This Time the Robot Settles for a Cost: A Quantitative Approach to Temporal Logic Planning with Partial Satisfaction

Morteza Lahijanian[1], **Shaull Almagor**[2], **Dror Fried**[1], **Lydia E. Kavraki**[1], and **Moshe Y. Vardi**[1]

[1] Department of Computer Science at Rice University, Houston, TX, 77005, USA
{morteza,dror.fried,kavraki,vardi}@rice.edu
[2] School of Computer Science and Engineering at The Hebrew University of Jerusalem, Israel
shaull@cs.huji.ac.il

Abstract

The specification of complex motion goals through temporal logics is increasingly favored in robotics to narrow the gap between task and motion planning. A major limiting factor of such logics, however, is their Boolean satisfaction condition. To relax this limitation, we introduce a method for quantifying the satisfaction of co-safe linear temporal logic specifications, and propose a planner that uses this method to synthesize robot trajectories with the optimal satisfaction value. The method assigns costs to violations of specifications from user-defined proposition costs. These violation costs define a distance to satisfaction and can be computed algorithmically using a weighted automaton. The planner utilizes this automaton and an abstraction of the robotic system to construct a product graph that captures all possible robot trajectories and their distances to satisfaction. Then, a plan with the minimum distance to satisfaction is generated by employing this graph as the high-level planner in a synergistic planning framework. The efficacy of the method is illustrated on a robot with unsatisfiable specifications in an office environment.

1 Introduction

Robots are rapidly evolving from simple instruments for repetitive tasks to increasingly sophisticated machines capable of performing challenging operations in our daily environment. As they make their way out of custom-made workspaces in factories, algorithms that *integrate task and motion planning* (ITMP) are needed to enable robots to autonomously execute high-level tasks in unstructured environments. Many works, e.g., (Dornhege et al. 2009; Srivastava et al. 2013), approach ITMP by combining discrete planning in the task space with motion planning in the robot's continuous state space. Traditionally, classical AI planners (Ghallab, Nau, and Traverso 2004) are employed at the task level. In another line of ITMP research, temporal logics, namely *linear temporal logic* (LTL) (Clarke, Grumberg, and Peled 1999), has been suggested to remove ambiguity from the specifications and formalize the planning procedure (Kress-Gazit, Fainekos, and Pappas 2007; Kloetzer and Belta 2008; Wongpiromsarn, Topcu, and Murray 2010; Bhatia et al. 2011). The power of these methods lie in the expressivity of LTL and the correctness and completeness guarantees provided by its synthesis algorithms (Bhatia, Kavraki, and Vardi 2010a; 2010b; DeCastro and Kress-Gazit 2013).

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

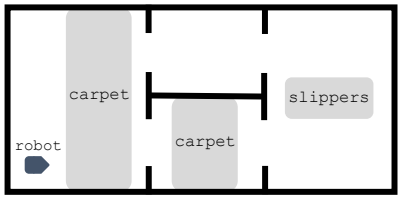


Figure 1: A mobile robot in a room with the task specification, "do not go on the carpets until the slippers are reached."

A major barrier of utilizing the existing LTL planners in realistic environments, however, is their Boolean satisfaction condition. These frameworks output a plan only if the plan fully satisfies the specification. Even though generating a satisfying plan is ideal, there are scenarios where the full realization of a task may not be possible; nevertheless, parts of the specification may still be achievable and of interest to the user. For example, consider the robotic scenario depicted in Fig. 1 with the user specification, "do not go on the carpets until the slippers are reached." Clearly, this specification is unsatisfiable as the robot must go on a carpet first to reach the slippers. Now imagine that reaching slippers is of a higher priority to the user than avoiding carpets. With this information, the desired robot behavior is obviously to reach the slippers by violating the carpet constraint only once. As robots enter unstructured environments, such scenarios become more common. This paper introduces a method for *quantification of satisfaction* of an LTL specification and proposes a planner that handles such scenarios.

Quantification of the satisfaction condition of temporal logic statements is an active area of research in theoretical computer science and logic. Recent work (Almagor, Boker, and Kupferman 2013) augments LTL with quantity-operators to allow reasoning about the quality of satisfaction. While this framework can be theoretically adapted to plan synthesis settings, in practice, the method suffers from overheads (constructing a large number of automata), which makes it, as of yet, impractical. Simulation distances have also been employed to define measures of correctness over temporal logic specifications (Černỳ, Henzinger, and Radhakrishna 2012). Their applicability in planning for robotic

systems, however, is unclear and remains to be investigated.

Existing LTL planning frameworks with partial satisfaction capabilities are still in their preliminary stages. The planner introduced in (Tumova et al. 2013) allows for a temporary violation of the specification for simple transition systems. That method decomposes the specification into fragments and asks the user to prioritize them. Once the specification is violated, the user-defined priority list is used to synthesize the least-violating strategy. The works in (Kim and Fainekos 2013; 2014) tackle the problem of minimal revision of an unsatisfiable LTL specification for a transition system. These works propose several measures of distance between Büchi automata representing the specification and introduce algorithms for specification revisions based on these measures. These revision problems are generally, however, NP-hard. The authors of (Maly et al. 2013) introduce a method of partial satisfaction based on the notion of *graph distance* in their LTL planner. Their method is conservative and prevents a violation to the specification, making the method suitable for only certain types of tasks. For the cases that a violation of the specification is required to make progress (e.g., example above), their method is undesirable.

Planning with partial satisfaction has also been studied in the AI discrete-planning community under the notion of *preference-based planning*, e.g., (Boutilier et al. 2004; Baier et al. 2008; Coles and Coles 2011). These works introduce various models of reasoning about conditional preferences such as CP-nets (Boutilier et al. 2004). One of such planners is $\mathcal{LPP}$ (Baier et al. 2008), which allows the specification of preference over plans using preference formulas. The planner requires explicit ranking by the user, and although complex preference structures can be implied by the formulas, every preference order has to be specified.

