Synthesizing Reactive Test Environments for Autonomous Systems: Testing Reach-Avoid Specifications with Multi-Commodity Flows

Apurva Badithela*[1], Josefine B. Graebener*[2], Wyatt Ubellacker[1], Eric V. Mazumdar[1], Aaron D. Ames[1], Richard M. Murray[1]

***Abstract*— We study automated test generation for testing discrete decision-making modules in autonomous systems. Linear temporal logic is used to encode the system specification — requirements of the system under test — and the test specification, which is unknown to the system and describes the desired test behavior. The reactive test synthesis problem is to find constraints on system actions such that in a test execution, both the system and test specifications are satisfied. To do this, we use the specifications and their corresponding Büchi automata to construct the specification product automaton. Then, a virtual product graph representing all possible test executions of the system is constructed from the transition system and the specification product automaton. The main result of this paper is framing the test synthesis problem as a multi-commodity network flow optimization. This optimization is used to derive reactive constraints on system actions, which constitute the test environment. The resulting test environment ensures that the system meets the test specification while also satisfying the system specification. We illustrate this framework in simulation using grid world examples and demonstrate it on hardware with the Unitree A1 quadruped, where we test dynamic locomotion behaviors reactively.**

I. Introduction

Operational testing of autonomous systems at various levels of abstraction — from low-level continuous dynamics to high-level discrete decision-making — is essential for verification and validation. In formal methods, testing often refers to falsification, where inputs to the system are found such that the resulting trace violates system requirements [1]–[7]. Falsification methods typically minimize a robustness metric associated with the formal specifications of the system to find inputs that result in falsifying traces [8]–[10]. However, another approach to testing is to have test engineers hand-design test scenarios as seen in the qualification tests of the DARPA Urban Challenge [11], [12]. In this work, we bridge these two approaches by leveraging test engineer expertise at the specification level for testing discrete, long-horizon decision-making in robotic systems, and then automatically constructing the corresponding test environment. The test engineer characterizes the desired test behavior in the test specification, and the test environment is constructed such that both the system and test specifications can be satisfied by a correctly designed system. In the last decade, the control synthesis community has demonstrated the effectiveness of using temporal logic to specify formal requirements for robotic systems [13]–[16]. Furthermore, we assume that via the use of rulebooks and industry standard manuals [17]–[19], a test engineer can provide these high-level descriptions on the desired test outcomes using temporal logic. Our notion of testing in this work is complementary to falsification — we seek to construct a test environment to observe a desired test behavior, after which falsification could be applied to determine the worst-case scenario. Our approach to test generation shares similarities with existing methods, but has key differences. Similar to [20], we characterize the mission requirements on the system as a system specification, and characterize the desired behavior to be observed during the test via a test specification, which is unknown to the system. However, unlike [20], we seek to construct a test environment, by constraining actions of the system, such that: a) a correctly designed system can still satisfy its requirements, and b) the test specification is satisfied if the system specification is satisfied (Problem 1). Additionally, we seek to synthesize tests in which the system is not too restricted in its decision-making (Problem 2).

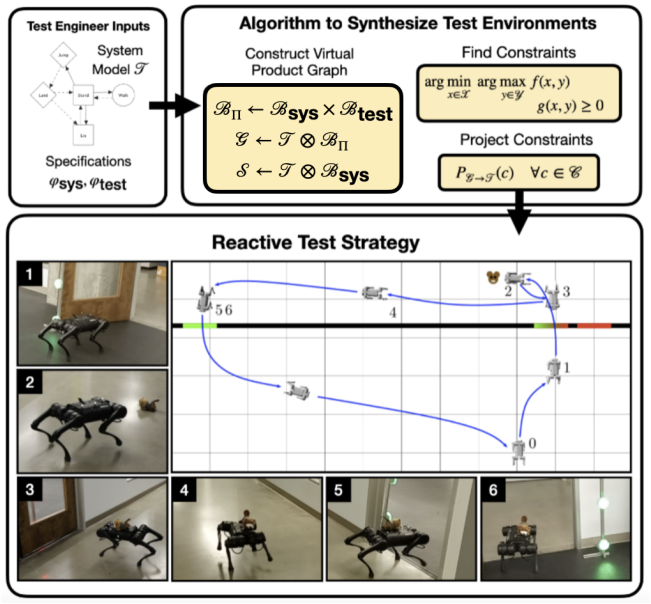


Fig. 1: Overview of the test environment synthesis framework and the hardware demonstration.

