometry-based conforming Delaunay triangulation of W_q that respects the propositional regions and the boundaries of the obstacles.

Recall from Sec. III that $W_q = \textsc{Proj}(X_q)$, where X_q is the domain of the continuous states of the hybrid system in mode q. Thus, the decomposition of W_q induces a partition in the hybrid state space. Let S_i^q denote the set of all hybrid states that correspond to the region r_i^q in W_q (i.e., $S_i^q = \{(q,x) \in S \mid x \in \textsc{Proj}^{-1}(r_i^q),\, r_i^q \subseteq W_q,\, q \in Q\}$). Then,

$$S = \bigcup_{q \in Q} \bigcup_{i=1}^{N_r^q} S_i^q.$$

We associate a unique discrete state $d \in D$ to each S_i^q. We model this correspondence with a family of maps $\{\Upsilon_q : S^q \to D \mid S^q = q \times X_q,\, q \in Q\}$; then the previous sentence can be written as $\Upsilon_q(S_i^q) = d$. Moreover, D can be written as $D = \cup_{q \in Q}\{\Upsilon_q(S_i^q) \mid 1 \leq i \leq N_r^q\}$.

We construct the transition relation $\to_D$ to include geometric adjacencies between regions of a given workspace as well as adjacencies between discrete modes enabled by GUARD functions of the hybrid system. Specifically, for each pair of geometrically adjacent regions r_j^q and r_k^q in W_q, we add the transition $\big(\Upsilon_q(S_j^q), \Upsilon_q(S_k^q)\big)$ to $\to_D$; furthermore for each pair of sets S_l^q and $S_m^{q'}$ between which a discrete jump is possible, we add the transition $(\Upsilon_q(S_l^q), \Upsilon_q(S_m^{q'}))$ to $\to_D$. All the hybrid states in S_i^q have the same labels since the triangulation of each workspace W_q respects the propositional regions. Hence, the labeling function L_D corresponds to the labeling function L from the definition of H; that is, $L_D(\Upsilon_q(S_i^q)) = L(s)$ for every $s \in S_i^q$. For further details, we refer the reader to our previous work [15].

It should be noted that the initial construction of $\mathcal{M}$ is based on the initial knowledge of the environment map. As the robot discovers unknown obstacles, the map is updated and $\mathcal{M}$ is patched to reflect the new workspace information. Given that this method is based on a triangulation of a two-dimensional space, patching the abstraction is fast. Furthermore, we initially assume transitions between all adjacent partitions of the workspace are realizable even though the dynamics of the robot may prevent some transitions. This does not create a problem in our planning framework because the synergistic framework biases its discrete search against unrealizable transitions. In fact, one of the advantages of our planning framework is that it does not require a bisimilar abstraction and therefore allows for inexpensive and fast construction of an approximate abstraction model.

B. Initializing the Product Automaton

The structure we use to guide the motion tree of system trajectories is a product automaton, which is computed as

$$\mathcal{P} = \mathcal{M} \times \mathcal{A}_{\text{cosafe}} \times \mathcal{A}_{\text{safe}}.$$

In lines 2-3 of Alg. IV.1, we compute the minimal DFAs $\mathcal{A}_{\text{cosafe}}$ and $\mathcal{A}_{\text{safe}}$ corresponding to the formulas φ_{cosafe} and φ_{safe}, respectively, as defined in Sec. II-B [28], [32]. Though each translation can require time doubly exponential with respect to the number of propositions in the formula, we only compute each DFA once, and so the translations can be seen as an offline step. Then, in line 4 of Alg. IV.1, we compute the product automaton $\mathcal{P}$. We refer to elements of the product automaton $\mathcal{P}$ as *high-level states*. The product automaton $\mathcal{P}$ is a directed graph in which there exists an edge from high-level state (d_i, z_i^c, z_i^s) to (d_j, z_j^c, z_j^s) if and only if

- d_i and d_j are adjacent in $\mathcal{M}$,
- $\mathcal{A}_{\text{cosafe}}.\delta\left(z_i^c, \mathcal{M}.L_D(d_j)\right) = z_j^c$,
- $\mathcal{A}_{\text{safe}}.\delta\left(z_i^s, \mathcal{M}.L_D(d_j)\right) = z_j^s$,

where $\mathcal{A}_{\text{cosafe}}.\delta$ and $\mathcal{A}_{\text{safe}}.\delta$ are the deterministic transition functions for $\mathcal{A}_{\text{cosafe}}$ and $\mathcal{A}_{\text{safe}}$, respectively. We call a high-level state $(d, z^c, z^s) \in \mathcal{P}$ an *accepting* state (or a *goal* state) if z^c is an accepting state in $\mathcal{A}_{\text{cosafe}}$ and z^s is an accepting state in $\mathcal{A}_{\text{safe}}$. Since all z^ss are accepting (see Sec. II-B), the high-level state (d, z^c, z^s) is accepting if z^c is accepting. As a result of this construction, all of the paths on $\mathcal{P}$ are non-violating with respect to φ_{safe}, and the paths that end in an accepting state of $\mathcal{P}$ are φ_{cosafe}-satisfying.

For each high-level state $(d, z^c, z^s) \in \mathcal{P}$, the synergy layer assigns a weight defined by

$$w(d, z^c, z^s) = \frac{\left(\textsc{cov}(d, z^c, z^s) + 1\right) \cdot \textsc{vol}(d)}{\textsc{DistToAcc}(z^c) \cdot \left(\textsc{numsel}(d, z^c, z^s) + 1\right)^2}, \tag{4}$$

where $\textsc{cov}(d, z^c, z^s)$ is the number of tree vertices associated with (d, z^c, z^s) (an estimate of coverage by the low-level planner), $\textsc{vol}(d)$ is the area of the workspace corresponding to the abstraction state d, and $\textsc{numsel}(d, z^c, z^s)$ is the number of times (d, z^c, z^s) has been selected for tree expansion in line 2 of Alg. IV.3. $\textsc{DistToAcc}(z^c)$ is the minimum distance from automaton state z^c to an accepting state in $\mathcal{A}_{\text{cosafe}}$.

Finally, to each directed edge $e = (h_i, h_j)$ between high-level states $h_i, h_j \in \mathcal{P}$, we assign the weight

$$w(e) = \frac{1}{w(h_i) \cdot w(h_j)}. \tag{5}$$

Algorithm IV.1 Framework for planning for a robotic system with an LTL specification in a partially unknown environment

Input: Robot model $H = (S, s_0, \text{INV}, \text{SENSE}, E, \text{GUARD}, \text{JUMP}, U, \text{FLOW}, \Pi, L)$, a bounded workspace $W \subset \mathbb{R}^2$, a set of initially known obstacles $O \subset W$, LTL formula $\varphi = \varphi_{\text{cosafe}} \wedge \varphi_{\text{safe}}$, a method of partial satisfaction MPS, and a time bound $t_{\max}$.

Output: Returns `true` if successful in moving the robot through the workspace to satisfy φ_{safe} and satisfy φ_{cosafe} as closely as possible; returns `false` otherwise.

1: $\mathcal{M} \leftarrow \text{COMPUTEABSTRACTION}(W, O, H.\Pi, H.L)$
2: $\mathcal{A}_{\text{cosafe}} \leftarrow \text{COMPUTEMINDFA}(\varphi_{\text{cosafe}}, W, H.L)$
3: $\mathcal{A}_{\text{safe}} \leftarrow \text{COMPUTEMINDFA}(\varphi_{\text{safe}}, W, H.L)$
4: $\mathcal{P} \leftarrow \text{COMPUTEPRODUCT}(\mathcal{M}, \mathcal{A}_{\text{cosafe}}, \mathcal{A}_{\text{safe}}, H.\Pi, H.L)$
5: $\{s_i\}_{i\geq 0} \leftarrow \text{PLAN}(H, W, O, \mathcal{P}, t_{\max}, \text{MPS})$
6: $t_{\text{plan}} \leftarrow$ time spent by PLAN in line 5
7: $t_{\max} \leftarrow t_{\max} - t_{\text{plan}}$
8: $j \leftarrow 1$
9: **while** $j < |\{s_i\}|$ **do**
10: Move robot from state s_{j-1} to state s_j
11: **if** $H.\text{SENSE}(s_j.x) = \top$ **then**
12: Apply braking operation to reach stopped robot state s'
13: $H.s_0 \leftarrow s'$
14: Add discovered obstacle o_{new} to O
15: $\mathcal{P} \leftarrow \text{PATCHPRODUCT}(\mathcal{P}, H.S, W, o_{\text{new}}, H.\Pi, H.L)$
16: $\{s_i\}_{i\geq 0} \leftarrow \text{PLAN}(H, W, O, H.\Pi, H.L, \mathcal{P}, t_{\max})$
17: **if** PLAN was unsuccessful **then**
18: **return** `false`
19: $t_{\text{plan}} \leftarrow$ time spent by PLAN in line 16
20: $t_{\max} \leftarrow t_{\max} - t_{\text{plan}}$
21: $j \leftarrow 1$
22: $j \leftarrow j + 1$
23: **return** `true`

The estimates in (4) and (5) have been shown to work well in previous work [14]. In general, a weighing scheme that incorporates more than just number-of-edge distance is useful to promote expansion in unexplored areas (i.e., where COV and NUMSEL are both small) and to discourage

expansion in areas where attempts at exploration have repeatedly failed (i.e., where NUMSEL $\gg$ COV).