Although several works (Bacchus and Kabanza 1998; De Giacomo and Vardi 1999; Kvarnström and Doherty 2000; Calvanese, Giacomo, and Vardi 2002) in AI planning use LTL for task specification, the main approach to using temporal reasoning in addition to preferences uses the PDDL3 framework (Gerevini et al. 2009), e.g., HTNPLAN-P (Sohrabi, Baier, and McIlraith 2009), LPRPG-P (Coles and Coles 2011), and MaxSat (Juma, Hsu, and McIlraith 2012)). PDDL3 is a specification language which enables the use of basic temporal operators in expressing (hard and soft) constraints and uses quantitative preferences to rank the soft constraints (Gerevini et al. 2009). Unlike LTL, the syntax of PDDL3 does not support arbitrary nesting of temporal operators; the *next* operator is also disallowed. Thus, the temporal expressivity of the language is limited.

This work proposes a quantitative approach to LTL planning. The approach views atomic propositions in an LTL specification as tasks, for which the user can assign priorities. These proposition priorities are then translated to costs of violations of the specification. These costs define a distance between system trajectories with respect to the satisfaction of the specification. Given a robotic system, a specification in a fragment of LTL, and costs over propositions, the planning framework automatically synthesizes a continuous plan with the minimum distance to satisfaction of the specification. Technically, the planner first constructs a weighted automaton by combining the proposition costs with the automaton that represents the specification. Next, the framework computes a discrete abstraction for the continuous system and composes the abstraction with the weighted automaton. The result is a weighted graph, which is employed to generate high-level plans in the multi-layered planning framework in (Maly et al. 2013). These high-level plans, through a synergy layer, guide a low-level planner to explore the state space of the robotic system to generate a system trajectory with the minimum distance to satisfaction.

The main contribution of this work is a quantitative method to LTL planning, which allows quantitative reasoning over the trajectories of a continuous robotic system with respect to the specification. The use of LTL enables more expressive and complex temporal specifications than PDDL3. Unlike the works in (Baier et al. 2008; Tumova et al. 2013), the costs in our work are defined over atomic propositions (rather than specification segments), and unlike the discrete planner in (Kim and Fainekos 2013; 2014), the complexity of our high-level planner is PTIME. Furthermore, this work extends (Maly et al. 2013) by enabling the planner to generate trajectories that violate the specification while maintaining the same planning efficiency. Hence, the proposed framework also allows for online planning and can be employed in partially known environments. The power of our method is illustrated through case studies in which robot trajectories were synthesized in seconds from rich specifications in both fully and partially known environments.

2 Preliminaries

We use syntactically co-safe LTL (Kupferman and Vardi 2001) to write the specifications of robotic tasks.

Definition 1 (syntax) *Let $\Pi = \{p_1, p_2, \ldots, p_N\}$ be a set of Boolean atomic propositions. A syntactically co-safe* LTL *formula over Π is inductively defined as following:*

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

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 (Semantics) *The semantics of syntactically co-safe* LTL *formulas are defined over infinite traces over 2^{Π}. Let $w = \{w_i\}_{i=1}^{\infty}$ with $w_i \in 2^{\Pi}$ be an infinite trace and $w^i = w_i w_{i+1} \ldots$ be the i-th suffix. $w \models \varphi$ indicates that w satisfies formula φ and is recursively defined as following:*

- $w \models p$ *if* $p \in w_1$;
- $w \models \neg p$ *if* $p \notin w_1$;
- $w \models \varphi_1 \vee \varphi_2$ *if* $w \models \varphi_1$ *or* $w \models \varphi_2$;
- $w \models \varphi_1 \wedge \varphi_2$ *if* $w \models \varphi_1$ *and* $w \models \varphi_2$;
- $w \models \mathcal{X}\varphi$ *if* $w^1 \models \varphi$;
- $w \models \varphi_1 \mathcal{U} \varphi_2$ *if* $\exists k \geq 0$, $w^k \models \varphi_2$, *and* $\forall i \in [0, k)$, $w^i \models \varphi_1$;
- $w \models \mathcal{F}\varphi$ *if* $\exists k \geq 0$, $w^k \models \varphi$.

A computation w satisfies a co-safe LTL formula φ iff there exists a "good" finite prefix $\overline{w}$ of w such that $\overline{w}\underline{w}$ satisfies φ for every suffix $\underline{w}$. Thus, we can restrict to analyzing the language of good prefixes, which consists of finite traces.

Given a co-safe LTL formula φ, a *deterministic finite automaton* (DFA) that precisely accepts all the good prefixes that satisfy φ can be constructed (Kupferman and Vardi 2001).

Definition 3 (DFA) *A* deterministic finite automaton *is given by a tuple $\mathcal{A} = (Z, \Sigma, \delta, z_0, F)$, where*

- *Z is a finite set of states;*
- *Σ is the input alphabet;*
- *$\delta : Z \times \Sigma \to Z$ is the transition function;*
- *$z_0 \in Z$ is the initial state;*
- *$F \subseteq Z$ is the set of accepting states.*

The transition function δ can be also viewed as a relation $\delta \subseteq Z \times \Sigma \times Z$, where every transition is a tuple $(z_1, \sigma, z_2) \in \delta$ iff $z_2 = \delta(z_1, \sigma)$ (with abuse of notation). A finite run of $\mathcal{A}$ on a word $w = w_1 \cdots w_n$ is a sequence of states $\mu = \mu_0 \mu_1 \ldots \mu_n$, where $\mu_0 = z_0$ and $(\mu_{i-1}, w_i, \mu_i) \in \delta$ for $i = 1, \ldots, n$. μ is called an accepting run if $\mu_n \in F$.

We denote the DFA that is constructed from a formula φ by $\mathcal{A}_\varphi$. An input word w that induces an accepting run μ in $\mathcal{A}_\varphi$ is called φ-satisfying. To reason quantitatively over the satisfaction of φ, we employ *weighted* DFA defined below.