For synthesizing a test environment that is consistent with the test specification, we borrow from automata-theoretic

* The authors contributed equally. Corresponding authors: A. Badithela, J.B. Graebener {apurva,jgraeben}@caltech.edu

We acknowledge funding from AFOSR Test and Evaluation Program, grant FA9550-19-1-0302, National Science Foundation award CNS-1932091, and Dow (#227027AT).

[1]Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA 91125, USA

[2]Graduate Aerospace Laboratories, California Institute of Technology, Pasadena, CA 91125, USA

constructions commonly used in reactive synthesis [21]. However, the problem does not reduce to synthesizing a test environment to a conjunction of the system specification and test specification. In reactive synthesis, either fully cooperative or fully adversarial approaches are considered [21]–[23], neither of which accurately capture the problem in our setting. Here, the tester can assume cooperation from the system with regards to the system specification, but since the system is unaware of the test specification, the system is adversarial with respect to the test specification in the worst-case. As a result, standard reactive synthesis tools cannot easily be ported over. Building upon the results of [24], the key contributions of this paper are (i) framing reactive test environment synthesis for reach-avoid specifications as a multi-commodity network flow problem, (ii) formulating the problem as a min-max optimization, whose solution results in a constrained test, and (iii) hardware demonstrations of the resulting test environment to reactively test dynamic locomotion behaviors of the Unitree A1 quadruped. A key advantage of our method is that the synthesized test is reactive — the constraints visible to the system under test are reactive to the system state and depend on the system's strategy, which is not known to the tester *a priori*.

II. Background

In this section, we provide a short background on temporal logic, automata, and network flows.

A. Temporal Logic, Transition Systems, and Automata

Linear temporal logic (LTL) can be used as a specification language to describe linear time properties [25]. The syntax of LTL is comprised of both logical ($\wedge$ *and*, $\vee$ *or*, and $\neg$ *negation*) and temporal operators ($\bigcirc$ *next*, $\Box$ *always*, $\Diamond$ *eventually*, and $\mathcal{U}$ *until*) operators. LTL can specify requirements on high-level decision-making in autonomous systems such as *safety* $\Box(\varphi^s_{\text{sys}})$, *progress* $\Diamond(\varphi^p_{\text{sys}})$, and *fairness* $\Box\Diamond(\varphi^f_{\text{sys}})$.

A nondeterministic Büchi automaton (NBA) [26] is a tuple $\mathcal{B} = (Q, \Sigma, \delta, Q_0, F)$, where Q represents the states, $\Sigma = 2^{AP}$ is the alphabet over the finite set of atomic propositions AP, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, $Q_0 \subseteq Q$ represents initial conditions, and $F \subseteq Q$ is the set of acceptance states. A transition system is a tuple $\mathcal{T} = (S, A, E, I, AP, L)$ where S is the set of states, A is the set of actions, $E : S \times A \to S$ is the transition relation, $I \subseteq S$ is the set of initial states, AP is the set of atomic propositions, and $L : S \to 2^{AP}$ is a labeling function that indicates the set of atomic propositions that evaluate to *true* at a particular state. A trace of the system $\mathcal{T}$ is an infinite sequence of states $\sigma = s_0 s_1 \dots$ where $s_i \in \mathcal{T}.S$. Given an LTL formula φ, we say $\sigma \models \varphi$ if $s_0 \models \varphi$.

Definition 1 (Product Automaton)**.** A *product automaton* $\mathcal{P} = \mathcal{T} \otimes \mathcal{B} = (S, A, E, I, AP, L)$ is the synchronous product of transition system $\mathcal{T}$ and NBA $\mathcal{B}$, where:

- $\mathcal{P}.S = \mathcal{T}.S \times \mathcal{B}.Q$, and $\mathcal{P}.A = \mathcal{T}.A$
- $\forall s, t \in \mathcal{T}.S$, $\forall q, p \in \mathcal{B}.Q$ and $a \in \mathcal{T}.A$, if $\mathcal{T}.E(s, a) = t$ and $\mathcal{B}.\delta(q, \mathcal{T}.L(t)) = p$, then $\mathcal{P}.E((s, q), a) = (t, p)$,
- $\mathcal{P}.I = \{(s_0, q_0) | s_0 \in \mathcal{T}.I,\ q_0 \in \mathcal{B}.Q_0\}$
- $\mathcal{P}.AP = \mathcal{B}.Q$, and
- $\mathcal{P}.L : \mathcal{P}.S \to 2^{\mathcal{P}.AP}$ such that $\mathcal{P}.L((s, q)) = \{q\}$.