C. Planning

Once the product automaton has been computed, line 5 of Alg. IV.1 computes a trajectory for the system that completely satisfies the safe formula φ_{safe} and satisfies the co-safe formula φ_{cosafe} as "closely as possible". The details of this approach are given in Alg. IV.2. The core loop of the planning algorithm is shown in lines 3, 4, and 5 of Alg. IV.2.

The subroutine COMPUTELEAD generates a lead, which is a path of high-level states over $\mathcal{P}$ that satisfies φ_{cosafe} as closely as possible, fully satisfies φ_{safe}, and minimizes the sum of the edge weights defined in (5). When φ is fully satisfiable, the returned lead is a path that ends in an accepting state of $\mathcal{P}$. If such a path does not exist (i.e., an accepting state is not reachable), φ becomes unsatisfiable. This situation takes place when the discovered obstacle blocks all of the satisfying trajectories of the system. We introduce three methods of generating leads to partially satisfy φ in such situations. These methods of partial satisfaction are described in Sec. V. Once a lead is obtained, it is then suggested to EXPLORE which attempts to guide the motion tree in the direction of the lead.

1) Guiding The Low-Level Tree Planner: The subroutine COMPUTEAVAILABLECELLS in line 4 of Alg. IV.2 creates a set of high-level states from the current lead that are nonempty (i.e., there exist vertices in the tree of motions that are annotated with these high-level states). To promote progress, we favor high-level states that are closest to the accepting state of the lead. Specifically, moving backwards along the lead, for each nonempty high-level state (d, z^c, z^s) we encounter, we add (d, z^c, z^s) to the set C of available high-level states and then quit early with probability 0.5. By quitting the process early with probability 0.5, we are biasing expansion toward the areas of the tree that have made the most progress along the lead, and therefore have made the most progress completing the task specification.

The subroutine EXPLORE, given in Alg. IV.3, corresponds to the low-level search layer of the framework. This function promotes tree expansion in high-level states from the set C. In line 2 of EXPLORE, a high-level state (d, z^c, z^s) is sampled with probability

$$\frac{w(d, z^c, z^s)}{\sum_{(d', z^{c\prime}, z^{s\prime}) \in C} w(d', z^{c\prime}, z^{s\prime})}.$$

Algorithm IV.2 PLAN: Temporal planning algorithm

Input: Robot model $H = (S, s_0, \text{INV}, \text{SENSE}, E, \text{GUARD}, \text{JUMP}, U, \text{FLOW}, \Pi, L)$, a bounded workspace $W \subset \mathbb{R}^2$, a set of initially known obstacles $O \subset W$, a product automaton $\mathcal{P}$, a time bound t_{max}, and a method of partial satisfaction MPS.

Output: Returns a sequence of triplets, each containing a robot system state, control, and corresponding high-level state, representing a system trajectory that satisfies the specification. Reports an error and aborts if no such trajectory could be found within time t_{max}.

1: $\mathcal{T} \leftarrow \text{INITIALIZETREE}(s_0)$
2: **while** TIME ELAPSED $< t_{\text{max}}$ **do**
3: $K \leftarrow \text{COMPUTELEAD}^{\text{MPS}}$
4: $C \leftarrow \text{COMPUTEAVAILABLECELLS}(K)$
5: $(v, \mathcal{P}) \leftarrow \text{EXPLORE}(H, W, O, \mathcal{T}, C, K, \mathcal{P}, \Delta t)$
6: **if** $v \neq$ `NULL` **then**
7: Follow v`.parent` to construct trajectory $\{s_i\}_i$
8: **return** $\{s_i\}_i$
9: Report unsuccessful and exit

Then, in line 3, a low-level tree planner attempts to create a new tree vertex corresponding to both a robot state s that maps to abstraction state d and a trajectory from the tree root that maps to automaton states z^c and z^s in $\mathcal{A}_{\text{cosafe}}$ and $\mathcal{A}_{\text{safe}}$, respectively. To do this, the planner first extends s by a trajectory segment ξ through sampling a control and using H.FLOW, H.GUARD, and H.JUMP as explained in Sec. II-A. Then, the validity of ξ is checked by using H.INV and H.L. Any tree-based motion planner can be used in this step. Our implementation uses an `EST`-like approach [5]. We note that the trajectories of hybrid systems are not always computationally nice objects; therefore, we employ numerical methods for the generation of ξ.

If z^c is an accepting state in $\mathcal{A}_{\text{cosafe}}$, then v is returned as the endpoint of a solution trajectory, which is constructed by line 7 of PLAN (z^c can just be as close as possible to an accepting state if φ_{cosafe} is unsatisfiable). Otherwise, if the new vertex v corresponds to a newly reached high-level state that is in the current lead, then the high-level state is added to the set of available cells in line 8 of EXPLORE to be considered in future iterations. In line 9, the weights of the high-level states in C are updated according to (4) with the new exploration information. These new weights are considered in the computation of a new lead in PLAN. We make no attempt in

Algorithm IV.3 EXPLORE: Tree-exploration subroutine

Input: Robot model $H = (S, s_0, \text{INV}, \text{SENSE}, E, \text{GUARD}, \text{JUMP}, U, \text{FLOW}, \Pi, L)$, a bounded workspace $W \subset \mathbb{R}^2$, a set of known obstacles $O \subset W$, a tree of motions $\mathcal{T}$, a set of available high-level states C, a lead K, a product automaton $\mathcal{P}$, and an exploration time Δt.

Output: Returns a tree vertex that reaches a goal high-level state if such a vertex was found; returns `NULL` otherwise. Also updates the weights of the high-level states of $\mathcal{P}$.

while TIME ELAPSED(Δt) **do**
  $(d, z^c, z^s) \leftarrow C$`.sample()`
  $v \leftarrow$ SELECTANDEXTEND($\mathcal{T}, H, (d, z^c, z^s), O, \mathcal{P}$)
  if $v.z^c \neq \varnothing$ and $v.z^s \neq \varnothing$ **then**
    if $v.z^c$`.isAccepting()` and $v.z^s$`.isAccepting()` **then**
      return $(v, \mathcal{P})$
    if $(v.d, v.z^c, v.z^s) \in K \setminus C$ **then**
      $C \leftarrow C \cup \{(v.d, v.z^c, v.z^s)\}$
  Update $\mathcal{P}$ with the new weights of the states in C according to (4).
return $(\texttt{NULL}, \mathcal{P})$

PLAN to smooth or shorten the continuous solution trajectory. Shortening a trajectory to satisfy both differential constraints and a logical specification remains a topic of future work.

D. Discovering an Obstacle and Replanning

Once a system trajectory that satisfies φ is computed, we begin moving the robot along the trajectory. At each state in the trajectory, we query the robot's range sensor in line 11 of Alg. IV.1. We assume that the robot's range sensor checks for obstacles within radius ρ of the center of the robot and reports a polygonal model of any previously unknown obstacle that it finds. If no new obstacles are discovered along the trajectory, then the robot reaches the final state of the planned trajectory and stops, having completed its mission. If an obstacle is discovered by the range sensor from some state s along the trajectory, then we apply a braking operation to the robot to reach some stopped state s' in line 12 of Alg. IV.1. The braking operation respects the dynamics of the system. In the general case, the robot should perform a contingency maneuver to avoid the newly discovered obstacle [31], [33]. The radius ρ of the range sensor is assumed to be large enough for the braking or contingency maneuver to be safely performed. Once the braking maneuver is complete, we patch the portion of the discrete abstraction $\mathcal{M}$ that intersects the new obstacle

Algorithm IV.4 PATCHPRODUCT: Subroutine to locally patch product automaton given a newly discovered obstacle

Input: A product automaton $\mathcal{P}$, robot model H, workspace W, a newly discovered obstacle o_{new}.