Definition 4 (WDFA) *A* weighted DFA *(*WDFA*) is a tuple $\mathcal{A}^\rho = (\mathcal{A}, \rho)$, where $\mathcal{A}$ is a* DFA*, and $\rho : \delta \to \mathbb{R}$ assigns a weight for every transition in δ. Consider a word $w = w_1, ..., w_n$, and let $\mu = \mu_0 \mu_1 \ldots \mu_n$ be the run of $\mathcal{A}^\rho$ on w, i.e., $(\mu_{i-1}, w_i, \mu_i) \in \delta$ for all $i \in \{1, \ldots, n\}$. We define the weight of w, with an abuse of notation, to be $\rho(w) = \sum_{i=1}^{n} \rho(\mu_{i-1}, w_i, \mu_i)$. Thus, the weight of a word is the sum of weights along the run of $\mathcal{A}$ on it. Therefore, a* WDFA *defines a function $h_{\mathcal{A}^\rho} : \Sigma^* \to \mathbb{R}$.*

3 Problem Formulation

Consider a robotic system with the following dynamics:

$$\dot{x} = f(x, u), \quad x \in X \subset \mathbb{R}^n, \qquad u \in U \subset \mathbb{R}^m, \qquad (1)$$

where X and U are compact sets representing state and input spaces, respectively, and $f : X \times U \to X$ is an integrable and possibly nonlinear function. We assume that state x is fully observable at all times. The robot moves in a 2-dimensional static environment W (i.e., W is the projection of X onto $\mathbb{R}^2$). It consists of a finite set of polytopic obstacles and a finite set of (possibly intersecting) polytopic regions of interest $R = \{r_0, \ldots, r_l\}$, where $l \in \mathbb{N}$. Each region r_i is labeled with an atomic proposition that becomes true when the robot visits r_i. Let $\Pi = \{p_0, \ldots, p_N\}$ denote the set of all atomic propositions.

Robot's high-level task specification is given as a co-safe LTL formula φ defined over Π. Furthermore, each proposition p_j is assigned a non-negative cost in accordance with its importance in φ. Let $c : \Pi \to \mathbb{R}^{\geq 0}$ denote this cost function. Conceptually, p_j represents a task in φ, and $c(p_j)$ is the penalty that the robot has to endure if it violates (fails to perform) task p_j during the execution of φ. We define a robot trajectory's "distance to satisfaction" of φ to be the summed cost of the violations of the tasks in φ (formal definition in Sec. 4). If the trajectory fulfills all of the specified tasks in φ, the distance to satisfaction of φ by the robot is zero.

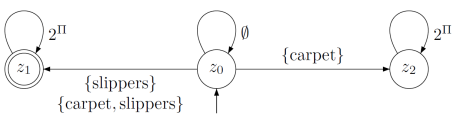


Figure 2: A DFA for $\varphi = (\neg\text{carpet}\ \mathcal{U}\ \text{slippers})$.

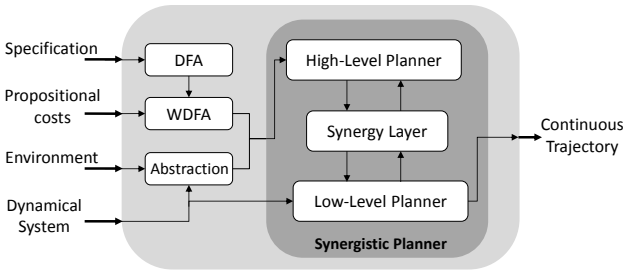


Figure 3: A block diagram representation of the approach.

To illustrate this setting, consider again the robot scenario depicted in Fig. 1. The regions of interest in this environment are the gray rectangles with the atomic propositions $\Pi = \{\text{carpet}, \text{slippers}\}$. The robot's mission translates to the co-safe LTL formula $\varphi = (\neg\text{carpet}\ \mathcal{U}\ \text{slippers})$. The DFA $\mathcal{A}_\varphi$ is shown in Fig. 2. Since the user considers slippers being of higher priority than the carpet, she assigns the following proposition costs: $c(\text{carpet}) = 1$ and $c(\text{slippers}) = 10$. There are many ways for the robot to partially satisfy the specifications. One way is to respect the carpet constraint and avoid going on it by remaining in its initial position. With this behavior, the robot never reaches slippers and endures a cost of 10. Another possible way to partially satisfy the specification is to reach slippers by violating the carpet constraint only once, i.e., only crossing over the first carpet. The cost of this behavior is 1. Obviously, the robot chooses the latter behavior to endure a smaller cost, which is of course in-line with the user's preference.

We are interested in computing a robot plan with the least distance to satisfaction of φ according to the user's preference. The formal statement of this problem follows.

Problem 1 *Given a robot that evolves in environment W according to System* (1)*, a high-level task specification given as a co-safe* LTL *formula φ, and an atomic proposition cost function c, compute a continuous trajectory for the robot that minimizes the distance to satisfaction of φ.*

To approach Problem 1, we first design a planner that is capable of producing discrete plans with minimum distance to satisfaction of φ, and then use it as the high-level planner in the multi-layer planning framework proposed in (Maly et al. 2013) to generate continuous robot trajectories. To construct the discrete planner, we augment the DFA representing φ with a cost function to capture the distance to satisfaction over the runs of the DFA. The resulting structure is a WDFA. We compose this WDFA with a geometric abstraction of System (1) to construct high-level plans (sequence of abstraction states). This composition is in the form of a

Cartesian product and captures both the specification objectives and the environment constraints. High-level paths over the abstraction with minimum distance to satisfaction of φ are suggested to the low-level planner by the synergy layer to construct a continuous trajectory for System (1) that minimizes the distance to satisfaction of φ. Fig. 3 illustrates a block digram representation of this approach.

4 Distance to Satisfaction

This section formalizes the notion of *distance to satisfaction*.

4.1 Weighted Skipping

Recall that for a co-safe LTL formula φ over the atomic propositions in Π, there exists a DFA $\mathcal{A}_\varphi$ that accepts word $w \in (2^\Pi)^*$ iff w satisfies φ. We refer to an accepting word as a *logical word* and denote it by w^φ. The set of all logical words constitutes the language of $\mathcal{A}_\varphi$, which is denoted by $\mathcal{L}(\mathcal{A}_\varphi)$. Recall that propositions in Π correspond to the regions of interest in environment W. Thus, by visiting these regions, the robot generates words also in $(2^\Pi)^*$. We refer to each of these words as a *physical word* and denote it by $w^\mathcal{R}$. Specifically, a physical word is the sequence of letters of the regions that the robot visits, where a letter is appended to the word only if it differs from the previous letter. The language of the robot, denoted by $\mathcal{L}(\mathcal{R})$, is the set of all possible physical words that the robot can generate in W. Therefore, the specification φ is said to be unsatisfiable by the robot in W if there does not exist a physical word that is accepted by $\mathcal{A}_\varphi$, i.e., $\mathcal{L}(\mathcal{R}) \cap \mathcal{L}(\mathcal{A}_\varphi) = \emptyset$. In such a case, we are interested in generating a robot trajectory with a $w^\mathcal{R} \in \mathcal{L}(\mathcal{R})$ that has the minimum distance to the language of $\mathcal{A}_\varphi$. We call the distance of a physical word to $\mathcal{L}(\mathcal{A}_\varphi)$ *distance to satisfaction* (of $w^\mathcal{R}$ to φ). To properly define this distance, we employ the concept of *weighted skipping*, which we explain through the following example.