Definition 2 (Asynchronous Product Automaton)**.** An *asynchronous product* of Büchi automata $\mathcal{B}_1, \dots, \mathcal{B}_n$ is the Büchi automaton $\mathcal{B}_\pi = \mathcal{B}_1 \times \dots \times \mathcal{B}_n = (Q, \Sigma, \delta, Q_0, F)$, where:

- $\mathcal{B}_\pi.Q := \mathcal{B}_1.Q \times \dots \times \mathcal{B}_n.Q$, the Cartesian product of the states of the individual automata,
- $\mathcal{B}_\pi.\Sigma := \bigcup_{i=1}^n \mathcal{B}_i.\Sigma$,
- $\mathcal{B}_\pi.\delta((u_1, \dots, u_n), l) = (v_1, \dots, v_n)$ if $\exists i \in \{1, \dots, n\}$ such that $\mathcal{B}_i.\delta(u_i, l) = v_i$, and $\forall j \neq i$, $u_j = v_j$,
- $\mathcal{B}_\pi.Q_0 = \{(q_1, \dots, q_n) | \forall i,\ 1 \leq i \leq n,\ q_i \in \mathcal{B}_i.Q_0\}$,
- $\mathcal{B}_\pi.F = \{(s_1, \dots, s_n) | \exists i, 1 \leq i \leq n, \text{ s.t. } s_i \in \mathcal{B}_i.F\}$.

B. System and Test Environment

The system specification and the test specification represent requirements on the system under test and the test environment, respectively [20]. The system is assumed to be designed according to its specification, and has no knowledge of the test specification. We consider the system and test specifications belonging to the reach-avoid fragment of LTL which captures safety and progress requirements as follows:

$$\varphi_{\text{sys}} = \Box\varphi^s_{\text{sys}} \wedge \Diamond\varphi^p_{\text{sys}},\ \varphi_{\text{test}} = \bigwedge_i \Diamond(\varphi^p_{\text{test}})_i. \tag{1}$$

We show that it is possible to model the set of test executions using network flows on an automaton.

Definition 3 (Flow Network)**.** A *flow network* is a tuple $\mathcal{N} = \langle V, E, c, s, t \rangle$ where V is a set of vertices, E is a set of directed edges, $E \subseteq V \times V$, c is a capacity function for the amount of flow that each edge can transfer, and $s \in V$ are the source vertices and $t \in V$ are the target sink vertices. The flow $\mathbf{f} \in \mathbb{R}^{|E|}$ maps each edge $e \in E$ to a non-negative real number, satisfying capacity and conservation constraints. The flow across an edge e is denoted as f^e. The total flow across the network is the net flow out of the source, $F = \sum_{v:(s,v)\in E} f^{(s,v)}$. A multi-commodity flow network has multiple source-sink pairs and their corresponding flows compete for edge capacity [27].

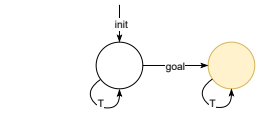
III. Synthesizing Reactive Test Environments

This section sets up the test synthesis problem statement and introduces a running example to illustrate our approach.

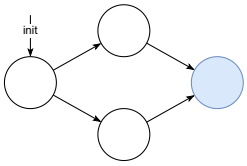
A. Problem Statement

The system and test specifications are written at the same level of abstraction as the model of the system characterized by the transition system $\mathcal{T}$. We require that the sub-formulas of the test specification, φ^s_{test} and φ^p_{test} in equation (1), be high-level descriptions of desired test scenarios provided by a test engineer. In this way, the task of describing the desired test behavior is left to the test engineer, but synthesizing a corresponding test environment can be automated.

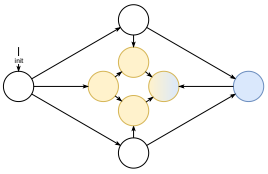
Problem 1. Given a discrete abstraction of a system model $\mathcal{T}$, and system and test specifications, φ_{sys} and φ_{test}, defined over the set AP, find the set of transitions of the system



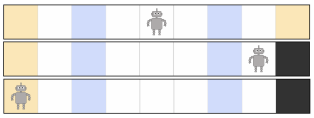
(a) System Büchi automaton B_{sys}.