Output: Returns a patched version of $\mathcal{P}$ that respects the newly discovered obstacle.

1: $R \leftarrow$ GETINTERSECTINGREGIONS$(\mathcal{P}.\mathcal{M}, o_{\text{new}})$
2: $(V, E) \leftarrow$ COMPUTEBOUNDARY(R)
3: $N \leftarrow$ DECOMPOSEPORTION$(V, E, H.S, H.\Pi, H.L)$
4: $\mathcal{P} \leftarrow$ PATCHABSTRACTION$(\mathcal{P}, R, N)$
5: **return** $\mathcal{P}$

by calling the subroutine PATCHPRODUCT (Alg. IV.4), and we obtain an updated instance of $\mathcal{M}$ that respects all known obstacles. After patching the discrete abstraction, PATCHPRODUCT patches the corresponding elements of $\mathcal{P}$. The PATCHPRODUCT routine operates in four steps. (1) Compute R, the set of triangles in the workspace that intersect with the newly discovered obstacle. (2) Compute the exterior boundary of the set R as a planar straight-line graph (V, E). (3) Compute a new triangulation N of the section of the workspace enclosed by (V, E). (4) Insert the new triangulation N into the abstraction $\mathcal{M}$ and propagate the changes to the product automaton $\mathcal{P}$.

After the product automaton has been patched, we replan a trajectory from s' in line 16 of Alg. IV.1, following the same planning approach described in Sec. IV-C. Once a new trajectory is found by the planner, we resume moving the robot from s' along the new trajectory. It is important to note that only the high-level states of $\mathcal{P}$ that intersect with the new obstacle are replaced, and their incident edge weights are lost and recomputed in the next planning iteration. All other high-level states and edge weights in $\mathcal{P}$ are retained.

V. METHODS FOR PARTIAL SATISFACTION

After the discovery of an unknown obstacle, φ becomes unsatisfiable if no accepting state is reachable in the newly patched $\mathcal{P}$. This is generally due to two reasons: (1) the discovered obstacle blocks all the possible paths to at least one of the propositional regions required by the task (e.g., closed office doors in the above janitor robot example), and (2) even though the

discovered obstacle does not completely block a propositional region, the dynamics of the robot do not allow a continuous trajectory to the propositional region. In this paper, we focus on the former case, which is detectable by a decomposition of the workspace. In the latter case, the feasibility of satisfying the specification can be estimated by using the weights defined in (4) and (5). One can declare the specification unsatisfiable due to dynamics of the robotic system with a high probability by imposing a threshold on these weights. The question of how to determine a reasonable threshold is interesting and left for future work.

In this section, we introduce three methods to generate leads that partially satisfy φ_{cosafe} and fully satisfy φ_{safe} once φ is determined to be unsatisfiable. Each method results in a unique robot behavior, and it is up to the user to choose the method that reflects the most desired outcome. The first method renders a robot behavior that is "conservative" with respect to the specification. In that, the robot only executes the portion of the specification that is satisfiable without violating the specified temporal ordering of the tasks (e.g., this method is suitable for Specification 1). The second method, "aggressive," is designed for the type of specifications that, if not fully satisfiable, only the execution of the last segment of the ordered tasks is preferred. Specification 3 is an example of such a specification type. The last method, called "moderate," results in a robot behavior that performs all the feasible tasks while respecting their temporal ordering. For instance, this method is suitable for Specification 2.

A. *Conservative Method*

Algorithm V.1 COMPUTELEAD$^{\text{CON}}$: Subroutine to compute conservative high-level guides

Input: A product automaton $\mathcal{P}$ and a starting high-level state $(d_0, z_0^c, z_0^s) \in \mathcal{P}$.

Output: Returns a lead, which is a sequence of high-level states beginning with the given start (d_0, z_0^c, z_0^s) and ending with a state that is accepting in $\mathcal{A}_{\text{safe}}$ and a minimal distance to an accepting state in $\mathcal{A}_{\text{cosafe}}$.

1: $S \leftarrow \{(d, z^c, z^s) \in \mathcal{P} \mid (d, z^c, z^s)$ is reachable from $(d_0, z_0^c, z_0^s)\}$

2: $F \leftarrow \arg\min_{(d,z^c,z^s)\in S} (\text{DISTTOACC}(z^c))$

3: Run Dijkstra's all-pairs shortest-path algorithm on $\mathcal{P}$ with source (d_0, z_0^c, z_0^s); store parent map `parent` and weight map `weight`

4: $(d_g, z_g^c, z_g^s) \leftarrow \arg\min_{(d,z^c,z^s)\in F}\{\texttt{weight}[(d, z^c, z^s)]\}$

5: Construct lead $K = ((d_0, z_0^c, z_0^s), \ldots, (d_g, z_g^c, z_g^s))$ using `parent` map

6: **return** K

We define a measure of satisfiability that uses the graph-based distance to an accepting state in the DFA $\mathcal{A}_{\text{cosafe}}$. Let DISTTOACC$(z^c)$ be the number of edges in the minimal length path in $\mathcal{A}_{\text{cosafe}}$ from the state z^c to an accepting state. Each high-level state (d, z^c, z^s) is labeled with DISTTOACC(z^c) corresponding to the automaton state $z^c \in \mathcal{A}_{\text{cosafe}}.Z$. In this method, we design COMPUTELEAD$^{\text{CON}}$ to compute paths that end in a high-level state (d_g, z^c_g, z^s_g) such that DISTTOACC(z^c_g) is minimum among all reachable states. The pseudocode of this algorithm is shown in Alg. V.1. In many cases, there are multiple candidate high-level states that tie under the DISTTOACC definition over the co-safe automaton states. To break ties, we choose the high-level state with minimal edge-weight distance from the starting high-level state, using the edge-weight function defined in (5).

The function DISTTOACC is an intuitive metric on the co-safe automaton. This measure, particularly, translates to a unique meaning for the cases that the propositional regions in the environment are not intersecting. In such cases, function DISTTOACC is the measure of the smallest number of remaining propositions to visit to fully satisfy φ_{cosafe}. Formally, let $z^c \in \mathcal{A}_{\text{cosafe}}.Z$ be the state that is reached by the input word $u \in (2^{\Pi})^*$ in $\mathcal{A}_{\text{cosafe}}$. The function DISTTOACC$(z^c)$ returns the length of the shortest suffix v that is needed to extend u to an accepting word of $\mathcal{A}_{\text{cosafe}}$, i.e., $u.v = \bar{\sigma} \in \mathcal{L}(\mathcal{A}_{\text{cosafe}})$. Therefore, the output word of a trajectory of $\mathcal{P}$ from the initial state to (d_g, z^c_g, z^s_g) that minimizes DISTTOACC(z^c_g) is a prefix of an accepting word of $\mathcal{A}_{\text{cosafe}}$, whose shortest suffix has minimum length among all other suffixes of the achievable prefixes given the safety and environment constraints. Note that this trajectory visits as many propositions as possible without violating φ_{cosafe}. Furthermore, if φ_{cosafe} is satisfiable in the environment, DISTTOACC$(z^c_g) = 0$, i.e., (d_g, z^c_g, z^s_g) is an accepting state.

In each call of the COMPUTELEAD$^{\text{CON}}$ (Alg. V.1), the returned lead is optimal in the following sense. Let $\bar{u}$ be the word that the robot achieves in the environment before discovering an unknown obstacle. Let $\underline{u}$ be the output word of the lead that is computed by COMPUTELEAD$^{\text{CON}}$ (Alg. V.1) at this point. Then, $\bar{u}.\underline{u}$ is a prefix of an accepting word $\bar{\sigma} = \bar{u}.\underline{u}.v$ of $\mathcal{A}_{\text{cosafe}}$, where v has the minimum length given the environment and safety constraints. This optimality property is valid for every call of the algorithm (replanning instance). For the overall robot word, which is achieved by multiple calls of the algorithm, however, no optimal guarantees can be given. Nevertheless, we can state the following lemma for the overall word.

Lemma 1: Let $\gamma_{\mathcal{P}} = (d_0, z^c_0, z^s_0) \cdots (d_n, z^c_n, z^s_n)$ be the overall high-level path achieved by

using the conservative method's COMPUTELEAD$^{\text{CON}}$, and let u denote the obtained word from $\gamma_{\mathcal{P}}$. Then, (1) u is a prefix of a word in $\mathcal{L}(\mathcal{A}_{\text{cosafe}})$, and (2) u is in $\mathcal{L}(\mathcal{A}_{\text{safe}})$.