Consider again the robotic scenario in Fig. 1 with the specification DFA $\mathcal{A}_\varphi$ shown in Fig. 2. Given that φ is unsatisfiable, the preferred robot behavior is to reach the slippers by going only over the first carpet, corresponding to the physical word $w^\mathcal{R} = \emptyset\{\text{carpet}\}\emptyset\{\text{slippers}\}$ ($\emptyset$ is the observation letter of the regions with no proposition). To construct this word from $\mathcal{A}_\varphi$, the robot can utilize the logical word $w^\varphi = \emptyset\emptyset\emptyset\{\text{slippers}\}$ as a guide, and substitute the physically unachievable letter $w_2^\varphi = \emptyset$ with the accessible one $w_2^\mathcal{R} = \{\text{carpet}\}$ at a certain cost. With this behavior, the robot essentially *skips* over a logical letter (i.e., a transition in $\mathcal{A}_\varphi$) by reading a different physical letter. We name this method *weighted skipping*. Another possible behavior of the robot is to avoid the carpet by remaining in its initial position. The physical word of this behavior is $w^\mathcal{R} = \emptyset$, which can be generated by following $w^\varphi = \{\text{slippers}\}$ and enduring the cost of skipping letter $w_0^\varphi = \{\text{slippers}\}$. Recall that this robot behavior, however, is not preferred by the user.

To generate the motion plans that do meet the user's preferences, we derive the cost of weighted skipping from the user-defined proposition cost function c. Let $\eta : 2^\Pi \times 2^\Pi \to \mathbb{R}^{\geq 0}$ be a function that measures the cost of skipping a letter τ in w^φ using letter σ in $w^\mathcal{R}$. Thus, $\eta(\sigma, \tau) = 0$ when $\sigma = \tau$. For $\sigma \neq \tau$, the value of $\eta(\sigma, \tau)$ can be derived from c in several ways. We advocate the following two methods:

- *max* semantics: $\eta(\sigma, \tau) = \max_{p \in \sigma \Delta \tau} \{c(p)\}$,
- *sum* semantics: $\eta(\sigma, \tau) = \sum_{p \in \sigma \Delta \tau} \{c(p)\}$,

where Δ is the symmetric difference. The two semantics of η capture different relationships between the atomic propositions in Π. In the *max* semantics, we essentially consider dependent tasks, where skipping over several tasks is no worse than skipping over the most important one. The *sum* semantics correspond to independent tasks, where the cost of skipping several tasks is the accumulation of their costs. Therefore, the user should choose the applicable η semantic based on her intuition of the problem.

Using the function η, we define the notion of distance to satisfaction. Let the distance between physical word $w^\mathcal{R}$ and logical word w^φ be:

$$\textsc{Dist}(w^\mathcal{R}, w^\varphi) = \begin{cases} \sum_{i=1}^{|w^\varphi|} \eta(w_i^\mathcal{R}, w_i^\varphi) & \text{if} \quad |w^\mathcal{R}| = |w^\varphi| \\ \infty & \text{if} \quad |w^\mathcal{R}| \neq |w^\varphi| \end{cases}$$

where $|w|$ denotes the number of letters in w. Then, the distance to satisfaction of $w^\mathcal{R}$ to φ is:

$$\textsc{DistToSat}(w^\mathcal{R}, \varphi) = \min_{w^\varphi \in \mathcal{L}(\mathcal{A}_\varphi)} \textsc{Dist}(w^\mathcal{R}, w^\varphi). \quad (2)$$

Conceptually, $\textsc{DistToSat}(w^\mathcal{R}, \varphi)$ is the minimum total cost of the weighted skips that the robot has to perform for the trajectory with $w^\mathcal{R}$. Therefore, we are interested in computing a trajectory whose $w^\mathcal{R}$ minimizes (2).

To illustrate these concepts, consider again the two robot behaviors described above. In the first behavior, the distance between $w^\mathcal{R} = \emptyset\{\text{carpet}\}\emptyset\{\text{slippers}\}$ and $w^\varphi = \emptyset\emptyset\emptyset\{\text{slippers}\}$ is essentially the distance between the letters $\{\text{carpet}\}$ and $\emptyset$. Since letter $\{\text{carpet}\}$ is a singleton, and $c(\text{carpet}) = 1$, then $\textsc{Dist}(w^\mathcal{R}, w^\varphi) = \eta(\{\text{carpet}\}, \emptyset) = 1$ using either *max* or *sum* semantics. In the second behavior, the distance between words $w^\mathcal{R} = \emptyset$ and $w^\varphi = \{\text{slippers}\}$ is 10 since $c(\text{slippers}) = 10$. Therefore, between these two behaviors, the robot chooses the first one, which has a smaller distance to satisfaction of φ.

4.2 Co-safe LTL to WDFA

In order to algorithmically compute (2), we incorporate the idea of weighted skipping along with the proposition cost function c into the specification DFA. The result is a WDFA with a pair of labels in $2^\Pi \times 2^\Pi$ on every edge; the first label is a physical letter that is used to skip the logical letter (second label) inherited from the DFA. The transition weights of the WDFA are given by η.

Formally, let $\Sigma = 2^\Pi$, $\mathcal{A}_\varphi = (Z, \Sigma, \delta, z_0, F)$ be the specification DFA, and $\eta : \Sigma \times \Sigma \to \mathbb{R}^{\geq 0}$. From $\mathcal{A}_\varphi$, we construct the WDFA $\mathcal{A}_\varphi^\rho = (Z, \Sigma \times \Sigma, \delta^\rho, z_0, F, \rho)$, where for every state $z \in Z$ and letter $(\sigma, \tau) \in \Sigma \times \Sigma$, the transition function is $\delta^\rho(z, (\sigma, \tau)) = \delta(z, \tau)$. Moreover, for every transition $(z, (\sigma, \tau), z') \in \delta^\rho$, the weight is defined to be $\rho((z, (\sigma, \tau), z')) = \eta(\sigma, \tau)$.