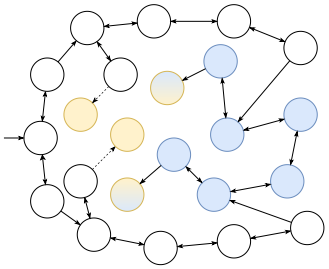
(b) Tester Büchi automaton $\mathcal{B}_{\text{test}}$.



(c) Specification product automaton $\mathcal{B}_\pi$.



(d) Resulting Test Execution.



(e) Virtual game graph $\mathcal{G} = \mathcal{T} \otimes \mathcal{B}_\pi$. The initial state corresponds to S, (partially) yellow states to T, and exclusively blue states to I.

Fig. 2: Büchi automata for the system specification, the test specification and the product automaton $\mathcal{B}_\pi = \mathcal{B}_{\text{test}} \times \mathcal{B}_{\text{sys}}$, and the virtual product graph $\mathcal{G}$ for the corridor navigation example. The accepting states of the system are shaded in yellow and the acceptance states of the tester are shaded in blue, with nodes shaded in both yellow and blue representing acceptance states of both tester and system. Transition labels and self-loops have been omitted in $\mathcal{B}_{\text{test}}$, $\mathcal{B}_\pi$, and $\mathcal{G}$ for clarity.

$E^{\text{cuts}}_{\mathcal{T}} \subset \mathcal{T}.E$ that need to be constrained such that all traces σ of the constrained system satisfy the property:

$$\exists \sigma \text{ s.t. } \sigma \models \varphi_{\text{sys}} \wedge (\sigma \models \varphi_{\text{sys}} \implies \sigma \models \varphi_{\text{test}}). \quad (2)$$

In other words, a trace of the constrained system that satisfies the system specification must also satisfy the test specification, and the constraints are synthesized such that there always exists a trace satisfying the system specification. In addition to determining constraints that result in test executions of the system abiding by equation (2), we do not want the system to be so constrained that it does not have freedom in decision-making during the test. In this work, we use maximum network flow as a proxy for the maximum freedom a system has to achieve its specifications, and we present an algorithmic framework that addresses both these problems on examples in both simulation and hardware.

B. Running Example: Robot in a corridor

Consider a corridor in a grid world shown in Figure 2d. The system under test is starting in the middle of the corridor with the goal of reaching either end. The dynamics are simple grid world dynamics enabling horizontal transitions to neighboring grid cells. The desired test behavior is to observe the system visit the two blue cells, $\varphi_{\text{test}} = \Diamond \text{key}_1 \wedge \Diamond \text{key}_2$. The system specification is given as $\varphi_{\text{sys}} = \Diamond \text{goal}$, which corresponds to the yellow grid cells. We then constrain the system transitions according to our algorithm by placing obstacles on the grid cells.

C. Constructing Product Automata

We use automata theory to define the specification product automaton and virtual product graph. The system product graph $\mathcal{G}_{\text{sys}} = \mathcal{T} \otimes \mathcal{B}_{\text{sys}}$ is the product automaton of the transition system $\mathcal{T}$ and the Büchi automaton of the specification φ_{sys}. The asynchronous product is used to construct the specification product automaton of the system and test Büchi automata.

Definition 4 (Specification Product Automaton)**.** The *specification product automaton* $\mathcal{B}_\pi = \mathcal{B}_{\text{sys}} \times \mathcal{B}_{\text{test}}$ is the asynchronous product of the Büchi automata of the system and the test specification. In particular, $\mathcal{B}_\pi.F = \{(q_{\text{sys}}, q_{\text{test}}) \in \mathcal{B}_\pi.Q | q_{\text{sys}} \in \mathcal{B}_{\text{sys}}.F\} \cup \{(q_{\text{sys}}, q_{\text{test}}) \in \mathcal{B}_\pi.Q | q_{\text{test}} \in \mathcal{B}_{\text{test}}.F\}$.

Definition 5 (Virtual Product Graph)**.** The synchronous product of transition system $\mathcal{T}$ and the NBA $\mathcal{B}_\pi$ is the *virtual product graph* $\mathcal{G} = \mathcal{T} \otimes \mathcal{B}_\pi$.

