Optimal Control of Non-deterministic Systems for a Computationally Efficient Fragment of Temporal Logic

Eric M. Wolff, Ufuk Topcu, and Richard M. Murray

***Abstract*— We develop a framework for optimal control policy synthesis for non-deterministic transition systems subject to temporal logic specifications. We use a fragment of temporal logic to specify tasks such as safe navigation, response to the environment, persistence, and surveillance. By restricting specifications to this fragment, we avoid a potentially doubly-exponential automaton construction. We compute feasible control policies for non-deterministic transition systems in time polynomial in the size of the system and specification. We also compute optimal control policies for average, minimax (bottleneck), and average cost-per-task-cycle cost functions. We highlight several interesting cases when these can be computed in time polynomial in the size of the system and specification. Additionally, we make connections between computing optimal control policies for an average cost-per-task-cycle cost function and the generalized traveling salesman problem. We give simulation results for motion planning problems.**

I. Introduction

The responsibilities given to robots, autonomous vehicles, and other cyberphysical systems continue to increase faster than our ability to reason about the correctness of their behavior. In safety-critical applications like autonomous driving and air traffic management, it is desirable to unambiguously specify the desired system behavior and automatically synthesize a controller that provably implements this behavior. Additionally, many applications involve non-determinism (in the system and environment) and demand efficient (not just feasible) controllers.

Linear temporal logic (LTL) is an expressive task-specification language for specifying a variety of tasks, such as safety (always avoid B), response (if A, then B), persistence (eventually always stay in A), and recurrence (infinitely often visit A). Given a finite abstraction of a dynamical system (see [1]–[7]) and an LTL specification, feasible control policies can be automatically created for non-deterministic systems [8]. Unfortunately, the complexity of synthesizing a control policy that ensures that a non-deterministic system satisfies an LTL formula is doubly-exponential in the formula length [9].

The poor worst-case complexity of controller synthesis for LTL has led to the development of fragments of LTL that are both useful and computationally efficient to reason about [10]–[15]. Notably, generalized reactivity(1) (GR(1)) [11] is an efficient fragment for synthesizing feasible control policies for non-deterministic systems. A GR(1) formula is of the form: environment assumptions *imply* system guarantees. This framework has been used to create reactive control protocols for robots and autonomous vehicles [7], [16]. The generalized Rabin(1) fragment subsumes GR(1) and is the largest fragment of LTL that can be solved in time polynomial in the size of the system and specification [13].

We use the fragment of temporal logic introduced by the authors in [15]. This fragment includes the same system guarantees in generalized Rabin(1), i.e., all system guarantees in GR(1) and persistence. However, we limit the environmental liveness assumptions. In this paper, we show that for this temporal logic fragment, one can compute correct and optimal control policies for non-deterministic transitions systems subject to a variety of relevant cost functions.

Prior work has focused on finding feasible control policies for systems subject to LTL specifications. For deterministic systems, control policies can be computed that optimize minimax (or bottleneck) [17], weighted average [18], and cost-per-task-cycle [12] cost functions. Chatterjee et al. [19] compute optimal policies for non-deterministic systems subject to parity constraints and an average cost function. Jing and Kress-Gazit [20] give an incomplete method for non-deterministic systems in the GR(1) framework.

Our main contributions are methods for optimal control policy synthesis for non-deterministic transition systems subject to constraints from an expressive temporal logic fragment. We compute control policies that are optimal for average, minimax (bottleneck), and average cost-per-task-cycle cost functions. Due to the non-determinism (in the system and environment), we minimize the worst-case for each cost function. As our techniques do not require the so-called property automaton construction, we avoid the potentially doubly-exponential blow-up in the length of the specification. Additionally, we make close connections with control policy synthesis and dynamic programming.

We extend a special case of the minimax (bottleneck) cost function considered in [17] to non-deterministic systems. We get an exponential time improvement for deterministic systems, as we do not require a Büchi automaton construction for the LTL formula. For an average cost-per-task-cycle cost function (see [12]), we show that finding an optimal control policy is equivalent to solving a generalized traveling salesman problem, which is NP-hard, but for which many efficient heuristic algorithms exist. Finally, we improve on our previous results [15] for computing feasible control policies.

Eric M. Wolff and Richard M. Murray are with the Department of Control and Dynamical Systems, California Institute of Technology, Pasadena, CA. Ufuk Topcu is with the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA. The corresponding author is ewolff@caltech.edu

II. PRELIMINARIES

In this section we give background on the system model and specification language that we consider. The notation follows that used in the authors' previous work [15].

An *atomic proposition* is a statement that is either $True$ or $False$. A *propositional formula* is composed of only atomic propositions and propositional connectives, i.e., $\wedge$ (and) and $\neg$ (not). The cardinality of a set X is denoted by $|X|$.

A. System model

We use non-deterministic finite transition systems to model the system behavior.

Definition 1. A *non-deterministic (finite) transition system* (NTS) is a tuple $\mathcal{T} = (S, A, R, s_0, AP, L, c)$ consisting of a finite set of states S, a finite set of actions A, a transition function $R : S \times A \to 2^S$, an initial state $s_0 \in S$, a set of atomic propositions AP, a labeling function $L : S \to 2^{AP}$, and a non-negative cost function $c : S \times A \times S \to \mathbb{R}$.

Let $A(s)$ denote the set of available actions at state s. Denote the parents of the states in the set $S' \subseteq S$ by $\mathit{Parents}(S') := \{s \in S \mid \exists a \in A(s) \text{ and } R(s,a) \cap S' \neq \varnothing\}$. The set $\mathit{Parents}(S')$ includes all states in S that can (possibly) reach S' in a single transition. We assume that the transition system is non-blocking, i.e., $|R(s,a)| \geq 1$ for each state $s \in S$ and action $a \in A(s)$. A *deterministic transition system* (DTS) is a non-deterministic transition system where $|R(s,a)| = 1$ for each state $s \in S$ and action $a \in A(s)$.

A *memoryless control policy* for a non-deterministic transition system $\mathcal{T}$ is a map $\mu : S \to A$, where $\mu(s) \in A(s)$ for state $s \in S$. A *finite-memory control policy* is a map $\mu : S \times M \to A \times M$ where the finite set M is called the memory and $\mu(s,m) \in A(s) \times M$ for state $s \in S$ and mode $m \in M$. An *infinite-memory* control policy is a map $\mu : S^+ \to A$, where S^+ is a finite sequence of states ending in state s and $\mu(s) \in A(s)$.

Given a state $s \in S$ and action $a \in A(s)$, there may be multiple possible successor states in the set $R(s,a)$, i.e., $|R(s,a)| > 1$. A single successor state $t \in R(s,a)$ is non-deterministically selected. We interpret this selection (or action) as an uncontrolled, adversarial environment resolving the non-determinism.