Proof: (1) The word u is achieved by the run $\gamma_{\mathcal{A}_{\text{cosafe}}} = z_0^c \cdots z_n^c$ in $\mathcal{A}_{\text{cosafe}}$, where z_0^c is the initial state. As φ_{cosafe} is valid (logically satisfiable), we have that $0 \leq \text{DISTTOACC}(z_n^c) < \infty$, which means that $\gamma_{\mathcal{A}_{\text{cosafe}}}$ can be extended to an accepting run of $\mathcal{A}_{\text{cosafe}}$. Therefore, u is a prefix of a word in $\mathcal{L}(\mathcal{A}_{\text{cosafe}})$. (2) The word u is achieved by the run $\gamma_{\mathcal{A}_{\text{safe}}} = z_0^s \cdots z_n^s$ in $\mathcal{A}_{\text{safe}}$, where z_0^s is the initial state. Since all the states in $\mathcal{A}_{\text{safe}}$ are accepting (see Sec. II), $\gamma_{\mathcal{A}_{\text{safe}}}$ is accepting. Therefore, $u \in \mathcal{L}(\mathcal{A}_{\text{safe}})$. □

The plans produced by the conservative method are best for the kind of specifications in which the specified order of the tasks are strict. For Specification 1, for example, this method can generate a trajectory that takes the robot to the regions p_1 and p_2 after discovery of the closed door of the room containing p_0. Given that p_0 is unreachable, this trajectory does not extend to any other propositional regions because it is a violation of the specified order.

The conservative method, although intuitive, has its own limitations. There are many specifications in which minimizing DISTTOACC does not necessarily yield the most enticing partially satisfying plans. Specification 2, in which the specified order of tasks is not strict but rather a preference, is an example of such specifications. Specifically, for any mission in which the tasks are favored, but not necessarily enforced, to be performed in a particular order, the conservative method does not necessarily generate the most desirable plans. In these situations, a more reasonable behavior for the robot is to "skip" those tasks that cannot be completed and continue with the remaining tasks in the same designated order. In the next two sections, we present algorithms that emit such plans.

B. Aggressive Method

Motivated by Specification 3, we introduce a method of computing a lead that partially satisfies φ_{cosafe} by planning for the *achievable* tasks that appear at the bottom of the ordered list of tasks. This method employs the concept of *skipping* which allows the robot to make progress by ignoring the tasks that cannot be completed. In this method, some tasks that can be completed may also be ignored, hence the name aggressive.

We define skipping letter $\tau \in 2^{\Pi}$ as assuming that τ is satisfied only with respect to φ_{cosafe}. That is, we assume that the robot has observed the propositions in τ only from the (logical)

Algorithm V.2 COMPUTELEAD$^{\text{AGG}}$: Subroutine to compute aggressive high-level guides

Input: A product automaton $\mathcal{P}$, a starting high-level state $(d_0, z_0^c, z_0^s) \in \mathcal{P}$, a satisfying finite word $\bar{\sigma} = \tau_0 \cdots \tau_n$, and index j of the last satisfied letter.

Output: Returns a lead, which is a sequence of high-level states that is accepting by $\mathcal{A}_{\text{safe}}$ and is a suffix of an accepting run by $\mathcal{A}_{\text{cosafe}}$.

$K \leftarrow \emptyset$

$F \leftarrow$ accepting states of $\mathcal{P}$

while $K = \emptyset$ **do**

  Run Dijkstra's all-pairs shortest-path algorithm on $\mathcal{P}$ with source $(d_0, \mathcal{A}_{\text{cosafe}}.\delta(z_0^c, \tau_j), z_0^s)$; store parent map `parent` and weight map `weight`

  $(d_g, z_g^c, z_g^s) \leftarrow \arg\min_{(d,z^c,z^s)\in F}\{\texttt{weight}[(d, z^c, z^s)]\}$

  Construct lead $K = \left((d_0, \mathcal{A}_{\text{cosafe}}.\delta(z_0^c, \tau_j), z_0^s), \ldots, \ (d_g, z_g^c, z_g^s)\right)$ using `parent` map

  $j \leftarrow j + 1$

return K

liveness perspective even though, in reality, the robot has not. Thus, the edge with label τ is traversed in $\mathcal{A}_{\text{cosafe}}$ but not in $\mathcal{A}_{\text{safe}}$. This is reflected in product automaton $\mathcal{P}$ by transitioning from the current high-level state $h_i = (d, z^c, z^s)$ to $h_j = (d, \mathcal{A}_{\text{cosafe}}.\delta(z^c, \tau), z^s)$. Note that this transition may not be valid since there is not necessarily a directed edge between h_i to h_j. However, this is purely a logical violation of φ_{cosafe} which is acceptable in the context of partial satisfaction. Meanwhile, the dynamics of the system and the safety requirements are respected by maintaining the current abstraction state d and safety automaton state z^s.

The aggressive method uses skipping to compute leads that achieve partial satisfaction of φ_{cosafe}. The pseudocode of COMPUTELEAD$^{\text{AGG}}$ for this method is shown in Alg. V.2. Recall that COMPUTELEAD$^{\text{AGG}}$ is called upon the discovery of an obstacle that makes φ unsatisfiable. Let the word obtained from the satisfying trajectory that the robot was initially set to follow be denoted by $\bar{\sigma} = \tau_0 \cdots \tau_n$, where $\tau_i \in 2^{\Pi}$. Let $j \in \{0, \ldots, n-1\}$ be the index of the letter in $\bar{\sigma}$ that the robot satisfied right before discovering the obstacle. The aggressive method generates a new lead in the newly patched $\mathcal{P}$ by skipping τ_j (i.e., first setting the current high-level state to $(d, \mathcal{A}_{\text{cosafe}}.\delta(z^c, \tau_j), z^s)$ from (d, z^c, z^s) and then computing a path to an accepting state). If a path to an accepting state does not exist in $\mathcal{P}$ from the new current high-level state, the algorithm keeps skipping letters along $\bar{\sigma}$ until either a path to an accepting state is found or τ_n is skipped.

The lead that is constructed by the aggressive algorithm always results in a word that is a suffix of an accepting word $\bar{\sigma}$. Consider the case that, after satisfying τ_j, the robot discovers an obstacle that makes proposition(s) in τ_{j+m} for some $m \in \{0, \ldots, n-j\}$ unreachable. Then, the aggressive method computes a lead to an accepting state by skipping $\tau_j, \cdots, \tau_{j+m}$. Thus, every time the aggressive method is called, a robot trajectory is obtained that satisfies the last segment of the ordered tasks in φ_{cosafe}. Furthermore, the word obtained from the actual execution of the trajectory that is computed by multiple calls of COMPUTELEAD$^{\text{AGG}}$ is a substring of a satisfying word of φ_{cosafe}. Nevertheless, this method does not guarantee the execution of all achievable tasks in φ_{cosafe}. The janitor robot example with Specification 3 is a scenario in which the aggressive method emits a desirable outcome. This method computes a plan for the robot to go to p_3 as soon as the closed door to p_5 is observed.

The properties of the overall trajectory produced by multiple calls to COMPUTELEAD$^{\text{AGG}}$ is formally stated in Lemma 2.

Lemma 2: Let $\gamma_{\mathcal{P}} = (d_0, z_0^c, z_0^s) \cdots (d_n, z_n^c, z_n^s)$ be the overall high-level path achieved by using the aggressive method's COMPUTELEAD$^{\text{AGG}}$, and let u denote the obtained word from $\gamma_{\mathcal{P}}$. Then, (1) u is a substring of a word in $\mathcal{L}(\mathcal{A}_{\text{cosafe}})$, and (2) u is in $\mathcal{L}(\mathcal{A}_{\text{safe}})$.