Constraints will be synthesized on $\mathcal{G}$ and then mapped to the transition system $\mathcal{T}$. A test execution is a trace σ defined over $\mathcal{T}$, which can be mapped to $\mathcal{G}$ and $\mathcal{G}_{\text{sys}}$. For simplicity, we denote σ as the trace on all of these transition systems, and infer the transition system from context.

Definition 6 (Source, Intermediate and Target Nodes)**.** The source (S), intermediate (I) and target (T) are the set of nodes on the virtual product graph $\mathcal{G}$ with the following properties:

$$\mathtt{S} = \{(s_0, q_0) \in \mathcal{G}.S | q_0 \in \mathcal{B}_\pi.Q_0\}$$
$$\mathtt{I} = \{(s, (q_{\text{sys}}, q_{\text{test}})) \in \mathcal{G}.S | q_{\text{test}} \in \mathcal{B}_{\text{test}}.F,\ q_{\text{sys}} \notin \mathcal{B}_{\text{sys}}.F\}$$
$$\mathtt{T} = \{(s, (q_{\text{sys}}, q_{\text{test}})) \in \mathcal{G}.S | q_{\text{sys}} \in \mathcal{B}_{\text{sys}}.F\}$$

The source nodes S represent the initial conditions of the test, the intermediate nodes I represent the acceptance states corresponding to the test specification, and the target nodes T represent the acceptance states corresponding to the system specification. We denote the flow $\mathbf{f}_{\mathtt{S}\to\mathtt{I}}$ for the flow network $(\mathcal{G}.S, \mathcal{G}.E, \mathbf{1}, \mathtt{S}, \mathtt{I})$. The flows $\mathbf{f}_{\mathtt{I}\to\mathtt{T}}$ and $\mathbf{f}_{\mathtt{S}\to\mathtt{T}}$ are similarly denoted. The flow $\mathbf{f}_{\mathtt{S}\to\mathtt{T}}$ is defined to have zero flow on edges into and out of the intermediate I, and is referred to as *bypass flow*. For the running example, the automata $\mathcal{B}_{\text{sys}}$, $\mathcal{B}_{\text{test}}$, $\mathcal{B}_\pi$, and the virtual product graph $\mathcal{G}$ with the corresponding S, I, and T nodes are illustrated in Figure 2.

Problem 2. Given the setting in Problem 1, synthesize the set of edge constraints $E^{\text{cuts}}_{\mathcal{G}}$ on the virtual product graph $\mathcal{G}$ such that flow from S to T (visiting nodes in I) is maximized, and bypass flow $\mathbf{f}_{\mathtt{S}\to\mathtt{T}}$ is cut.

D. Multi-Commodity Flows and Bilevel Optimization

To synthesize constraints $E^{\text{cuts}}_{\mathcal{G}}$ on $\mathcal{G}$, we define multi-commodity flows on $\mathcal{G}$ and solve a bilevel optimization.

The constraints $E^{\text{cuts}}_{\mathcal{G}}$ are such that a) $\exists \sigma$ s.t. $\sigma \models \varphi_{\text{sys}}$, and for every such σ, the test specification is also satisfied (equation (2)), and b) the flow from S to T (through I) is maximized. These constraints are then mapped to system constraints $E^{\text{cuts}}_{\mathcal{T}}$. A brute force approach to solving Problems 1 and 2 is not viable as it involves finding sets of paths $P_{\mathtt{S}\to\mathtt{I}}$ and $P_{\mathtt{I}\to\mathtt{T}}$ realizing max-flow from S to I, and I to T, respectively, such that $P_{\mathtt{S}\to\mathtt{I}}$ and $P_{\mathtt{I}\to\mathtt{T}}$ are disjoint except at intermediate I. Finding such a feasible pair of $P_{\mathtt{S}\to\mathtt{I}}$ and $P_{\mathtt{I}\to\mathtt{T}}$ would take exponential time because enumerating all paths is exponential in the size of the graph [27].