A *run* $\sigma = s_0 s_1 s_2 \ldots$ of $\mathcal{T}$ is an infinite sequence of its states, where $s_i \in S$ is the state of the system at index i (also denoted σ_i) and for each $i = 0, 1, \ldots$, there exists $a \in A(s_i)$ such that $s_{i+1} \in R(s_i, a)$. A *word* is an infinite sequence of labels $L(\sigma) = L(s_0)L(s_1)L(s_2)\ldots$ where $\sigma = s_0 s_1 s_2 \ldots$ is a run. The set of runs of $\mathcal{T}$ with initial state $s \in S$ induced by a control policy μ is denoted by $\mathcal{T}^\mu(s)$.

Connections to graph theory: We will often consider a non-deterministic transition system as a graph with the natural bijection between the states and transitions of the transition system and the vertices and edges of the graph. Let $G = (S, R)$ be a directed graph (digraph) with vertices S and edges R. There is an edge e from vertex s to vertex t if and only if $t \in R(s,a)$ for some $a \in A(s)$. A digraph $G = (S, R)$ is *strongly connected* if there exists a path between each pair of vertices $s, t \in S$ no matter how the non-determinism is resolved. A digraph $G' = (S', R')$ is a *subgraph* of $G = (S, R)$ if $S' \subseteq S$ and $R' \subseteq R$. The subgraph of G restricted to states $S' \subseteq S$ is denoted by $G|_{S'}$. A digraph $G' \subseteq G$ is a *strongly connected component* if it is a maximal strongly connected subgraph of G.

B. A fragment of temporal logic

We use the fragment of temporal logic introduced in [15] to specify tasks such as safe navigation, immediate response to the environment, persistent coverage, and surveillance. For a propositional formula φ, the notation $\Box\varphi$ means that φ is always true, $\Diamond\varphi$ means that φ is eventually true, $\Box\Diamond\varphi$ means that φ is true infinitely often, and $\Diamond\Box\varphi$ means that φ is eventually always true [21].

Syntax: We consider formulas of the form

$$\varphi = \varphi_{\text{safe}} \wedge \varphi_{\text{resp}} \wedge \varphi_{\text{per}} \wedge \varphi_{\text{task}} \wedge \varphi^{\text{ss}}_{\text{resp}}, \tag{1}$$

where

$$\begin{aligned}
\varphi_{\text{safe}} &:= \Box\psi_1, \\
\varphi_{\text{resp}} &:= \bigwedge_{j \in I_2} \Box(\psi_{2,j} \implies \bigcirc\phi_{2,j}), \\
\varphi_{\text{per}} &:= \Diamond\Box\,\psi_3, \\
\varphi_{\text{task}} &:= \bigwedge_{j \in I_4} \Box\Diamond\,\psi_{4,j}, \\
\varphi^{\text{ss}}_{\text{resp}} &:= \bigwedge_{j \in I_5} \Diamond\Box\,(\psi_{5,j} \implies \bigcirc\phi_{5,j}).
\end{aligned}$$

Note that $\Box\psi_1 = \Box\bigwedge_{j\in I_1}\psi_{1,j} = \bigwedge_{j\in I_1}\Box\psi_{1,j}$ and $\Diamond\Box\,\psi_3 = \Diamond\Box\bigwedge_{j\in I_3}\psi_{3,j} = \bigwedge_{j\in I_3}\Diamond\Box\,\psi_{3,j}$. In the above definitions, $I_1, \ldots, I_5$ are finite index sets and $\psi_{i,j}$ and $\phi_{i,j}$ are propositional formulas for any i and j.

We refer to each $\psi_{4,j}$ in φ_{task} as a recurrent *task*.

Remark 1. Guarantee and obligation, i.e., $\Diamond\psi$ and $\Box(\psi \implies \Diamond\phi)$ respectively, are not included in (1). Neither are disjunctions of formulas of the form (1). This fragment of LTL is incomparable to other commonly used temporal logics, such as computational tree logic and GR(1). See Wolff et al. [15] for details.

Remark 2. Our results easily extend to include a fixed order for some or all of the tasks in φ_{task}, as well as ordered tasks with different state constraints between the tasks.

Semantics: We use set operations between a run σ of $\mathcal{T} = (S, A, R, s_0, AP, L, c)$ and subsets of S where particular propositional formulas hold to define satisfaction of a temporal logic formula [22]. We denote the set of states where propositional formula ψ holds by $[\![\psi]\!]$. A run σ *satisfies* the temporal logic formula φ, denoted by $\sigma \vDash \varphi$, if and only if certain set operations hold.

Let σ be a run of the system $\mathcal{T}$, $\mathit{Inf}(\sigma)$ denote the set of states visited infinitely often in σ, and $\mathit{Vis}(\sigma)$ denote the set of states visited at least once in σ. Given propositional formulas ψ and ϕ, we relate satisfaction of a temporal logic formula of the form (1) with set operations as follows:

- $\sigma \vDash \Box\psi$ iff $\mathit{Vis}(\sigma) \subseteq [\![\psi]\!]$,

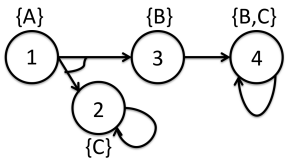

Fig. 1. Example of a non-deterministic transition system

- $\sigma \models \Diamond \Box \psi$ iff $\mathit{Inf}(\sigma) \subseteq [\![\psi]\!]$,
- $\sigma \models \Box \Diamond \psi$ iff $\mathit{Inf}(\sigma) \cap [\![\psi]\!] \neq \varnothing$,
- $\sigma \models \Box(\psi \implies \bigcirc\phi)$ iff $\sigma_i \notin [\![\psi]\!]$ or $\sigma_{i+1} \in [\![\phi]\!]$ for all i,
- $\sigma \models \Diamond \Box (\psi \implies \bigcirc\phi)$ iff there exists an index j such that $\sigma_i \notin [\![\psi]\!]$ or $\sigma_{i+1} \in [\![\phi]\!]$ for all $i \geq j$.

A run σ *satisfies* a conjunction of temporal logic formulas $\varphi = \bigwedge_{i=1}^{m} \varphi_i$, denoted by $\sigma \models \varphi$, if and only if the set operations for each temporal logic formula φ_i holds. A system $\mathcal{T}$ under control policy μ *satisfies* the formula φ at state $s \in S$, denoted $\mathcal{T}^{\mu}(s) \models \varphi$ if and only if $\sigma \models \varphi$ for all $\sigma \in \mathcal{T}^{\mu}(s)$. Given a system $\mathcal{T}$, state $s \in S$ is *winning* (with respect to the non-determinism) for φ if there exists a control policy μ such that $\mathcal{T}^{\mu}(s) \models \varphi$. Let $W \subseteq S$ denote the set of winning states.