Note that the transition functions δ^ρ of $\mathcal{A}_\varphi^\rho$ and δ of $\mathcal{A}_\varphi$ are the same with respect to the logical letter τ. The difference is that δ^ρ reads an additional (physical) letter σ, by

which δ^ρ allows skipping τ. Intuitively, $\mathcal{A}^\rho_\varphi$ uses δ of $\mathcal{A}_\varphi$ and τ to determine the output state of a transition once a physical letter σ is read, and assigns the cost (weight) of $\eta(\sigma, \tau)$ to this transition. Therefore, a word $w \in (\Sigma \times \Sigma)^*$ is accepted by $\mathcal{A}^\rho_\varphi$ iff the projection of w on its logical labels is accepted by $\mathcal{A}_\varphi$. The projection of an accepting word of $\mathcal{A}^\rho_\varphi$ on its physical labels, however, may not be accepted by $\mathcal{A}_\varphi$.

The construction of the WDFA $\mathcal{A}^\rho_\varphi$ holds no significant blowup. The states of $\mathcal{A}^\rho_\varphi$ are the same as the states of the DFA $\mathcal{A}_\varphi$. The number of edges of $\mathcal{A}^\rho_\varphi$ is at most quadratic in the number of edges of $\mathcal{A}_\varphi$ since there are $|\Sigma|$ outgoing edges from each state of $\mathcal{A}_\varphi$. Furthermore, the explicit construction of $\mathcal{A}^\rho_\varphi$ is not even required for the implementation purposes as it can be simulated on-the-fly using $\mathcal{A}_\varphi$ (for the state to transition to) and η (for the cost of the transition).

5 Planning Framework

Our planning framework is based on the synergistic planner proposed in (Maly et al. 2013), which consists of three main layers: a high-level planner, a low-level search layer, and a synergy layer. We replace the high-level planner with a new one to allow quantitative partial satisfaction. This high-level planner generates a discrete path that best satisfies the specification (in the sense of Sec. 4) by searching over a *product graph* that combines a discrete abstraction of the robotic system with the WDFA. This path is suggested to the low-level planner which uses the dynamics of the system to extend a sampling-based tree in the direction of the path. During the tree expansion, the synergy layer collects data to learn the relationship between the abstraction and the dynamics of the system. This information is continually updated and passed to the high-level planner to generate informed paths. The synergy of these planing layers results in a trajectory for System (1) that minimizes distance to satisfaction of φ.

5.1 Abstraction

To construct high-level plans, an abstraction of System (1) is first generated. This abstraction is denoted by $\mathcal{R} = (D, d_0, \rightarrow_D, \Pi, L)$, 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 $L : D \rightarrow 2^\Pi$ is a labeling function.

To obtain $\mathcal{R}$, environment W is decomposed into a set of discrete regions that respects the regions of interests in R and the boundaries of the obstacles (Maly et al. 2013). This decomposition of W induces a discretization in state space X of System (1) since the projection of X onto $\mathbb{R}^2$ (the x and y components of X) is W. Each of these discrete regions is represented as a discrete state $d \in D$ in the abstraction $\mathcal{R}$. The transition $\rightarrow_D$ captures the adjacency relationship between the discrete regions, i.e., $(d, d') \in \rightarrow_D$ for $d, d' \in D$ iff the corresponding regions in X share a facet. Finally, the labeling function L assigns to each discrete region in X its corresponding environment atomic proposition.

5.2 Product Graph

High-level plans are computed over the product graph $\mathcal{P} = \mathcal{R} \times \mathcal{A}^\rho_\varphi$. Recall that the input consists of a specification φ over the atomic propositions Π, and a cost function $c : \Pi \rightarrow \mathbb{R}^{\geq 0}$. From these, WDFA $\mathcal{A}^\rho_\varphi = (Z, 2^\Pi \times 2^\Pi, \delta^\rho, z_0, F, \rho)$ is constructed per Sec. 4. Then $\mathcal{R}$ is composed with $\mathcal{A}^\rho_\varphi$ to obtain a labeled weighted graph $\mathcal{P} = (Q, q_0, E, T, C_\varphi, C_\mathcal{R})$, where

- $Q = D \times Z$ is a set of high-level states,
- $q_0 = (d_0, z_0)$ is the initial high-level state,
- $E \subseteq Q \times Q$ is a set of edges,
- $T = D \times F$ is a set of terminal states,
- $C_\varphi : E \rightarrow \mathbb{R}$ is the *logical weight function* which assigns to each edge $e \in E$ its corresponding weight in $\mathcal{A}^\rho_\varphi$ as explained below,
- $C_\mathcal{R} : E \rightarrow \mathbb{R}$ is the *abstraction weight function* which assigns to each edge $e \in E$ a weight representing the "difficulty" of realizing e by the dynamics of System (1).

The construction of E and C_φ are as follows. From high-level states $q = (z, d)$ to $q' = (z', d')$, there exists an edge $e = (q, q') \in E$ if,

- $(d, d') \in \rightarrow_D$, $L(d') = \emptyset$ and $z' = z$, then $C_\varphi(e) = 0$, or
- $(d, d') \in \rightarrow_D$, $L(d') = \sigma$ and $z' = \delta^\rho(z, (\sigma, \tau))$ where $\tau \in 2^\Pi$, then $C_\varphi(e) = \rho(z, (\sigma, \tau), z')$.

The abstraction weight $C_\mathcal{R}(e) = 1/\gamma(q)\gamma(q')$, where $\gamma(q)$ is the level of difficulty for System (1) to navigate in the region corresponding to $q = (d, z)$. The value of $\gamma(q)$ is estimated by the volume of d, the number of tree vertices generated by the low-level planner in (d, z), and the number of attempts to expand the tree in (d, z). The exact form of γ is given in (Maly et al. 2013). As the planning framework progresses, the abstraction weights are continually updated.