To address this combinatorial problem, we formulate a bilevel optimization that relaxes edge cuts $E^{\text{cuts}}_{\mathcal{G}}$ to take fractional values. The choice of objective function and the relaxation make the optimization tractable. While the cut values of some edges take on fractional values due to the relaxation, we find empirically that these fractional cuts are not relevant to constraining the flow. The system and tester are players that optimize for different flows on the same virtual product graph $\mathcal{G}$. The system player maximizes bypass flow $F_{\mathtt{S}\to\mathtt{T}}$, which represents system traces satisfying φ_{sys} without satisfying φ_{test}. The tester maximizes flows $F_{\mathtt{S}\to\mathtt{I}}$ and $F_{\mathtt{I}\to\mathtt{T}}$, and indirectly constrains bypass flow by placing cuts on system transitions. Unlike the canonical multi-commodity flow framework [28], our flows do not compete for edge capacities, but are equally constrained by edge cuts. The total flow through I is defined as follows,

$$F_{\text{total}} = \min\{F_{\mathtt{S}\to\mathtt{I}}, F_{\mathtt{I}\to\mathtt{T}}\}. \tag{3}$$

Network flow constraints are written in normalized form by the auxiliary variable $t := 1/F_{\text{total}}$. For brevity, we use the same notation to denote the normalized flows. The variable $\mathbf{d} \in \mathbb{R}^{|\mathcal{G}.E|}_{\geq 0}$ denotes edge cuts on $\mathcal{G}$, and d^e is the constraint on edge $e \in \mathcal{G}.E$ — $d^e = t$ implies that e is cut or fully constrained and $d^e = 0$ is unconstrained. As the outer (min) player, the tester variables are the flows $\mathbf{f}_{\mathtt{S}\to\mathtt{I}}$ and $\mathbf{f}_{\mathtt{I}\to\mathtt{T}}$, edge cuts $\mathbf{d}$, and the auxiliary variable t. The objective function is such that the tester maximizes the total flow F_{total} and minimizes the total bypass flow $F_{\mathtt{S}\to\mathtt{T}}$. Likewise, the system player maximizes bypass flow $F_{\mathtt{S}\to\mathtt{T}}$. Next, the constraints of the bilevel optimization are detailed. Capacity constraints for normalized variables in this optimization are,

$$\begin{aligned} \forall e \in \mathcal{G}.E, \quad & 0 \leq d^e \leq t, \qquad 0 \leq f^e_{\mathtt{S}\to\mathtt{I}} \leq t, \\ & 0 \leq f^e_{\mathtt{S}\to\mathtt{T}} \leq t, \quad 0 \leq f^e_{\mathtt{I}\to\mathtt{T}} \leq t. \end{aligned} \tag{c1}$$

Cut constraints correspond to the cut variable and flow variable of an edge competing for its capacity. For all $k \in \{\mathtt{S}\to\mathtt{I}, \mathtt{I}\to\mathtt{T}, \mathtt{S}\to\mathtt{T}\}$, the cut constraints are as follows,

$$\forall e \in \mathcal{G}.E, \quad d^e + f^e_k \leq t. \tag{c2}$$

Flow conservation ensures that the flow entering a node is equal to the flow leaving the node (unless the node is a source or a target). For $k \in \{\mathtt{S}\to\mathtt{I}, \mathtt{I}\to\mathtt{T}, \mathtt{S}\to\mathtt{T}\}$, the conservation constraints are as follows,

$$\forall v \in \mathcal{G}.S \sum_{u:(u,v)\in\mathcal{G}.E} f^{(u,v)}_k = \sum_{u:(v,u)\in\mathcal{G}.E} f^{(v,u)}_k. \tag{c3}$$

Since the tester is maximizing F_{total} in the objective, equation (3) is captured as the following (normalized) constraint,

$$1 \leq \sum_{v:(\mathtt{S},v)\in\mathcal{G}.E} f^{(\mathtt{S},v)}_{\mathtt{S}\to\mathtt{I}}, \quad 1 \leq \sum_{v:(\mathtt{I},v)\in\mathcal{G}.E} f^{(\mathtt{I},v)}_{\mathtt{I}\to\mathtt{T}}. \tag{c4}$$

Our framework synthesizes test environments by constraining, not forcing, system actions. Under the synthesized constraints, a system trace σ on $\mathcal{G}$ satisfying the system specification is guaranteed to exist. However, the synthesized constraints on system actions could result in the system specification becoming unsatisfiable on the system product automaton $\mathcal{G}_{\text{sys}}$, that is the system cannot re-plan to satisfy its requirements. If possible, our framework should return constraints for which at every temporal instance of the test execution, the system should find a feasible path to satisfying its requirements. We add the following constraints such that at every system state during the test execution there exists a path to the acceptance states of φ_{sys} on $\mathcal{G}_{\text{sys}}$.