An example is given in Figure 1. The non-deterministic transition system $\mathcal{T}$ has states $S = \{1,2,3,4\}$; labels $L(1) = \{A\}$, $L(2) = \{C\}$, $L(3) = \{B\}$, $L(4) = \{B,C\}$; a single action called 0; and transitions $R(1,0) = \{2,3\}$, $R(2,0) = \{2\}$, $R(3,0) = \{4\}$, $R(4,0) = \{4\}$. From the acceptance conditions, it follows that $W = \{2,4\}$ for formula $\Box(A \vee C)$, $W = \{2,3,4\}$ for formula $\Box(A \implies \bigcirc B)$, $W = \{1,2,3,4\}$ for formula $\Diamond\Box(A \implies \bigcirc B)$, $W = \{1,2,3,4\}$ for formula $\Box\Diamond C$, and $W = \{3,4\}$ for formula $\Diamond\Box B$. State 4 is winning for all of the above formulas.

III. Problem Statement

We now formally state the two main problems of the paper and give an overview of our solution approach.

Problem 1. Given a non-deterministic transition system $\mathcal{T}$ and a temporal logic formula φ of the form (1), determine whether there exists a control policy μ such that $\mathcal{T}^{\mu}(s_0) \models \varphi$. Return the control policy μ if it exists.

We introduce a general cost function J to distinguish among solutions to Problem 1. Let J map a set of runs $\mathcal{T}^{\mu}(s_0)$ and the corresponding control policy μ to $\mathbb{R} \cup \infty$.

Problem 2. Given a non-deterministic transition system $\mathcal{T}$ and a temporal logic formula φ of the form (1), determine whether there exists an optimal control policy μ^* such that $\mathcal{T}^{\mu^*}(s_0) \models \varphi$ and $J(\mathcal{T}^{\mu^*}(s_0)) \leq J(\mathcal{T}^{\mu}(s_0))$ for all feasible μ. Return the control policy μ^* if it exists.

We begin by defining the value function in Section IV, which is a key component of all later algorithms. Then, we create feasible control policies (i.e., solve Problem 1) in Section V. Although feasible control policies were created in [15], we present a significantly improved algorithm. Then, we introduce average cost-per-task-cycle, minimax (bottleneck), and average cost functions in Sections VI-A, VI-B, and VI-C, respectively. We discuss procedures for computing optimal control policies for these cost functions in Section VI.

Remark 3. The restriction to the fragment in (1) is critical. For the full LTL, even Problem 1 is intractable in general. Determining if there exists such a control policy takes time doubly-exponential in the length of φ [9].

IV. The value function and reachability

We introduce standard dynamic programming notions [23], as applied to non-deterministic systems.

We define *controlled reachability* in a non-deterministic transition system $\mathcal{T}$ with a value function. Let $B \subseteq S$ be a set of states that the controller wants the system to reach. Let the *controlled value function* for system $\mathcal{T}$ and target set B be a map $V^{c}_{B,\mathcal{T}} : S \to \mathbb{R} \cup \infty$, whose value $V^{c}_{B,\mathcal{T}}(s)$ at state $s \in S$ is the minimum (over all possible control policies) cost needed to reach the set B, under the worst-case resolution of the non-determinism. If the value $V^{c}_{B,\mathcal{T}}(s) = \infty$, then the non-determinism can prevent the system from reaching set B from state $s \in S$. For example, consider the system in Figure 1 with unit cost on edges and $B = \{4\}$. Then, $V^{c}_{B}(1) = \infty$, $V^{c}_{B}(2) = \infty$, $V^{c}_{B}(3) = 1$, and $V^{c}_{B}(4) = 0$. The system cannot guarantee reaching set B from states 1 or 2.

The value function satisfies the optimality condition

$$V^{c}_{B,\mathcal{T}}(s) = \min_{a \in A(s)} \max_{t \in R(s,a)} V^{c}_{B,\mathcal{T}}(t) + c(s,a,t),$$

for all $s \in S$.

An optimal control policy μ_B for reaching the set B is memoryless [23] and can be computed at each state $s \in S$ as

$$\mu_B(s) = \arg\min_{a \in A(s)} \max_{t \in R(s,a)} V^{c}_{B,\mathcal{T}}(t) + c(s,a,t).$$

If multiple actions achieve the minimum, select an action in this set with a minimal number of transitions to reach B.

We use the value function to define the *controllable predecessor* set, $\mathit{CPre}^{\infty}_{\mathcal{T}}(B)$, for a given system $\mathcal{T}$ with target set $B \subseteq S$. Let $\mathit{CPre}^{\infty}_{\mathcal{T}}(B) := \{s \in S \mid V^{c}_{B,\mathcal{T}}(s) < \infty\}$ be the set of all states that can reach a state in B for any resolution of the non-determinism.

We define *forced reachability* similarly. Let the *forced value function* for system $\mathcal{T}$ and target set B be a map $V^{f}_{B,\mathcal{T}} : S \to \mathbb{R} \cup \infty$, whose value $V^{f}_{B,\mathcal{T}}(s)$ at state $s \in S$ is the maximum (over all possible control policies) cost of reaching the set B. The forced value function satisfies the optimality condition

$$V^{f}_{B,\mathcal{T}}(s) = \max_{a \in A(s)} \max_{t \in R(s,a)} V^{f}_{B,\mathcal{T}}(t) + c(s,a,t).$$

For a given system $\mathcal{T}$ with target set $B \subseteq S$, the *forced predecessor* set $\mathit{FPre}^{\infty}_{\mathcal{T}}(B) := \{s \in S \mid V^{f}_{B,\mathcal{T}}(s) < \infty\}$, is the set of all states from which no control policy can avoid reaching a state in B.

Remark 4. We consider the case where the controller selects an action, and then the environment selects the next state. Our results easily extend to the case, used in GR(1) [11],

where the environment first resolves the non-determinism (selects an action) and then the controller selects its action.

Remark 5. The value function can be computed for all states in $O(|S|\log|S|+|R|)$ time (see Algorithm 4 in the Appendix). The predecessor sets can be computed in $O(|S|+|R|)$ time since unit costs can be assumed [15].

V. Feasible control policy

We now solve Problem 1 by creating feasible control policies for non-deterministic transition systems that must satisfy a temporal logic formula of the form (1). Algorithm 2 summarizes the main results from [15]. New results here are the addition of the steady-state, next-step response formula $\varphi^{\text{ss}}_{\text{resp}}$ and an improved algorithm for computing the winning set for the recurrent task formula φ_{task}. The $\varphi^{\text{ss}}_{\text{resp}}$ formula is handled in a similar manner to φ_{resp} and is not detailed here.