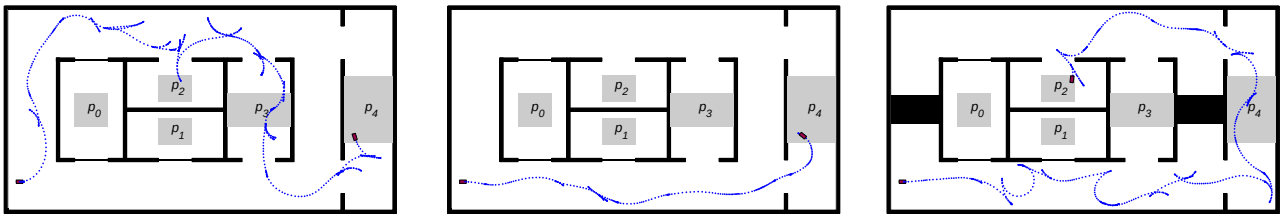
Recall that the goal of the high-level planner is to suggest paths over $\mathcal{P}$ to the low-level planner that are minimal with respect to the distance to satisfaction of φ and are easy to navigate by System (1). Let $\mu = (d_0, z_0) \dots (d_k, z_k)$ with $(d_k, z_k) \in T$ be a path over $\mathcal{P}$. The projection of μ onto $\mathcal{A}_\varphi$ is an accepting run with the logical word $w^\varphi = \tau_1 \dots \tau_k$. The projection of μ on $\mathcal{R}$ realizes physical word $w^\mathcal{R} = \sigma_1 \dots \sigma_k$ and is, in fact, the actual word that System (1) achieves in the environment. As defined in Sec. 4, the distance to satisfaction of φ for the high-level path μ is the distance between words w^φ and $w^\mathcal{R}$. This distance to satisfaction can be computed by

$$\sum_{i=0}^{k-1} C_\varphi \left((\mu_i, \mu_{i+1}) \right). \tag{3}$$

Furthermore, the difficulty of generating a continuous trajectory in the direction of path μ for System (1) is

$$\sum_{i=0}^{k-1} C_\mathcal{R} \left((\mu_i, \mu_{i+1}) \right). \tag{4}$$

Therefore, a path that minimizes both (3) and (4) is sought by the high-level planner. To compute this path, we treat the minimization of the distance to satisfaction in (3) as a hard constraint and the minimization of the abstraction weights in (4) as a soft constraint. Thus, this path can be found by a lexicographic order on $\mathcal{P}$.



(a) Specification 1 (b) Specification 2 (c) Specification 3

Figure 4: Sample trajectories for the case study 1. The robot is shown as a small burgundy rectangle, and its initial position was on the lower left of the environment. The large black rectangles in (c) represent obstacles.

5.3 Continuous Planning

Once the optimal high-level path μ is computed, μ is suggested to the low-level search layer. This planning layer extends a sampling-based motion tree in the high-level states in μ. If parts of μ are already explored, state $q \in \mu$ is picked for tree expansion with probability $\gamma(q)/\sum_{q' \in \mu} \gamma(q')$. Then one iteration of the low-level tree planner is performed to extend the set of tree vertices associated with the high-level state q. This process repeats until a terminal high-level state in T is reached (Maly et al. 2013).

6 Case Studies

We evaluated the performance of the proposed planner by conducting simulation experiments on a car-like robot in an indoor environment. We employed the robot used in (Maly et al. 2013), which had rectangle geometry with the length of $l_r = 0.2$ and width of 0.1. The dynamics were $\dot{x} = v\cos\theta$, $\dot{y} = v\sin\theta$, $\dot{\theta} = \frac{v}{l_r}\tan\psi$, $\dot{v} = u_1$, $\dot{\psi} = u_2$, where $x \in [0, 12]$ and $y \in [0, 6]$ indicate the location of the rear axle of the robot, $v \in [-\frac{1}{2}, \frac{1}{2}]$ is the linear velocity, $\theta \in [-\frac{\pi}{4}, \frac{\pi}{4}]$ is the heading angle, and $\psi \in [-\frac{\pi}{6}, \frac{\pi}{6}]$ is the steering angle. The control inputs were $u_1 \in [-\frac{1}{2}, \frac{1}{2}]$ and $u_2 \in [-\frac{\pi}{18}, \frac{\pi}{18}]$.

The indoor environment consisted of corridors and five rooms, four offices and a large conference room, with doors (see Fig. 4). The regions of interest in this environment are represented as gray rectangles with labels $p_0, \ldots, p_3$ in the offices and p_4 in the conference room. Two variations of this environment were considered for experiments. In the first environment, the corridors were obstacle free as shown in Fig. 4a and 4b. In the second environment, the corridors were blocked by two obstacles as shown in Fig. 4c. In both environments, the office doors to p_0 and p_1 were closed.

Robot's task specifications in the first environment were:

Specification 1 *Visit regions $p_0, \ldots, p_4$ in that order.*

Specification 2 *Either visit region p_1 followed by p_3 or visit region p_0 followed by p_4.*

Robot's task specification in the second environment was,

Specification 3 *Without passing over p_3 and p_4, first visit p_2 and then either visit p_1 followed by p_3 or visit p_0.*

Setting $\Pi = \{p_0, \ldots, p_4\}$, these specifications translate to:

$$\varphi_1 = \mathcal{F}(p_0 \wedge \mathcal{X}\mathcal{F}(p_1 \wedge \mathcal{X}\mathcal{F}(p_2 \wedge \mathcal{X}\mathcal{F}(p_3 \wedge \mathcal{X}\mathcal{F}p_4)))),$$
$$\varphi_2 = \mathcal{F}(p_1 \wedge \mathcal{X}\mathcal{F}p_3) \vee \mathcal{F}(p_0 \wedge \mathcal{X}\mathcal{F}p_4),$$
$$\varphi_3 = \neg(p_3 \vee p_4)\,\mathcal{U}\,(p_2 \wedge \mathcal{X}\,(\mathcal{F}(p_1 \wedge \mathcal{X}\mathcal{F}p_3) \vee \mathcal{F}p_0))\,,$$

where φ_1, φ_2, and φ_3 correspond to Specifications 1, 2, and 3, respectively. Furthermore, the propositional costs were: $c(p_0) = c(p_2) = c(p_4) = 1$, $c(p_1) = 3$, $c(p_3) = 2$. From c, we constructed η per Sec. 4. Note that η returns the same values using either methods of *max* and *sum* since the regions of interest are non-intersecting in the environments.