The tester places constraints reactive to the system state, and it is not necessary that all constraints on the virtual product graph $\mathcal{G}$ are active at every temporal instance. To reason about constraints that would be visible to the system at $s \in \mathcal{G}.S$, we define mappings between the product automata.

Each state in the specification product automaton $\mathcal{B}_\pi$ represents a temporal event during the test execution. For each $q \in \mathcal{B}_\pi.Q$ the corresponding edges that are active at the same time are:

$$C_{\mathcal{G}}(q) = \{((s,q),(s_1,q_1)) \in \mathcal{G}.E\}. \tag{4}$$

The constraints on this set of active edges is the maximum number of constraints that can be simultaneously present. For every $q \in \mathcal{B}_\pi.Q$ and edge cuts d^e for active edges $e \in C_{\mathcal{G}}(q)$, we need to ensure that there remains a path in $\mathcal{G}_{\text{sys}}$ to the system acceptance states. This requires mapping the edge cut values of edges in $C_{\mathcal{G}}(q)$ to edges in $\mathcal{G}_{\text{sys}}$. To enable this, we first map nodes from $\mathcal{G}$ to $\mathcal{G}_{\text{sys}}$. A node $g = (s,(q_{\text{sys}}, q_{\text{test}}))$ in $\mathcal{G}$ is mapped to $\mathcal{G}_{\text{sys}}$ via the following projection,

$$P_{\mathcal{G}\to\mathcal{G}_{\text{sys}}}(g) = (s, q_{\text{sys}}). \tag{5}$$

For each $q \in \mathcal{B}_\pi.Q$, the cut values of active edges in $C_{\mathcal{G}}(q)$ are mapped to corresponding edges on $\mathcal{G}_{\text{sys}}$. For every $(u,v) \in C_{\mathcal{G}}(q)$, the corresponding edge on $\mathcal{G}_{\text{sys}}$ shares the same cut value,

$$d^e_{\mathcal{G}_{\text{sys}}}(q) = d^{(u,v)}, \tag{6}$$

where $e = (P_{\mathcal{G}\to\mathcal{G}_{\text{sys}}}(u), P_{\mathcal{G}\to\mathcal{G}_{\text{sys}}}(v)) \in \mathcal{G}_{\text{sys}}.E$.

For every $q \in \mathcal{B}_\pi.Q$, let $\mathbf{f}_{\mathtt{S}_{\text{sys}}\to\mathtt{T}_{\text{sys}}}(q)$ denote the maximum flow from source $\mathtt{S}_{\text{sys}} := P_{\mathcal{G}\to\mathcal{G}_{\text{sys}}}(\mathtt{S})$ to target $\mathtt{T}_{\text{sys}} := P_{\mathcal{G}\to\mathcal{G}_{\text{sys}}}(\mathtt{T})$ on system product graph $\mathcal{G}_{\text{sys}}$ with cut values set by $d^e_{\mathcal{G}_{\text{sys}}}(q)$. For brevity, we do not elaborate the constraints here, but the flow respects the standard network flow constraints analogous to equations (c1)-(c3). Since the constraints are synthesized agnostic to the system controller, to ensure that the active cuts do not prohibit satisfaction of the system specification, we require the following condition to be satisfied:

$$\forall q \in \mathcal{B}_\pi.Q, \quad \sum_{e=(\mathtt{S}_{\text{sys}},v)\in\mathcal{G}_{\text{sys}}.E} f^e_{\mathtt{S}_{\text{sys}}\to\mathtt{T}_{\text{sys}}}(q) \geq t. \tag{c5}$$

Since the above constraint is defined from a fixed source $\mathtt{S}_{\text{sys}}$ to target $\mathtt{T}_{\text{sys}}$ on $\mathcal{G}_{\text{sys}}$, we assume that from every state $(s, q_{\text{sys}}) \in \mathcal{G}_{\text{sys}}.S$, there exists a path back to the source $\mathtt{S}_{\text{sys}}$. This ensures that there are no trap states as the system can return to the source, and from there a path to the target is guaranteed to exist. For the examples of this paper, this assumption is satisfied. In future work, we would like to prove these properties for a larger class of specifications and transition systems. Therefore, the bilevel optimization for synthesizing reactive constraints is as follows,