Effectively, the formulas besides φ_{task} restrict states that can be visited or transitions that can be taken. Safety, φ_{safe}, limits certain states from ever being visited, next-step response, φ_{resp}, limits certain transitions from ever being taken, persistence, φ_{per}, limits certain states from being visited infinitely often, and steady-state, next-step response, $\varphi^{\text{ss}}_{\text{resp}}$, limits certain transitions from being taken infinitely often. The remaining formula is recurrence, φ_{task}, which constrains certain states to be visited infinitely often. One creates the subgraph that enforces all constraints except for φ_{task} and then computes a finite-memory control policy that repeatedly visits all φ_{task} constraints. Then, one relaxes the constraints that only have to be satisfied over infinite time, and computes all states that can reach the states that are part of this policy. A feasible control policy exists for all and only the states in this set. Details are in Algorithm 2.

We give an improved algorithm (compared to that in [15]) for the formula φ_{task} in Algorithm 1. The key insight is that the ordering of tasks does not affect feasibility, which is not the case for optimality (see Section VI).

Proposition 1. *Algorithm 1 computes exactly the winning set for φ_{task}.*

Proof: To satisfy the acceptance condition $Inf(\sigma) \cap [\![\psi_{4,j}]\!] \neq \varnothing$ for all $j \in I_4$, there must exist non-empty sets $F_j \subseteq [\![\psi_{4,j}]\!]$ such that $F_i \subseteq CPre^{\infty}_{\mathcal{T}}(F_j)$ holds for all $i, j \in I_4$, i.e., all tasks are completed infinitely often. Algorithm 1 selects an arbitrary order on the tasks, e.g., $F_{i+1} \subseteq CPre^{\infty}_{\mathcal{T}}(F_i)$ for all $i = 1, 2, \ldots, |I_4| - 1$ and $F_1 \subseteq CPre^{\infty}_{\mathcal{T}}(F_{|I_4|})$, without loss of generality since tasks are completed infinitely often and the graph does not change. Starting with sets $F_j := [\![\psi_{4,j}]\!]$ for all $j \in I_4$, all and only the states in each F_j that do not satisfy the constraints are iteratively removed. At each iteration, at least one state in F_1 is removed or the algorithm terminates. At termination, each F_j is the largest subset of $[\![\psi_{4,j}]\!]$ from which φ_{task} can be satisfied. ∎

The outer while loop runs at most $|F_1|$ iterations. During each iteration, $CPre^{\infty}_{\mathcal{T}}$ is computed $|I_4|$ times, which dominates the time required to compute the set intersections (when using a hash table). Thus, the total complexity of Algorithm 1 is $O(|I_4|F_{\min}(|S|+|R|)$, where $F_{\min} = \min_{j\in I_4}|F_j|$.

Algorithm 1 BUCHI ($\mathcal{T}$, $\{[\![p_{4,j}]\!]$ for $j \in I_4\}$)

Input: NTS $\mathcal{T}$, $F_j := [\![p_{4,j}]\!] \subseteq S$ for $j \in I_4$
Output: Winning set $W \subseteq S$

1: **while** $True$ **do**
2: **for** $i = 1, 2, 3, \ldots, |I_4| - 1$ **do**
3: $F_{i+1} \leftarrow F_{i+1} \cap CPre^{\infty}_{\mathcal{T}}(F_i)$
4: **if** $F_{i+1} = \varnothing$ **then**
5: **return** $W \leftarrow \varnothing$, $F_j \leftarrow \varnothing$ for all $j \in I_4$
6: **end if**
7: **end for**
8: **if** $F_1 \subseteq CPre^{\infty}_{\mathcal{T}}(F_{|I_4|})$ **then**
9: **return** $W \leftarrow CPre^{\infty}_{\mathcal{T}}(F_{|I_4|})$, F_j for all $j \in I_4$
10: **end if**
11: $F_1 \leftarrow F_1 \cap CPre^{\infty}_{\mathcal{T}}(F_{|I_4|})$
12: **end while**

Remark 6. Formula φ_{task} is can be treated for deterministic transition systems more efficiently by computing strongly connected components [24] or performing nested depth-first search [25]. These $O(|S| + |R|)$ procedures replace Algorithm 1.

We now detail feasible control policy synthesis in Algorithm 2. Compute the set W of states that are winning for φ (lines 1-7). If the initial state $s_0 \notin W$, then no control policy exists. If $s_0 \in W$, compute the memoryless control policy $\mu_{S_{\mathcal{A}}}$ which reaches the set $S_{\mathcal{A}}$. Use $\mu_{S_{\mathcal{A}}}$ until a state in $S_{\mathcal{A}}$ is reached. Then, compute the memoryless control policies μ_j induced from $V^{c}_{F_j,\mathcal{T}_{\text{safe}}}$ for all $j \in I_4$ (line 11, also see Algorithm 1). The finite-memory control policy μ is defined as follows by switching between memoryless policies depending on the current task. Let $j \in I_4$ denote the current task set F_j. The system uses μ_j until a state in F_j is visited. Then, the system updates its task to $k = (j + 1, \mod |I_4|) + 1$ and uses control policy μ_k until a state in F_k is visited, and so on. The total complexity of the algorithm is $O((|I_2| + |I_5| + |I_4||F_{\min}|)(|S| + |R|))$, which is polynomial in the size of the system and specification.

VI. Optimal control policy

We now solve Problem 2 by computing control policies for non-deterministic systems that satisfy a temporal logic formula of the form (1) and also minimize a cost function. We consider average cost-per-task-cycle, minimax, and average cost functions. The last two admit polynomial time solutions for deterministic systems and restricted non-deterministic systems. However, we begin with the average cost-per-task-cycle cost function as it is quite natural.

The values these cost functions take are independent of any finite sequence of states. Thus, we optimize the infinite behavior of the system, which corresponds to completing tasks specified by φ_{task} on a subgraph of $\mathcal{T}$ as constructed in Section V. We assume that we are given $\mathcal{T}^{\text{ss}}_{\text{resp}}$ (denoted hereafter by $\mathcal{T}_{\text{inf}}$) and the task sets $F_j \subseteq S$ returned by

Algorithm 2 Overview: Feasible synthesis for NTS

Input: Non-deterministic system $\mathcal{T}$ and formula φ
Output: Control policy μ