Proof: (1) The projection of $\gamma_{\mathcal{P}}$ onto $\mathcal{A}_{\text{cosafe}}$ is the sequence of states $\gamma_{\mathcal{A}_{\text{cosafe}}} = z_0^c \cdots z_n^c$, where z_0^c is the initial state, and z_n^c is an accepting state of $\mathcal{A}_{\text{cosafe}}$. The path $\gamma_{\mathcal{A}_{\text{cosafe}}}$ is not necessarily a valid path on $\mathcal{A}_{\text{cosafe}}$ due to skipping. By the definition of skipping, however, it is guaranteed that every $z_i^c \in \gamma_{\mathcal{A}_{\text{cosafe}}}$ is reachable from $z_{i-1}^c \in \gamma_{\mathcal{A}_{\text{cosafe}}}$ for every $i \in (0, n]$. Hence, by inserting the intermediate states, $\gamma_{\mathcal{A}_{\text{cosafe}}}$ can be expanded to an accepting run of $\mathcal{A}_{\text{cosafe}}$. Let $\bar{\sigma}$ denote the corresponding finite accepting word. Since u is the sequence of the labels (letters) over only the valid transitions along $\gamma_{\mathcal{A}_{\text{cosafe}}}$, u is a substring of $\bar{\sigma} \in \mathcal{A}_{\text{cosafe}}$. (2) From the definition of skipping, it is trivial that u is a valid word in $\mathcal{A}_{\text{safe}}$ and gives rise to the valid run of $\gamma_{\mathcal{A}_{\text{safe}}} = z_0^s \cdots z_n^s$. Since every state in $\mathcal{A}_{\text{safe}}$ is accepting (see Sec. II), and $\gamma_{\mathcal{A}_{\text{safe}}}$ is a valid run, $\gamma_{\mathcal{A}_{\text{safe}}}$ is an accepting run. Hence, $u \in \mathcal{L}(\mathcal{A}_{\text{safe}})$. □

C. Moderate Method

The moderate method is designed for the types of missions that require completing all of the achievable tasks. In other words, the trajectory generated by this method ignores *only* the tasks that are impossible to perform due to the discovered obstacles. Specification 2 for the janitor

robot example is a scenario that the moderate method produces an appealing plan.

The moderate algorithm, called COMPUTELEAD$^{\text{MOD}}$, combines the conservative method with the concept of skipping. In this algorithm, the conservative COMPUTELEAD$^{\text{CON}}$ is used at every instance of replanning. If no further leads can be computed by the conservative method, the algorithm skips an unreachable letter and calls COMPUTELEAD$^{\text{MOD}}$ recursively. Since an accepting state of $\mathcal{P}$ is bound to be reached, this process eventually terminates.

Recall that the conservative method generates a plan that makes the most possible progress towards satisfying φ_{cosafe} without violating it. The resulting word of the execution of this plan is the a prefix of the accepting words of $\mathcal{A}_{\text{cosafe}}$, whose suffix cannot be achieved in the environment. Therefore, the first letter in the corresponding suffix must be unsatisfiable in the environment. In order for the robot to advance to the remaining portion of the specification, a violation of φ_{cosafe} is needed. The moderate method performs this violation by skipping the unsatisfiable letter.

The algorithm finds this letter by first mapping the current high-level state $h_c = (d_c, z^c, z^s)$ on to the initial product automaton $\mathcal{P}_{\text{init}}$, which is constructed from the first initial map of the environment. Let $h_{\text{init}} = (d_{\text{init}}, z^c, z^s) \in \mathcal{P}_{\text{init}}$ be the corresponding high-level state. Then, an accepting path over $\mathcal{P}_{\text{init}}$ from h_{init} is computed. The word resulting from this path is an accepting suffix of the executed word. The first element of this suffix is the unsatisfiable letter and is chosen to be skipped by the algorithm. Thus, by employing the moderate method, the robot achieves a word that satisfies φ_{safe} and is a substring of an accepting word of $\mathcal{A}_{\text{cosafe}}$, where the missing letters are those that are unsatisfiable in the environment. The pseudocode of the moderate method is shown in Alg. V.3.

It should be noted that, to find the unsatisfiable letters, $\mathcal{A}_{\text{cosafe}}$ could also be used. We prefer $\mathcal{P}_{\text{init}}$ because it incorporates both the safety requirements and the environment map in addition to co-safety tasks. Moreover, $\mathcal{P}_{\text{init}}$ is guaranteed to include satisfying trajectories by the assumption that the specification is fully satisfiable in the robot's initial map.

The properties of the overall trajectory computed by the moderate method is stated in the following lemma, whose proof is identical to the one of Lemma 2.

Lemma 3: Let $\gamma_{\mathcal{P}} = (d_0, z_0^c, z_0^s) \cdots (d_n, z_n^c, z_n^s)$ be the final high-level path obtained by using the moderate method's COMPUTELEAD$^{\text{MOD}}$, and let u denote the obtained word from $\gamma_{\mathcal{P}}$. Then, (1) u is a substring of a word in $\mathcal{L}(\mathcal{A}_{\text{cosafe}})$, and (2) u is in $\mathcal{L}(\mathcal{A}_{\text{safe}})$.

Proof: The proof is identical to that of Lemma 2. □

Algorithm V.3 COMPUTELEAD$^{\text{MOD}}$: Subroutine to compute moderate high-level guides

Input: A product automaton $\mathcal{P}$, a starting high-level state $(d_0, z_0^c, z_0^s) \in \mathcal{P}$, and initial product automaton $\mathcal{P}_{\text{init}}$.

Output: Returns a lead, which is a sequence of high-level states, that is fully accepting by $\mathcal{A}_{\text{safe}}$ and is a substring of an accepting run on $\mathcal{A}_{\text{cosafe}}$ with the minimum number of greedy deletions.

$K \leftarrow \text{COMPUTELEAD}^{\text{CON}}(\mathcal{P}, (d_0, z_0^c, z_0^s))$
if $K = \emptyset$ **then**
  $(d_{\text{init}}, z_0^c, z_0^s) \leftarrow$ map (d_0, z_0^c, z_0^s) on to $\mathcal{P}_{\text{init}}$
  $K' \leftarrow \text{COMPUTELEAD}^{\text{CON}}(\mathcal{P}_{\text{init}}, (d_{\text{init}}, z_0^c, z_0^s))$
  $\tau_0 \leftarrow$ the first letter of the word corresponding to K'
  $K \leftarrow \text{COMPUTELEAD}^{\text{MOD}}(\mathcal{P}, (d_0, \mathcal{A}_{\text{cosafe}}.\delta(z_0^c, \tau_0), z_0^s))$
return K

VI. CORRECTNESS AND COMPLETENESS

The proposed LTL planning framework is correct in that the robot never violates φ_{safe} at every point of the plan and achieves φ_{cosafe} with the property of the chosen partial satisfaction method. The overall trajectory is produced by the low-level sampling-based planner guided by high-level plans. The high-level plans are correct by construction. By the assumption that the trajectory segments between sampled states are valid, i.e., collision-free with at most one label change (see Sec. II-A), the produced trajectories by the low-level planner accurately follow high-level plans. In other words, by this assumption, the event-driven trace of the trajectory accurately captures the trace that the robot obtains by executing the generated plan. Since the event-driven trace is identical to the one from the high-level plan, the final robot trajectory is correct.

Furthermore, the proposed framework is probabilistically complete for fully known environments [14]–[16]. That is, if there exists a continuous robot trajectory that satisfies the LTL specification, the probability of failing to find it goes to zero as the planning time increases. This is because the planning layers work in a synergistic manner. By assigning weights to the product automaton edges according to the exploration data, the high-level planner eventually considers all satisfying leads on $\mathcal{P}$ infinitely often. These high-level leads are assigned to the low-level planner, which is probabilistically complete, infinitely often. Such a guarantee, however, cannot be generally given for partially known environments due to the nature of iterative planning strategy. For instance, consider a robotic system, whose dynamics do not allow a reverse motion, and an environment consisting of two narrow corridors, one of which is blocked by an initially

unknown obstacle. By executing an initially satisfying trajectory that passes through the corridor with the obstacle, the robot fails to complete the task since the robot becomes stuck in the narrow passage upon the discovery of the obstacle. Instead, if the robot takes an initial trajectory that passes through the other corridor, the robot completes the task fully. Since the knowledge of the obstacle is not known *a priori*, the framework cannot guarantee that the robot can complete the mission with the computed satisfying trajectory. Nevertheless, for partially unknown environments, the framework is probabilistically complete at every (re)planning instance given the current state of the robot, the executed trajectory, and the current knowledge of the environment map. In the case that the LTL specification becomes unsatisfiable, the probabilistic completeness holds for the property of the chosen partial satisfaction method explained in Sec. V.

We note that the correctness of the partial satisfaction methods introduced above are invariant to the DFA representation, but the solution obtained by each method might change depending on the DFA representation.