The implementation of our algorithm is in C++ using OMPL (Şucan, Moll, and Kavraki 2012). All of the case studies were run on an AMD FX-4100 Quad-Core machine with 16 GB RAM.

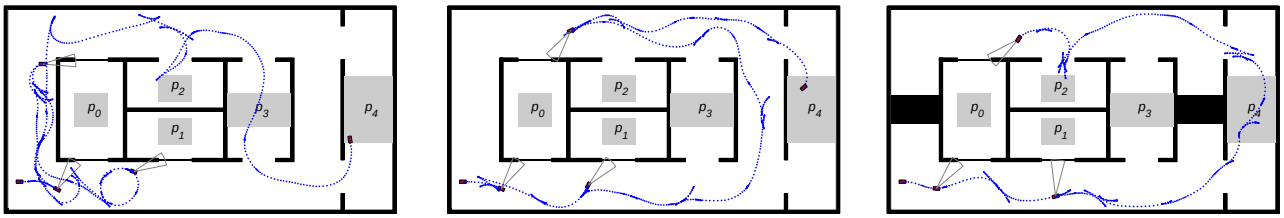
6.1 Case Study 1: Fully Known Environments

In the first set of experiments, the robot was given the full knowledge of the environments. It is clear that all three specifications are unsatisfiable in these environments due to the closed doors. To partially satisfy the specifications according to the user's preference, motion plans were computed with the least distances to satisfaction of φ_1, φ_2, and φ_3 using the proposed planning framework. Samples of the computed plans for the robot are shown in Fig. 4, and the planning data for 50 runs are shown in Table 1.

For Specification 1, which requires a sequential visit to all of the proposition regions, the computed motion plan took the robot to p_2, p_3, and p_4 by skipping the unaccessible propositions p_0 and p_1 as shown in Fig 4a. The distance to satisfaction of this trajectory is 4. As shown in Fig. 4b, the sample trajectory with the least distance to satisfaction of Specification 2 only visited p_4. Recall that Specification 2 mandated the robot to visit either "p_1 followed by p_3" or "p_0 and then p_4." Given that both p_0 and p_1 were unaccessible, the robot chose the latter because violating p_0 is less costly than violating p_1. The distance to satisfaction of this trajectory was 1. The trajectory with minimum distance to satisfaction of Specification 3 is shown in Fig. 4c. Recall that this specification required the robot to avoid p_3 and p_4 until p_2 and then either p_1 and then p_3 or p_0 are visited. Since the corridors are blocked in this environment, the robot had to pass through either p_3 or p_4 to visit p_2. It decided to violate p_4, which has a lower violation cost than p_3, and terminated in p_2 because skipping p_0 is less costly than violating p_1. The distance to satisfaction of this trajectory was 2. All these robot behaviors are in-line with the user's preferences.

6.2 Case Study 2: Partially Known Environments

In the second set of experiments, the robot was not provided the knowledge of the status of the doors in the environments.



(a) Specification 1 (b) Specification 2 (c) Specification 3

Figure 5: Sample trajectories for case study 2. The arcs in the figures indicate the moment the closed doors were detected.

	Case Study 1	Case Study 2	
Spec.	T_{plan} (s)	$N_{\text{plannerCalls}}$	T_{plan} (s)
φ_1	4.13	4.0	23.17
φ_2	2.36	4.0	9.69
φ_3	2.66	4.0	7.69

Table 1: Planning data for φ_1, φ_2, and φ_3 in case studies 1 and 2. $N_{\text{plannerCalls}}$ and T_{plan} are the average number of planner calls and planning times, respectively, over 50 runs.

The robot, instead, was equipped with a range sensor that could detect closed doors as it came within a unit distance from them. To achieve each specification, the robot's strategy was iterative planning (Maly et al. 2013). The robot first computed a trajectory with minimum distance to satisfaction using the current knowledge of the environment. During the execution of the trajectory, if the robot encountered a closed door, it updated its knowledge of the environment, and replanned with the new information. Samples of the trajectories for φ_1, φ_2, and φ_3 are shown in Fig. 5.

In these figures, the locations at which the robot sensed the closed doors are indicated by a burgundy rectangle with a gray arc which represent the robot and its sensor readings, respectively. After the discovery of each closed door, the robot applied a braking operation to avoid a collision with the door. Replanning was performed after the robot came to a complete stop. We note that the variability and curviness of the computed trajectories are due to the dynamics of the robot and the use of sampling-based motion planners.

For Specification 1, the robot's initial plan was fully satisfying. During the execution of this plan, the robot discovered that the bottom door to the office containing p_0 was closed as shown in Fig. 5a. At that point, the robot performed a braking operation and computed another fully satisfying plan. This plan instructed the robot to visit p_0 through the top door. As the robot was following these instructions, it discovered the top door was also closed, making p_0 unaccessible. A new plan with distance to satisfaction of 1 was generated at the spot, which required the robot to skip p_0 and visit $p_1, \ldots, p_4$. During the execution of this plan, the robot discovered the door to p_1 to be also closed. At this point, the robot obtained the full knowledge of the environment and generated a plan, following which the robot visited all the accessible propositions in the correct order. The final plan had a distance to satisfaction of 4.

The initial motion plan that the robot computed for Specification 2 was also fully satisfying. After discovering that the bottom door to p_0 was closed, the robot generated a motion plan to satisfy φ_2 by visiting p_1 and then p_3. Once p_1 was found unaccessible, another fully satisfying plan was computed requiring a visit to p_0 through the top entrance followed by a visit to p_4. During the execution of this plan, the robot discovered the top door was also closed. At this point, no other fully satisfying plans existed. Thus, the robot computed a trajectory that skipped p_0 and visited p_4 with the distance to satisfaction of 1 as shown in Fig. 5b.

Similarly, the robot's initial plan for Specification 3 was also fully satisfying. It required the robot to visit p_2 by going through the office containing p_0. However, during the execution of this plan, the robot discovered the closed door that made the office unaccessible from the bottom corridor (see Fig. 5c). At this point, the specification could not be fully satisfied. The robot computed a new plan with minimum distance to satisfaction of 1, which visited p_2 by the violation of passing over p_4 and then visiting p_0. During the execution of this plan, the robot discovered the closed door to p_1. The robot replanned a new trajectory which followed a similar path as the previous one. After visiting p_2 and while on its way to p_0, the robot discovered the top door to p_0 was also closed. The new plan for the robot was to skip p_0 and terminate at this point. The traveled trajectory of the robot violated p_4 and p_0 and had the distance to satisfaction of 2.