Compute $\mathcal{T}_{\text{resp}}$ on $\mathcal{T}$
$\mathcal{T}_{\text{safe}} \leftarrow \mathcal{T}_{\text{resp}}|_{S-FPre^{\infty}(S-[\![\psi_1]\!])}$
$\mathcal{T}_{\text{per}} \leftarrow \mathcal{T}_{\text{safe}}|_{S-FPre^{\infty}(S-[\![\psi_3]\!])}$
Compute $\mathcal{T}^{\text{ss}}_{\text{resp}}$ on $\mathcal{T}_{\text{per}}$
$\Psi := \{[\![\psi_{4,j}]\!] \text{ for all } j \in I_4\}$
$S_{\mathcal{A}}, F := \{F_1, \ldots, F_{|I_4|}\} \leftarrow \text{BUCHI}(\mathcal{T}^{\text{ss}}_{\text{resp}}, \Psi)$
$W := CPre^{\infty}_{\mathcal{T}_{\text{safe}}}(S_{\mathcal{A}})$
if $s_0 \notin W$ **then**
  return "no control policy exists"
end if
$\mu_{S_{\mathcal{A}}} \leftarrow$ control policy induced by $V^c_{S_{\mathcal{A}},\mathcal{T}_{\text{safe}}}$
$\mu_j \leftarrow$ control policy induced by $V^c_{F_j,\mathcal{T}_{\text{safe}}}$ for all $j \in I_4$
return Control policies $\mu_{S_{\mathcal{A}}}$ and μ_j for all $j \in I_4$

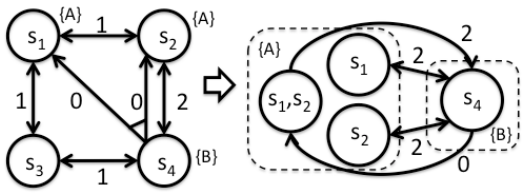

Fig. 2. A non-deterministic transition system and its task graph.

Algorithm 1 (see Algorithm 2). Note that $\mathcal{T}_{\text{inf}}$ is the largest subgraph of $\mathcal{T}$ where all constraints from φ_{safe}, φ_{resp}, φ_{per}, and $\varphi^{\text{ss}}_{\text{resp}}$ hold. Each F_j is the largest set of states for the jth task that are part of a feasible control policy. The problem is now to compute a feasible (winning) control policy that also minimizes the relevant cost function.

Since only the recurrent tasks in φ_{task} on $\mathcal{T}_{\text{inf}}$ will matter for optimization, we construct a new graph that encodes the cost of moving between all tasks. We construct the *task graph* $G' = (V', E')$ which encodes the cost of optimal control policies between all tasks in φ_{task} (see Figure 2). Let V' be partitioned as $V' = \bigcup_{j \in I_4} V'_j$, where $V'_i \cap V'_j = \varnothing$ for all $i \neq j$. Let $F_j \subseteq S$ denote the set of states that correspond to the jth task in φ_{task}, as returned from Algorithm 1. Create a state $v \in V'_j$ for each of the $2^{|F_j|} - 1$ non-empty subsets of F_j that are reachable from the initial state. Define the map $\tau : V' \to 2^S$ from each state in V' to subsets of states in S. For each state $v \in V'$, compute the controlled value function $V^c_{\tau(v),\mathcal{T}_{\text{inf}}}$ on $\mathcal{T}_{\text{inf}}$. For all states $u \in V'_i$ and $v \in V'_j$ where $i \neq j$, define an edge $e_{uv} \in E'$. Assign a cost to edge e_{uv} as $c_{uv} := \max_{s \in \tau(u)} V^c_{\tau(v),\mathcal{T}_{\text{inf}}}(s)$. The cost c_{uv} is the maximum worst-case cost of reaching a state $t \in \tau(v)$ from a state $s \in \tau(u)$, when using an optimal control policy.

It is necessary to consider all subsets of states, as the cost of reaching each subset may differ due to the non-determinism. For deterministic systems, one can simply create a state in V'_j for each state in F_j. This is because the cost of all subsets of F_j can be determined by the costs to reach the individual states in F_j.

Remark 7. It may not be necessary to create the entire task graph at once. For example, one can create a task graph with $|I_4|$ states where each state corresponds to the set F_j. This gives a control policy that leads to an upper bound on the cost of an optimal policy. Additionally, by defining edges in the task graph as the minimum worst-case cost $\min_{s \in \tau(u)} V^c_{\tau(v),\mathcal{T}_{\text{inf}}}(s)$ between tasks, one can compute a lower bound on the cost of an optimal policy. One can use the current control policy and improve performance in an anytime manner by adding more states to the subgraph corresponding to subsets of each F_j.

Algorithm 3 Overview: Optimal synthesis for NTS

Input: NTS $\mathcal{T}$, formula φ, cost function J
Output: Optimal control policy μ^*

Compute $\mathcal{T}^{\text{ss}}_{\text{resp}}$, $S_{\mathcal{A}}$, and F_j for all $j \in I_4$ (see Alg. 2)
Compute $F^*_j \subseteq F_j$ for all $j \in I_4$ and optimal task order
$\mu^*_{F^*} \leftarrow$ control policy from $V^c_{F^*,\mathcal{T}_{\text{safe}}}$ where $F^* = \cup_{j \in I_4} F^*_j$
$\mu^*_j \leftarrow$ control policy from $V^c_{F^*_j,\mathcal{T}_{\text{safe}}}$ for all $j \in I_4$
return $\mu^*_{F^*}$, μ^*_j for all $j \in I_4$ and optimal task order

A. Average cost-per-task-cycle

Recall that for $\varphi_{\text{task}} = \bigwedge_{j \in I_4} \square \diamond \psi_{4,j}$, the propositional formula $\psi_{4,j}$ is the jth *task*. A run σ of system $\mathcal{T}$ *completes* the jth task at time t if and only if $\sigma_t \in [\![\psi_{4,j}]\!]$. A *task cycle* is a sequence of states that completes each task at least once, i.e., it intersects $[\![\psi_j]\!]$ for each $j = 1, \ldots, m$ at least once. Similarly to [12], we minimize the average cost-per-task-cycle, or equivalently the maximum cost of a task cycle in the limit. For a deterministic system, this corresponds to finding a cycle of minimal cost that completes every task.

We define the cost function over a run σ. Let σ be a run of $\mathcal{T}$ under control policy μ, $\mu(\sigma)$ be the corresponding control input sequence, and $I_{TC}(t) = 1$ indicate that the system completes a task cycle at time t and $I_{TC}(t) = 0$ otherwise. The *average cost per task cycle* of run σ is

$$J'_{TC}(\sigma, \mu(\sigma)) := \limsup_{n \to \infty} \frac{\sum_{t=0}^{n} c(\sigma_t, \mu(\sigma_t), \sigma_{t+1})}{\sum_{t=0}^{n} I_{TC}(t)},$$

which maps runs and control inputs of $\mathcal{T}$ to $\mathbb{R} \cup \infty$. This map is well-defined when (i) $c(\sigma_t, \mu(\sigma_t), \sigma_{t+1})$ is bounded for all $t \geq 0$, and (ii) there exists a $t' \in \mathbb{N}$ such that $I_{TC}(t) = 1$ for infinitely many $t \geq t'$. We assume that (i) is true in the sequel and note that (ii) holds for every run that satisfies a formula φ with at least one task. If there are no tasks in φ, one can add the task $\square \diamond \mathit{True}$ so that $I_{TC}(t) = 1$ at every time instance (see Section VI-C).