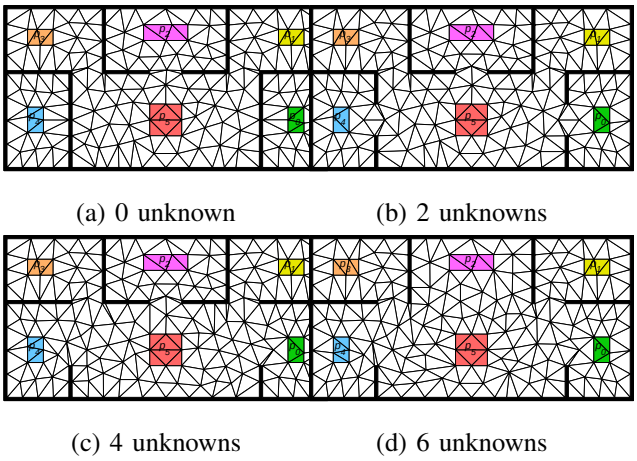
VII. Case Studies

To evaluate our planning framework, we chose a model system that is simple but illustrates the power of the approach. The simulation experiments were performed on a 3-gear car-like robot in the office environment shown in Fig. 1. The geometry of the robot was modeled as a rectangle with length of $l = 0.2$ and width of 0.1. The continuous dynamics of the robot were:

$$\dot{x} = v\cos\theta \ \ , \ \ \dot{y} = v\sin\theta \ \ , \ \ \dot{\theta} = \tfrac{v}{l}\tan\psi,$$

$$\dot{v} = u_1 \ \ , \ \ \dot{\psi} = u_2,$$

where $x \in [0, 10]$ and $y \in [0, 5]$ indicate the location of the robot, $v \in [-\frac{1}{6}, 1]$ is the linear velocity, $\theta \in [-\pi, \pi]$ is the heading angle, and $\psi \in [-\frac{\pi}{6}, \frac{\pi}{6}]$ is the steering angle. Let $g \in \{1, 2, 3\}$ denote the current gear of the robot. The robot switches to gear $g+1$ from gears 1 and 2 as soon as its velocity exceeds $\frac{g}{6}$. When in gear 2 or 3, it switches to gear $g - 1$ as soon as its velocity drops below $\frac{g-1}{6}$. In each gear, the control inputs were acceleration u_1 and steering angle velocity u_2, and their values were bounded by $u_1 \in [-\frac{1}{6}, \frac{g}{6}]$ and $u_2 \in [-\frac{\pi}{18}, \frac{\pi}{18}]$, respectively. While the robot could move in the lobby in any gear, its velocity was limited in the rooms. To enter rooms with propositions p_0, p_1, and p_2, the robot had to be in gear $g \leq 2$, and it could only move in the rooms containing p_3 and p_4 in $g = 1$. Furthermore, the robot was equipped with a range



(a) 0 unknown (b) 2 unknowns

(c) 4 unknowns (d) 6 unknowns

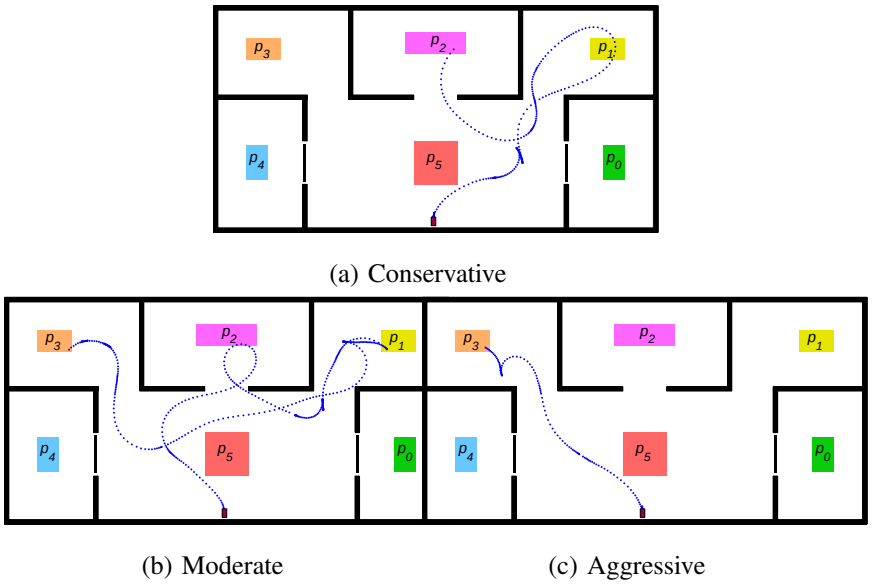
Fig. 4: Initial triangulation of environments with 0, 2, 4, and 6 unknown obstacles.

sensor that detects the unknown obstacles that are within 1.5 unit length from its center. The braking operation was performed by setting $u_1 = -\frac{1}{6}$ until the robot came to complete stop.

We modeled the motion of the robot in this environment as a hybrid system. Each gear was represented by a discrete mode of the hybrid system. The guards were the gear switching rules, and the jump function was the identity. To capture the room restrictions on the robot's velocity (gear) in the hybrid system, only the lobby was included in gear-3 mode. Moreover, all the rooms and the lobby were mapped to gear-1 mode, and gear-2 mode included the rooms with propositions p_0, p_1, and p_2 and the lobby.

We considered Specifications 1, 2, and 3 in Sec. I and III as the robot's tasks in our experiments. As explained in Sec. III, one LTL formula represents all three specifications (i.e., $\varphi = \varphi_{\text{cosafe}} \wedge \varphi_{\text{safe}}$, where φ_{cosafe} and φ_{safe} are given in (2) and (3), respectively). To capture the differences between the specifications and find the closest satisfying motion plans, we employed our planning framework and used the partial satisfaction methods of conservative, moderate, and aggressive for Specifications 1, 2, and 3, respectively. In general, the choice of the partial satisfaction method is made based on simple intuitive criteria and the intentions of the specification that are not captured by the LTL formula. The implementation of the algorithms were in C++ using OMPL [34], and all of the experiments were conducted on an AMD FX-4100 Quad-Core machine with 16 GB RAM.

In these experiments, the robot was given a partial map of the environment with 2 initially



(a) Conservative

(b) Moderate

(c) Aggressive

Fig. 5: Sample trajectories of the robot satisfying Specifications 1, 2, and 3 using conservative, moderate, and aggressive methods of partial satisfaction, respectively, in the office envrionment with 2 unkown obstacles.

unknown obstacles which is shown in Fig. 4b. The actual map is shown in Fig. 4a. It is clear that the full satisfaction of the specifications was not possible in this environment, and the robot needed to partially satisfy them. Samples of the computed motion plans for all three specifications are shown in Fig. 5. As illustrated in these figures, all the produced plans by the partial satisfaction methods respect φ_{safe} and partially satisfy φ_{cosafe} according to different measures.

We chose the conservative method for Specification 1 because it requires the type of partial satisfaction that respects the specified temporal ordering of the propositions. For the sample path shown in Fig. 5a, the robot's initial plan was a fully satisfying one since the robot planned according to the given initial map. However, on its way to p_0, the robot discovered the closed door that made p_0 inaccessible. At that point, the specification became unsatisfiable, and the robot used the conservative method of replanning. It resulted in a path that visited p_1 and p_2.

For Specification 2, we used the moderate method of partial satisfaction which finds a plan that visits all the accessible propositions. Fig. 5b shows a sample path of this method. Following the initial plan, the robot first visited p_2, and on its way to p_0, it observed the closed door. It

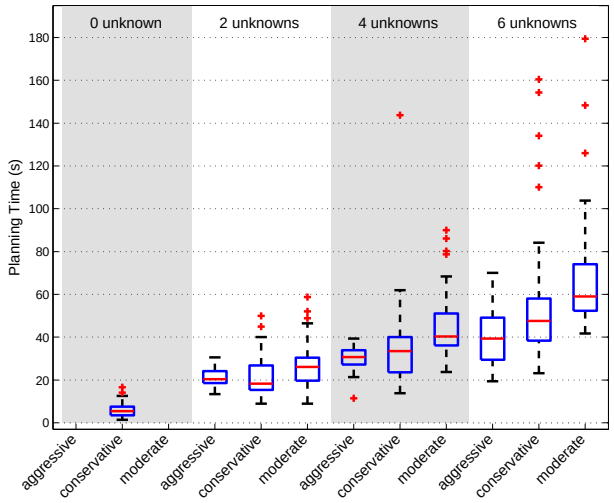


Fig. 6: Box plots of the total planning times for the robot to satisfy Specifications 1, 2, and 3 in the environments with 0, 2, 4, and 6 unknown obstacles (see Fig. 4) over 50 runs.

then replanned using the moderate method to visit p_1, p_4, and p_3. After visiting p_1 and moving towards p_4, it discovered that the door to p_4 is also closed. At that point, it replanned again to find a path to p_3.