For each specification, 50 motion plans were computed. The planning data is shown in Table 1. Note that the total planning times are larger for case study 2 than those for case study 1. That is because the robot had to perform (re)planning at every instance of discovery of an obstacle (4 instances) in case study 2, whereas in case study 1 the planner was called only once.

7 Conclusion

By deploying methods for *quantification of satisfaction* of an LTL specification, we have constructed a planning framework that can handle complicated specifications that may not be satisfiable by the robot in the environment. Our framework is designed to require from the designer only a co-safe LTL formula along with the cost function over the proposition. Indeed, this framework can be easily modified so that the designer can define her own weighted skipping function or even the weighted automaton representing the specification and weighted skipping. This allows to fine-tune

WDFA to the specific needs of the user. Furthermore, in this work, we did not allow the proposition costs nor the status of the obstacles to change over time. An interesting scenario to consider, for instance, is the case that cost/importance of propositions drop as time passes and doors alternate between open/close. These challenges require a more sophisticated method of quantification and are a prospect of future work.

8 Acknowledgments

The authors would like to thank Professor Orna Kupferman from Hebrew University for her valuable comments. Work on this paper by the authors has been supported by NSF grants 1317849, 1139011, and 1018798 and BSF grant 9800096.

References

Almagor, S.; Boker, U.; and Kupferman, O. 2013. Formalizing and reasoning about quality. In *Automata, Languages, and Programming*. Springer. 15–27.

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Ann. of Math. and Artif. Intell.* 22:5–27.

Baier, J. A.; Fritz, C.; Bienvenu, M.; and McIlraith, S. 2008. Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In *AAAI Conference on Artificial Intelligence*, 1509–1512.

Bhatia, A.; Maly, M.; Kavraki, L. E.; and Vardi, M. Y. 2011. Motion planning with complex goals. *Robotics Automation Magazine, IEEE* 18(3):55–64.

Bhatia, A.; Kavraki, L. E.; and Vardi, M. Y. 2010a. Motion planning with hybrid dynamics and temporal goals. In *Conf. on Decision and Control*, 1108–1115. IEEE.

Bhatia, A.; Kavraki, L. E.; and Vardi, M. Y. 2010b. Sampling-based motion planning with temporal goals. In *Int. Conf. on Robotics and Automation*, 2689–2696. IEEE.

Boutilier, C.; Brafman, R. I.; Domshlak, C.; Hoos, H. H.; and Poole, D. 2004. Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *J. Artif. Intell. Res.(JAIR)* 21:135–191.

Calvanese, D.; Giacomo, G. D.; and Vardi, M. 2002. Reasoning about actions and planning in LTL action theories. In *Int. Conf. on the Principles of Knowledge Representation and Reasoning*, 593–602. Morgan Kaufmann.

Černỳ, P.; Henzinger, T. A.; and Radhakrishna, A. 2012. Simulation distances. *Theo. Comp. Science* 413(1):21–35.

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

Coles, A. J., and Coles, A. 2011. LPRPG-P: Relaxed plan heuristics for planning with preferences. In *Int. Conf. on Automated Planning and Scheduling*, 26–33.

Şucan, I. A.; Moll, M.; and Kavraki, L. E. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine* 19(4):72–82.

De Giacomo, G., and Vardi, M. 1999. Automata-theoretic approach to planning for temporally extended goals. In *Proc. European Conf. on Planning*, Lecture Notes in AI 1809, 226–238. Springer.

DeCastro, J. A., and Kress-Gazit, H. 2013. Guaranteeing reactive high-level behaviors for robots with complex dynamics. In *Int. Conf. on Intell. Rob. and Sys.*, 749–756. IEEE.

Dornhege, C.; Gissler, M.; Teschner, M.; and Nebel, B. 2009. Integrating symbolic and geometric planning for mobile manipulation. In *Safety, Security & Rescue Robotics, Int. Workshop on*, 1–6. IEEE.

Gerevini, A. E.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. *Artif. Intell.* 173(5):619–668.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning: theory & practice*. Elsevier.

Juma, F.; Hsu, E. I.; and McIlraith, S. A. 2012. Preference-based planning via MaxSAT. In *Advances in Artificial Intelligence*. Springer. 109–120.

Kim, K., and Fainekos, G. 2013. Minimal specification revision for weighted transition systems. In *Int. Conf. on Robotics and Automation*, 4068–4074. IEEE.

Kim, K., and Fainekos, G. 2014. Revision of specification automata under quantitative preferences. In *Int. Conf. on Robotics and Automation*, 5339–5344. IEEE.

Kloetzer, M., and Belta, C. 2008. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Trans. on Auto. Control* 53(1):287–297.

Kress-Gazit, H.; Fainekos, G.; and Pappas, G. J. 2007. Where's waldo? sensor-based temporal logic motion planning. In *Int. Conf. on Robotics and Auto.*, 3116–3121. IEEE.

Kupferman, O., and Vardi, M. Y. 2001. Model checking of safety properties. *Formal Methods in System Design* 19:291–314.

Kvarnström, J., and Doherty, P. 2000. Talplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence* 30(1-4):119–169.

Maly, M. R.; Lahijanian, M.; Kavraki, L. E.; Kress-Gazit, H.; and Vardi, M. Y. 2013. Iterative temporal motion planning for hybrid systems in partially unknown environments. In *Int. Conf. on Hybrid Systems: Computation and Control*, 353–362. ACM.

Sohrabi, S.; Baier, J. A.; and McIlraith, S. A. 2009. HTN planning with preferences. In *Int. Joint Conf. on Artificial Intelligence*, 1790–1797.

Srivastava, S.; Riano, L.; Russell, S.; and Abbeel, P. 2013. Using classical planners for tasks with continuous operators in robotics. In *Intl. Conf. on Automated Planning and Scheduling*, 27–35.

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

Wongpiromsarn, T.; Topcu, U.; and Murray, R. M. 2010. Receding horizon control for temporal logic specifications. In *Int. Conf. on Hybrid Systems: Computation and Control*, 101–110. ACM.