We define the average per-task-cycle cost function

$$J_{TC}(\mathcal{T}^{\mu}(s)) := \max_{\sigma \in \mathcal{T}^{\mu}(s)} J'_{TC}(\sigma, \mu(\sigma)) \tag{2}$$

over the set of runs of system $\mathcal{T}$ starting from initial state s under control policy μ. The cost function (2) does not depend on any finite behavior of the system, intuitively because any short-term costs are averaged out in the limit.

We next show that Problem 2 with cost function J_{TC} is at least as hard as the NP-hard generalized traveling salesman problem [26].

Generalized traveling salesman problem [26]: Let $G = (V, E)$ be a digraph with vertices V, edges E, and a non-negative cost c_{ij} on each edge $(i, j) \in E$. Set V is the disjoint union of p vertex sets, i.e., $V = V_1 \cup \ldots \cup V_p$, where $V_i \cap V_j = \varnothing$ for all $i \neq j$. There are no edges between states in the same vertex set. The *generalized traveling salesman problem*, GTSP = $\langle (V, E), c \rangle$, is to find a minimum cost cycle that includes a single state from each V_i for all $i = 1, \ldots, p$.

Theorem 1. *Any instance of the generalized traveling salesman problem can be formulated as an equivalent instance of Problem 2 with the cost function J_{TC}.*

Proof: The proof is by construction. Given an instance of the GTSP $\langle (V, E), c \rangle$, we solve Problem 2 on a deterministic transition system $\mathcal{T} = (S, A, R, s_0, AP, L, c)$ and formula φ. Let $S = V \cup \{s_0\}$. Define the transitions as $R(u, a_v) = v$, with action $a_v \in A(u)$, and costs $c(u, a_v, v) = c_{uv}$ for each edge $e_{uv} \in E$. Label all states in vertex set V_i with atomic proposition L_i and let $\varphi = \bigwedge_{i \in p} \square \diamond L_i$. Finally, add transitions from s_0 to every other state $s \in S$. Although Problem 2 does not require that each task is only completed once per cycle, an optimal solution always exists which completes each task once per cycle. ∎

Recall that I_4 is the index set of all recurrent tasks. We can fix an arbitrary task ordering, denoted $I_4 = 1, \ldots, |I_4|$ (with some abuse of notation), without loss of generality for feasible control policies. However, the order that we visit tasks matters for optimal control policies. Additionally, we can select this task order ahead of time or update it during execution. We now (potentially conservatively) assume that we will select this task order ahead of time. This assumption is not necessary in Sections VI-B or VI-C.

Assumption 1. A control policy attempts to complete tasks in a fixed order.

We will optimize the task order over all permutations of fixed task orders. This optimization is a generalized traveling salesman problem on the task graph. While this is an NP-hard problem, practical methods exist for computing exact and approximate solutions [26]. Once the optimal ordering of tasks is computed, the finite-memory control policy switches between these tasks in a similar manner described in Section V.

B. Minimax (bottleneck) costs

We now consider a minimax (bottleneck) cost function, minimizes the maximum accumulated cost between completion of tasks. The notation loosely follows [17] which considers a generalization of this cost function for deterministic transition systems with LTL. Let $\mathbb{T}_{\text{task}}(\sigma, i)$ be the accumulated cost at the ith completion of a task in φ_{task} along a run σ. The *minimax* cost of run σ is

$$J'_{\text{bot}}(\sigma, \mu(\sigma)) := \limsup_{i \to \infty} (\mathbb{T}_{\text{task}}(i+1) - \mathbb{T}_{\text{task}}(i)), \quad (3)$$

which maps runs and control inputs of $\mathcal{T}$ to $\mathbb{R} \cup \infty$.

Define the worst-case minimax cost function as

$$J_{\text{bot}}(\mathcal{T}^\mu(s)) := \max_{\sigma \in \mathcal{T}^\mu(s)} J'_{\text{bot}}(\sigma, \mu(\sigma)) \quad (4)$$

over the set of runs of system $\mathcal{T}$ starting from initial state s under control policy μ.

We now solve Problem 2 for the cost function J_{bot}. First, compute the task graph as in Section VI. The edges in the task graph correspond to the maximum cost accumulated between completion of tasks, assuming that the system uses an optimal strategy. Thus, a minimal value of J_{bot} can be found by minimizing the maximum edge in the task graph, subject to the constraint that a vertex corresponding to each task can be reached. Select an estimate of J_{bot} and remove all edges in the task graph that are greater than this value. If there exists a strongly connected component of the task graph that contains a state corresponding to each task and is reachable from the initial state, then we have an upper bound on J_{bot}. If not, we have a lower bound. This observation leads to a simple procedure where one selects an estimate of J_{bot} as the median of edge costs that satisfy the previously computed bounds, removes all edges with costs greater than this estimate, determines if the subgraph is strongly connected (with respect to the tasks), and then updates the bounds. Each iteration requires the computation of strongly connected components and the median of edge costs, which can be done in linear time [24]. It is easy to see that this procedure terminates with the correct value of J_{bot} in $O(\log|E'|)$ iterations. Thus, the total complexity is $O(\log|E'|(|V'| + |E'|))$, giving a polynomial time algorithm for deterministic transition systems and non-deterministic transition systems with a single state per task, i.e., $|F_j| = 1$ for all $j \in I_4$.

C. Average costs

The *average* cost of run σ is

$$J'_{\text{avg}}(\sigma, \mu(\sigma)) := \limsup_{n \to \infty} \frac{\sum_{t=0}^{n} c(\sigma_t, \mu(\sigma_t), \sigma_{t+1})}{n},$$

which maps runs and control inputs of $\mathcal{T}$ to $\mathbb{R} \cup \infty$.

We now define the worst-case average cost function,

$$J_{\text{avg}}(\mathcal{T}^\mu(s)) := \sup_{\sigma \in \mathcal{T}^\mu(s_0)} J'_{\text{avg}}(\sigma, \mu(\sigma)) \quad (5)$$

over the set of runs of system $\mathcal{T}$ starting from initial state s under control policy μ.

We now solve Problem 2 for the cost function J_{avg}. Note that this cost function corresponds to J_{TC} when $\varphi_{\text{task}} = \square \diamond True$, but without additional tasks.