Lastly, we used the aggressive method of partial satisfaction for Specification 3. A sample path of it is shown in Fig. 5c. In this run, the initial plan of the robot was to visit p_2 first. However, on its way to p_2, it discovered the closed door to p_4 which made the specification unsatisfiable. The robot then replanned according to aggressive method, which finds a suffix of a satisfying run. Since the specification required the sequential visit of any oder of p_0, p_1, and p_2, then p_4, and then p_3, the computed plan directed the robot to p_3.

To show the robustness of the framework, we varied the number of unknown obstacles in the initial map of the robot from 0 to 6. The initial triangulation of these maps are shown in Fig. 4. For each map and each specification, we computed motion plans 50 times. The box plots of the planning and abstraction times are shown in Fig. 6 and Fig. 7, respectively. The abstraction times include the times spent to calculate the initial abstraction in addition to the updates of it upon the discovery of an unknown obstacle. Similarly, the planning times include the computation time of the initial plan plus the replanning times needed for each run.

As illustrated in Fig. 6, the total planning time increased as the number of unknown obstacles

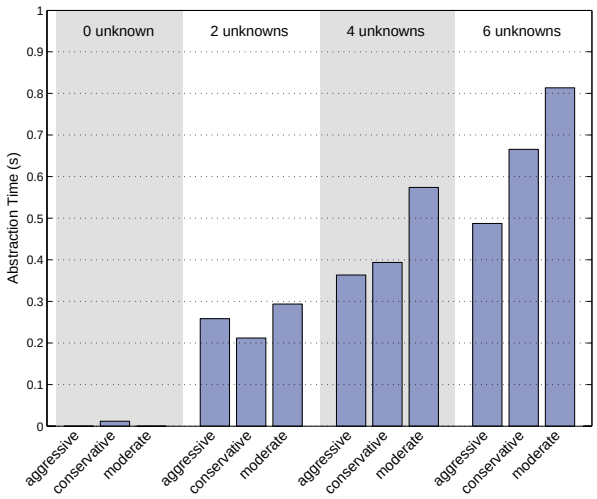


Fig. 7: Average abstraction times of the plans in Fig. 6.

increased for all three methods of partial satisfaction. For each initial map, the moderate method took the longest in planning since it needs to compute a path that visits all the accessible propositions, while the other two methods require the visits of only some of the accessible propositions. The aggressive and conservative methods took similar computation times for the initial maps with 2 and 4 unknown obstacles. However, in the 6-unknown-obstacle case, the aggressive method took less planning time than the conservative method. That is because as soon as the robot discovers the wall and door to p_4, the aggressive method directs it to p_3. As a result, the other unknown obstacles may remain undiscovered. In other words, the minimum number of unknown-obstacle discovery in the aggressive method is 2, while it is 4 for the conservative method (see Fig. 4d). Hence, the average number of replanning incidences was lower for the aggressive method than the conservative method. Moreover, on average, each replanning instance for aggressive, conservative, and moderate method took 6.17s, 7.87s, and 10.31s, respectively. With the observation that the planning times scale with the number of unknowns in the initial map, one can conclude that the iterative planning framework is best suitable for the cases where the number of unknown features in the map is small.

Abstraction times follow a similar trend as shown in Fig. 7. That is because the total abstraction time is directly related to the number of discovered obstacles. For all the variations of the initial map, the moderate method discovered all the unknown obstacles, while the other two methods

only found some of the unknown obstacles. Hence, the abstraction times for the moderate method is the highest. However, all the computed abstraction times are small which illustrate the efficiency of our method. For instance, the largest total abstraction time in the experiments (moderate method with 6 unknown-obstacle map) is less than a second on average over 50 trials.

Note that no data is presented for aggressive and moderate methods for the fully known map (0 unknown obstacle). That is because the algorithms for these methods assume that the specification is initially fully satisfiable. This is clearly not the case for the 0-unknown-obstacle map. The conservative method, however, does not hold such an assumption.

VIII. Discussion and Related Work

The problem of planning for robotic systems to satisfy high-level temporal logic specifications has been the topic of many recent studies. There are generally two approaches to this problem: automaton based and reactive synthesis based. Both methods employ the idea of a finite discrete abstraction of the motion of the continuous robotic system in its environment. In the automaton-based approach, the abstraction is used in conjunction with the automaton that represents the specification to compute a satisfying trajectory for the robot (e.g., [9], [11], [14]–[16]). In the reactive synthesis-based approach, the abstraction model along with the assumptions on the environment is used to construct a two-player game. Then, a strategy is synthesized as a state machine (hybrid controller) that encodes the necessary robot actions in response to its sensor readings to satisfy the specification (e.g., [10], [13], [22], [35]). The synthesis of the hybrid controller, however, is computationally expensive and requires time and space polynomial in the size of the reachable state space of the system [22]. To mitigate this problem, receding horizon techniques are suggested [13].

For both automaton-based and synthesis-based approaches, a (bi)simulation relation between the discrete abstraction model and the continuous system is required to guarantee correctness and completeness. Construction of such an abstraction, however, is only possible for simple dynamical systems. To allow temporal logic planning for systems with hybrid and/or complex dynamics, sampling-based motion planners are augmented to the automaton-based approaches [14]–[16], [25], [26], [36]. These works relax the (bi)simulation relation requirement and provide probabilistic completeness guarantees (i.e., if a satisfying trajectory exists, the probability of finding it grows to one over time). Recent work [37] attempts reactive synthesis for nonlinear

systems, but constructing controllers that retain the bisimilar relationship remains computationally difficult.

The results of the existing works (e.g., [18]–[21], [38]–[41]) show that robot motion planning given a high-level specification under environment uncertainty is a challenging task. In reactive synthesis-based approaches, if the geometry of the environment changes, whether due to an unknown region becoming reachable [21] or a known region becoming unreachable [19], [20], then the hybrid controller must be updated to incorporate the change. As global resynthesis of the hybrid controller is expensive, there exist approaches to locally patch the controller to incorporate the changes in less time; still, initial works in this area have shown that patching the hybrid controller can require significant time to complete [19], [20]. In automaton-based approaches, incremental planning algorithms are usually used to handle environment uncertainties, e.g., [23], [24], [41]. The main challenge with these methods also lies in the replanning time. That is because the computations of the automaton from the specification and a bisimilar discrete abstraction for the continuous system are very expensive. The work in [41] partially addresses this problem and achieves low replanning times by using the specification automaton as a monitor to ensure that the reconstruction of the automaton is not needed. That work, however, does not consider continuous dynamics for the robot and assumes that a bisimilar discrete abstraction is already available.

Our work in this paper is most closely related to [14]–[16], [24]–[26], [41], in that we are taking an automaton-based approach with iterative replanning strategy to deal with partially unknown environments. A trajectory is first computed based on the known state of the environment. During execution of the trajectory, if an unforeseen problem is faced, a new trajectory is quickly generated from the current state. This approach is inspired by replanning scenarios in robotics [30], [33].

A key question in iterative planning approaches is what to do when a specification is determined to be unsatisfiable. Many recent LTL planning studies focus on this question (e.g., [39], [40], [42]–[46]). The work in [42] is one of the first studies that tries to address this problem. That work presents an algorithm to report a reason as to why an LTL specification in the framework of reactive synthesis is unrealizable. The automaton-based works [39], [40] introduce methods of changing an unsatisfiable nondeterministic Büchi automaton into the "closest" satisfiable one, where all actions of the robot are represented using a finite state machine. To specialize the notion

of "closeness" for different tasks, the work in [43] proposes three metrics (based on simulation games), each for a type of specification. Then, based on the user's choice of the metric, a new automaton is constructed for verification/planning purposes. The LTL planner introduced in [44] allows for a temporary violation of the specification in the case that the specification is unsatisfiable. That method decomposes the specification into fragments and asks the user to prioritize them. Then, the user-defined priority list is used to synthesize the least-violating strategy. Similarly, the work in [45] employs user-defined costs to define a distance to satisfaction of co-safe LTL specifications. The costs in that work, however, are over propositions, and the authors introduce a method of constructing a weighted automoton and generating a robot plan with the least distance to satisfaction.

The work in [46] decomposes the specification into soft and hard constraints and produces a least-violating plan through the construction of a weighted product Büchi automaton. This automaton is composed of the automata representing the hard and soft constraints along with a weight function that defines the cost of violation to the soft specification. Our approach to unsatisfiable specifications differs in that we do not change/reconstruct the specification automata. Instead, we provide a metric to define "closeness" to a specification and complement it with three methods of temporary modifications to the product automaton. This combination results in three partial satisfaction techniques, each of which suitable for a set of scenarios, while allowing for online (re)planning.