For non-deterministic transition systems, Problem 2 reduces to solving a mean-payoff parity game on $\mathcal{T}_{\text{inf}}$ [19]. An optimal control policy will typically require infinite memory, as opposed to the finite-memory control policies that we have considered. Such a policy alternates between a feasible control policy and an unconstrained minimum cost control policy, spending an increasing amount of time using the unconstrained minimum cost control policy. Given the subgraph $\mathcal{T}_{\text{inf}}$ and the feasible task sets $F_j \subseteq S$ (see

TABLE I

COMPLEXITY OF FEASIBLE POLICY SYNTHESIS

Language	DTS	NTS
Frag. in (1)	$O(\|\varphi\|(\|S\|+\|R\|))$	$O(\|\varphi\|F_{\text{min}}(\|S\|+\|R\|))$
GR(1)	$O(\|\varphi\|\|S\|\|R\|)$	$O(\|\varphi\|\|S\|\|R\|)$
LTL	$O(2^{(\|\varphi\|)}(\|S\|+\|R\|))$	$O(2^{2^{(\|\varphi\|)}}(\|S\|+\|R\|))$

Algorithm 2), one can compute an optimal control policy using the results of Chatterjee et al. [19]. For deterministic systems, extensions to a more general weighted average cost function can be found in [18].

VII. COMPLEXITY

We summarize our complexity results for feasible control policy synthesis and compare with LTL and GR(1) [11]. We assume that set membership is determined in constant time with a hash function [24]. We denote the length of a temporal logic formula by $|\varphi|$. Let $|\varphi| = |I_2| + |I_4| + |I_5|$ for the fragment in (1), $|\varphi| = mn$ for a GR(1) formula with m assumptions and n guarantees, and $|\varphi|$ be the formula length for LTL [21]. Recall that $F_{\text{min}} = \min_{j \in I_4} |[[p_{4,j}]]|$. For typical motion planning specifications, $F_{\text{min}} \ll |S|$ and $|\varphi|$ is small. We use the non-symbolic complexity results for GR(1) in [11]. Results are summarized in Table I.

We now summarize the complexity of optimal control policy synthesis. The task graph $G' = (V', E')$ has $O(\sum_{i \in I_4} 2^{|F_i|} - 1)$ states and can be computed in $O((\sum_{i \in I_4} 2^{|F_i|} - 1)(|S|\log|S| + |R|))$ time. Computing an optimal control policy for J_{TC} requires solving an NP-hard generalized traveling salesman problem on G'. Computing an optimal control policy for J_{bot} requires $O(\log|E'|(|V'|+|E'|)$ time. An optimal control policy for J_{avg} can be computed in pseudo-polynomial time [19]. For deterministic systems, the task graph has $O(\sum_{i \in I_4} |F_i|)$ states and can be computed in $O((\sum_{i \in I_4} |F_i|)(|S|\log|S| + |R|))$ time. An optimal control policy for J_{avg} can be computed in $O(|S||R|)$ time. Thus, we can compute optimal control policies for deterministic transition systems with cost functions J_{bot} and J_{avg} in time polynomial in the size of the system and specification. Additionally, for non-deterministic transition systems where $|F_j| = 1$ for all $j \in I_4$, we can compute optimal control policies for J_{bot} in time polynomial in the size of the system and specification.

Remark 8. The fragment in (1) is not handled well by standard approaches. Using ltl2ba [27], we created Büchi automaton for formulas of the form φ_{resp}. The automaton size and time to compute it both increased exponentially with the number of conjunctions in φ_{resp}.

VIII. EXAMPLES

The following examples (based on those in [15]) demonstrate the techniques developed in Sections V and VI for tasks motivated by robot motion planning in a planar environment (see Figure 3). All computations were done in Python on a dual-core Linux desktop with 2 GB of memory. All computation times were averaged over five arbitrarily generated problem instances and include construction of the transition system. Due to lack of space, we only consider the average cost-per-task-cycle cost function.

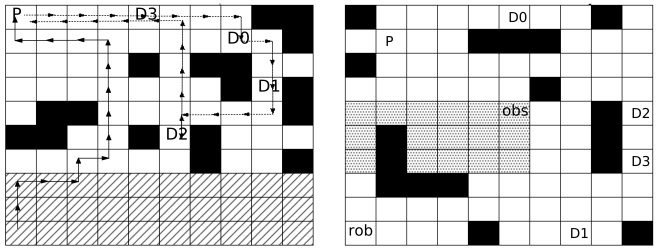

Fig. 3. Left: Diagram of deterministic setup (n = 10). Only white cells are labeled 'stockroom.' Right: Diagram of non-deterministic setup (n = 10). A dynamic obstacle (obs) moves within the shaded region.

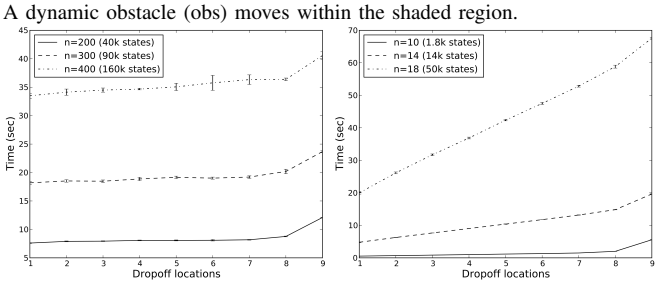

Fig. 4. Control policy synthesis times for deterministic (left) and non-deterministic (right) grids.

A. Deterministic transition system

Consider a gridworld where a robot occupies a single cell at a time and can choose to either remain in its current cell or move to one of four adjacent cells at each step. We consider square grids with static obstacle densities of 20 percent. The robot's task is to eventually remain in the stockroom while repeatedly visiting a pickup location P and multiple dropoff locations D_0, D_1, D_2, D_3. The robot must never collide with a static obstacle. The set of atomic propositions is $\{P, D_0, D_1, D_2, D_3, \text{storeroom}, \text{obs}\}$. This task is formalized by $\varphi = \Diamond\Box\text{stockroom} \wedge \Box\Diamond P \wedge \bigwedge_{j \in I_4} \Box\Diamond D_j \wedge \Box\neg\text{obs}$. In all following results, D_j holds at a single state in the transition system. Results for optimal control policy synthesis are shown in Figure 4 for $n \times n$ grids where $n \in \{200, 300, 400\}$.

B. Non-deterministic transition system

We now consider a similar setup with a dynamically moving obstacle. The state of the system is the product of the robot's location and the obstacle's location, both of which can move as previously described for the robot. The robot selects an action and then the obstacle non-deterministically moves. The robot's task is similar to before and is formalized as $\varphi = \Box\Diamond P \wedge \bigwedge_{j \in I_4} \Box\Diamond D_j \wedge \Box\neg\text{obs}$. Results for optimal control policy synthesis are shown in Figure 4 for $n \times n$ grids where $n \in \{10, 14, 18\}$.

IX. CONCLUSIONS

We have presented a framework for optimal control policy synthesis for non-deterministic transition systems with specifications from a fragment of temporal logic. Our approach is simple and makes explicit connections with dynamic programming through our extensive use of value functions.

Additionally, optimal policies can be computed in polynomial time for certain combinations of cost functions and system restrictions. Future work will investigate incremental computation of the value function, extensions to Markov decision processes, and exploring the underlying automata structure of this fragment of temporal logic.

APPENDIX

Algorithm 4 computes the controlled value function as defined in Section IV. This algorithm should be viewed as a minor extension of Dijkstra's algorithm [24] to non-deterministic transition systems. Similar reasoning applies to the forced value function. Set Q below is a priority queue with standard operations.

Algorithm 4 Value function (controlled)

Input: NTS $\mathcal{T}$, set $B \subseteq S$
Output: The (controlled) value function $V^c_{B,\mathcal{T}}$

```
V^c_{B,T}(s) ← ∞ for all s ∈ S − B
V^c_{B,T}(s) ← 0 for all s ∈ B
Q ← S
while Q ≠ ∅ do
    u ← EXTRACT-MIN(Q)
    if V^c_{B,T}(u) = ∞ then
        return V^c_{B,T}
    for s ∈ Parents(u) ∩ Q do
        tmp ← min_{a∈A(s)} max_{t∈R(s,a)} V^c_{B,T}(t) + c(s,a,t)
        if tmp < V^c_{B,T}(s) then
            V^c_{B,T}(s) ← tmp
            DECREASE-KEY(Q, s, V^c_{B,T}(s))
return V^c_{B,T}
```

Theorem 2. *Algorithm 4 computes the controlled value function for all states in S in $O(|S|log|S| + |R|)$ time.*

Proof: (sketch) The proof follows that of Dijkstra's algorithm (see Ch. 24.3 in [24]), with a slight modification to account for non-determinism. Let $V^*(s)$ denote the optimal cost of reaching set B from state $s \in S$ and $V(s)$ denote the current upper bound. We show that $V(u) = V^*(u)$ whenever a state u is added to $S - Q$, i.e., $u \leftarrow$ EXTRACT-MIN(Q). This is trivially true initially. By contradiction, assume state u is the first state added to $S - Q$ with $V(u) > V^*(u)$. Thus, $V^*(u) < \infty$ and there exists a policy to reach B. Consider a policy from u that reaches a state $y \in Q$ that reaches subset $X \subset S - Q$ in a single action, i.e., there exists $a \in A(y), R(y,a) \subseteq X$. Such a state exists since B is reachable from u. Then, $V(y) = V^*(y)$ since all states in X have optimal costs. This with the non-negativity of edge weights implies that $V(y) = V^*(y) \leq V^*(u) \leq V(u)$. However, $V(u) \leq V(y)$, which is the contradiction. The algorithm uses $O(|S|\log|S|+|R|)$ time since EXTRACT-MIN is called at most $|S|$ times and DECREASE-KEY is called at most $|R|$ times. ∎

ACKNOWLEDGEMENTS

This work was supported by a NDSEG fellowship, the Boeing Corporation, and AFOSR award FA9550-12-1-0302.

REFERENCES

[1] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proc. IEEE*, vol. 88, no. 7, pp. 971–984, 2000.

[2] C. Belta and L. C. G. J. M. Habets, "Controlling of a class of nonlinear systems on rectangles," *IEEE Trans. on Automatic Control*, vol. 51, pp. 1749–1759, 2006.

[3] A. Bhatia, M. R. Maly, L. E. Kavraki, and M. Y. Vardi, "Motion planning with complex goals," *IEEE Robotics and Automation Magazine*, vol. 18, pp. 55–64, 2011.

[4] L. Habets, P. J. Collins, and J. H. van Schuppen, "Reachability and control synthesis for piecewise-affine hybrid systems on simplices," *IEEE Trans. on Automatic Control*, vol. 51, pp. 938–948, 2006.

[5] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *Proc. of IEEE Conf. on Decision and Control*, 2009.

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

[7] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Trans. on Automatic Control*, 2012.

[8] B. Yordanov, J. Tumova, I. Cerna, J. Barnat, and C. Belta, "Temporal logic control of discrete-time piecewise affine systems," *IEEE Trans. on Automatic Control*, vol. 57, pp. 1491–1504, 2012.

[9] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proc. Symp. on Princp. of Prog. Lang.*, 1989, pp. 179–190.

[10] R. Alur and S. La Torre, "Deterministic generators and games for LTL fragments," *ACM Trans. Comput. Logic*, vol. 5, no. 1, pp. 1–25, 2004.

[11] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of Reactive(1) designs," *Journal of Computer and System Sciences*, vol. 78, pp. 911–938, 2012.

[12] Y. Chen, J. Tumova, and C. Belta, "LTL robot motion control based on automata learning of environmental dynamics," in *IEEE Int. Conf. on Robotics and Automation*, 2012.

[13] R. Ehlers, "Generalized Rabin(1) synthesis with applications to robust system synthesis," in *NASA Formal Methods*. Springer, 2011.

[14] O. Maler, A. Pnueli, and J. Sifakis, "On the synthesis of discrete controllers for timed systems," in *STACS 95*. Springer, 1995, vol. 900, pp. 229–242.

[15] E. M. Wolff, U. Topcu, and R. M. Murray, "Efficient reactive controller synthesis for a fragment of linear temporal logic," in *In Proc. Int. Conf. on Robotics and Automation*, 2013.

[16] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal logic-based reactive mission and motion planning," *IEEE Trans. on Robotics*, vol. 25, pp. 1370–1381, 2009.

[17] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, pp. 1695–1708, 2011.

[18] E. M. Wolff, U. Topcu, and R. M. Murray, "Optimal control with weighted average costs and temporal logic specifications," in *Proc. of Robotics: Science and Systems*, 2012.

[19] K. Chatterjee, T. A. Henzinger, and M. Jurdzinski, "Mean-payoff parity games," in *Annual Symposium on Logic in Computer Science (LICS)*, 2005.

[20] G. Jing and H. Kress-Gazit, "Improving he continuous execution of reactive ltl-based controllers," in *Proc. of Int. Conf. on Robotics and Automation*, 2013.

[21] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[22] E. Grädel, W. Thomas, and T. Wilke, Eds., *Automata, Logics, and Infinite Games: A Guide to Current Research*. Springer-Verlag New York, Inc., 2002.

[23] D. P. Bertsekas, *Dynamic Programming and Optimal Control (Vol. I and II)*. Athena Scientific, 2001.

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms: 2nd ed*. MIT Press, 2001.

[25] G. Holzmann, *Spin Model Checker, The Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[26] C. E. Noon and J. C. Bean, "An efficient transformation of the generalized traveling salesman problem," *INFOR*, vol. 31, pp. 39–44, 1993.

[27] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," in *Proc. of the 13th Int. Conf. on Computer Aided Verification*, 2001.