IX. Conclusion and Future Work

In this paper, we have presented an iterative motion-planning framework for a hybrid robotic system with general dynamics in a partially unknown environment given a temporal logic specification consisting of co-safe and safe formulas. We have also introduced three methods of partial satisfaction in cases where obstacles in the environment prevent full satisfaction of the co-safe component of the specification. In such cases, the robotic system satisfies the co-safe specification as closely as possible according to the designer's choice of the method of partial satisfaction, while still guaranteeing that the robotic system does not violate the safe specification.

One strength of our framework lies in the ability to consider obstacles being discovered in any part of the environment. As shown in the case studies, our solution times, however, scale

with the number of obstacles that are discovered by the robot, and therefore, our framework performs best when a few obstacles are missing from the robot's initial map of the environment. Additionally, our approach is novel in how we deal with a newly discovered obstacle. Our hybrid system abstraction is geometric and solely depends on the workspace decomposition. Therefore, re-abstraction is fast upon discovery of a new obstacle. Moreover, our product automaton, the high-level structure we use for planning, keeps the task specification and the hybrid system abstraction separate. Hence, reworking the product automaton to incorporate the new obstacle does not require recomputing the specification automata, which is the most time-intensive task.

This work can be extended in at least two directions. The presented framework may benefit from a "greedy" temporal motion planning approach that begins executing a partial trajectory along a lead in the product automaton. This is to prevent the framework from generating an entire solution trajectory for a large specification, only to discover an obstacle early in that trajectory, stop, and recompute another solution trajectory. Another possible extension of this work is to add support for obstacles to disappear from the robot's initial map. One could assume a probabilistic distribution on where and when obstacles might appear and then generate trajectories that maximize probability of successful satisfaction of the task.

ACKNOWLEDGMENT

The authors would like to thank Ryan Luna and Mark Moll of Rice University for their helpful feedback and suggestions.

REFERENCES

[1] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion Theory, Algorithms, and Implementation.* Cambridge: MIT Press, 2005.

[2] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug 1996.

[3] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Department of Computer Science, Iowa State University, Ames, IA, Tech. Rep. 98-11, 1998.

[4] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics and Research*, vol. 20, no. 5, pp. 378–400, 2001.

[5] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *Intl. J. of Computational Geometry and Applications*, vol. 9, no. 4-5, pp. 495–512, 1999.

[6] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[7] I. A. Şucan and L. E. Kavraki, "A sampling-based tree planner for systems with complex dynamics," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 116–131, 2012.

[8] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 469–482, 2010.

[9] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *Int. Conf. on Robotics and Automation*. Barcelona, Spain: IEEE, 2005, pp. 2020–2025.

[10] H. Kress-Gazit, G. Fainekos, and G. J. Pappas, "Where's waldo? sensor-based temporal logic motion planning," in *Int. Conf. on Robotics and Automation*. Rome, Italy: IEEE, 2007, pp. 3116–3121.

[11] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008.

[12] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[13] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon control for temporal logic specifications," in *Int. Conf. on Hybrid Systems: Computation and Control*, 2010, pp. 101–110.

[14] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *Int. Conf. on Robotics and Automation*. IEEE, May 2010, pp. 2689–2696.

[15] ——, "Motion planning with hybrid dynamics and temporal goals," in *IEEE Conf. on Decision and Control*, 2010, pp. 1108–1115.

[16] A. Bhatia, M. Maly, L. E. Kavraki, and M. Y. Vardi, "Motion planning with complex goals," *Robotics Automation Magazine, IEEE*, vol. 18, no. 3, pp. 55–64, Sep. 2011.

[17] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT Press, 1999.

[18] Y. Chen, J. Tumová, and C. Belta, "LTL robot motion control based on automata learning of environmental dynamics," in *Int. Conf. on Robotics and Automation*. IEEE, 2012, pp. 5177–5182.

[19] S. C. Livingston, R. M. Murray, and J. W. Burdick, "Backtracking temporal logic synthesis for uncertain environments," in *Int. Conf. Robotics and Automation*. IEEE, 2012, pp. 5163–5170.

[20] S. C. Livingston, P. Prabhakar, A. B. Jose, and R. M. Murray, "Patching task-level robot controllers based on a local μ-calculus formula," in *Int. Conf. on Robotics and Automation*. IEEE, 2013, pp. 4588–4595.

[21] S. Sarid, B. Xu, and H. Kress-Gazit, "Guaranteeing high-level behaviors while exploring partially known maps," *Robotics*, pp. 377–384, 2013.

[22] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu, "Correct, reactive robot control from abstraction and temporal logic specifications," *IEEE Robotics and Automation Magazine*, vol. 18, no. 3, pp. 65–74, 2011.

[23] M. R. Maly, M. Lahijanian, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi, "Iterative temporal motion planning for hybrid systems in partially unknown environments," in *Int. Conf. on Hybrid Systems: Computation and Control*. Philadelphia, PA, USA: ACM, Apr. 2013, pp. 353–362.

[24] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Revising motion planning under linear temporal logic specifications in partially known workspaces," in *Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2013, pp. 5025–5032.

[25] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Falsification of LTL safety properties in hybrid systems," in *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, ser. ETAPS'09/TACAS'09, 2009, pp. 368–382.

[26] ——, "Falsification of LTL safety properties in hybrid systems," *Int. J. on Software Tools for Technology Transfer (STTT)*, vol. 15, no. 4, pp. 305–320, 2013.

[27] J. E. Hopcroft, *Introduction to automata theory, languages, and computation*. Pearson Addison Wesley, 2007.

[28] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, vol. 19, pp. 291–314, 2001.

[29] J. E. Hopcroft, *Introduction to automata theory, languages, and computation*. Pearson Education India, 1979.

[30] K. E. Bekris and L. E. Kavraki, "Greedy but safe replanning under kinodynamic constraints," in *Int. Conf. on Robotics and Automation*. IEEE, 2007, pp. 704–710.

[31] T. Fraichard, "A short paper about motion safety," in *Int. Conf. on Robotics and Automation*. IEEE, 2007, pp. 1140–1145.

[32] T. Latvala, "Efficient model checking of safety properties," in *Model Checking Software*. Springer, 2003, pp. 74–88.

[33] K. E. Bekris, D. K. Grady, M. Moll, and L. E. Kavraki, "Safe distributed motion coordination for second-order systems with different planning cycles," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 129–150, 2012.

[34] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.

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

[36] C. Vasile and C. Belta, "Sampling-based temporal logic path planning," in *Int. Conf. on Intelligent Robots and Systems*. IEEE, Nov 2013, pp. 4817–4822.

[37] J. A. DeCastro and H. Kress-Gazit, "Guaranteeing reactive high-level behaviors for robots with complex dynamics," in *Int. Conf. on Intelligent Robots and Systems*. IEEE, 2013, pp. 749–756.

[38] R. Bloem, K. Greimel, T. A. Henzinger, and B. Jobstmann, "Synthesizing robust systems," in *Formal Methods in Computer-Aided Design*. IEEE, 2009, pp. 85–92.

[39] K. Kim and G. Fainekos, "Approximate solutions for the minimal revision problem of specification automata," in *Int. Conf. on Intelligent Robots and Systems*. IEEE, 2012, pp. 265–271.

[40] ——, "Revision of specification automata under quantitative preferences," in *Int. Conf. on Robotics and Automation*. IEEE, 2014, pp. 5339–5344.

[41] A. Ayala, S. Andersson, and C. Belta, "Temporal logic motion planning in unknown environments," in *Int. Conf. on Intelligent Robots and Systems*, Nov 2013, pp. 5279–5284.

[42] V. Raman and H. Kress-Gazit, "Analyzing unsynthesizable specifications for high-level robot behavior using LTLMoP," in *Computer Aided Verification*. Springer, 2011, pp. 663–668.

[43] P. Černỳ, T. A. Henzinger, and A. Radhakrishna, "Simulation distances," *Theoretical Computer Science*, vol. 413, no. 1, pp. 21–35, 2012.

[44] J. Tumova, G. C. Hall, S. Karaman, E. Frazzoli, and D. Rus, "Least-violating control strategy synthesis with safety rules," in *Int. Conf. on Hybrid systems: computation and control*. ACM, 2013, pp. 1–10.

[45] M. Lahijanian, S. Almagor, D. Fried, L. E. Kavraki, and M. Y. Vardi, "This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction," in *The Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI. Austin, TX: AAAI, Jan. 2015, pp. 3664–3671.

[46] M. Guo and D. Dimarogonas, "Distributed plan reconfiguration via knowledge transfer in multi-agent systems under local ltl specifications," in *Int. Conf. on Robotics and Automation (ICRA)*. IEEE, May 2014, pp. 4304–4309.