$$\begin{aligned} &\text{MCF-OPT}(\lambda): \\ &\underset{\substack{\mathbf{f}_{\mathtt{S}\to\mathtt{I}}\mathbf{f}_{\mathtt{I}\to\mathtt{T}},\mathbf{d},t,\\ \mathbf{f}_{\mathtt{S}_{\text{sys}}\to\mathtt{T}_{\text{sys}}}(q),\,\forall q\in\mathcal{B}_\pi.Q}}{\operatorname{argmin}} \quad \underset{\mathbf{f}_{\mathtt{S}\to\mathtt{T}}}{\operatorname{argmax}} \quad t+\lambda \sum_{v:e=(\mathtt{S},v)\in\mathcal{G}.E} f^e_{\mathtt{S}\to\mathtt{T}} \qquad (7)\\ &\qquad\qquad \text{s.t.} \qquad\qquad (\text{c1})-(\text{c5}), \end{aligned}$$

where the regularization parameter λ penalizes the tester (and rewards the system) on the total bypass flow $F_{\mathtt{S}\to\mathtt{T}}$ flow. This optimization is in the form of a min-max Stackelberg game with dependent constraint sets studied in [29]. The optimization (7) returns fractional cut values d^e for edges on the virtual product graph $\mathcal{G}$. Edges with cut values close to t are fully constrained and denoted as $E^{\text{cuts}}_{\mathcal{G}}$. Lower fractional values for d^e still allow flow to pass through and are not considered cut.

Algorithm 1: Constraining Virtual Product Graph $\mathcal{G}$

procedure AUTOMATA($\mathcal{T}, \varphi_{\text{sys}}, \varphi_{\text{test}}$)
  $\mathcal{B}_{\text{sys}} \leftarrow \text{BA}(\varphi_{\text{sys}})$ ▷ System Büchi automaton
  $\mathcal{B}_{\text{test}} \leftarrow \text{BA}(\varphi_{\text{test}})$ ▷ Tester Büchi automaton
  $\mathcal{B}_\pi \leftarrow \mathcal{B}_{\text{sys}} \times \mathcal{B}_{\text{test}}$ ▷ Specification product
  $\mathcal{G}_{\text{sys}} \leftarrow \mathcal{T} \otimes \mathcal{B}_{\text{sys}}$ ▷ System product
  $\mathcal{G} \leftarrow \mathcal{T} \otimes \mathcal{B}_\pi$ ▷ Virtual Product Graph
  return $\mathcal{G}, \mathcal{G}_{\text{sys}}, \mathcal{B}_\pi, \mathcal{B}_{\text{sys}}, \mathcal{B}_{\text{test}}$

procedure CONSTRAINTS($\mathcal{T}, \mathcal{G}, \mathcal{G}_{\text{sys}}, \mathcal{B}_\pi, \mathcal{B}_{\text{sys}}, \mathcal{B}_{\text{test}}$)
  Identify nodes S, I, T on $\mathcal{G}$
  Choose regularization λ
  $\mathbf{f}^*_{\mathtt{S}\to\mathtt{I}}, \mathbf{f}^*_{\mathtt{I}\to\mathtt{T}}, \mathbf{f}^*_{\mathtt{S}\to\mathtt{T}}, \mathbf{d}^*, t^* \leftarrow$ MCF-OPT(λ)
  $E^{\text{cuts}}_{\mathcal{G}} = \emptyset$ ▷ To store cuts of $\mathcal{G}$
  for $e \in \mathcal{G}.E$ **do**
    if $d^{*e} = 1$ **then** ▷ Ignore fractional cuts
      $E^{\text{cuts}}_{\mathcal{G}} \leftarrow E^{\text{cuts}}_{\mathcal{G}} \cup e$
  $\mathcal{G}.E \leftarrow \mathcal{G}.E \setminus E^{\text{cuts}}_{\mathcal{G}}$
  Verify $\mathcal{G}$ has no $F_{\mathtt{S}\to\mathtt{T}}$ flow.
  return $E^{\text{cuts}}_{\mathcal{G}}$

E. Projecting the constraints onto the physical space

To constrain the system's actions during the test execution, the cut edges $E^{\text{cuts}}_{\mathcal{G}}$ are mapped onto the the physical system transitions $\mathcal{T}$. We define the projection,

$$P_{\mathcal{G}\to\mathcal{T}}(g) = s \in \mathcal{T}.S \mid g = (s, (q_{\text{sys}}, q_{\text{test}})) \in \mathcal{G}.S, \qquad (8)$$

which maps each state g in the virtual product graph $\mathcal{G}$ to its corresponding state in the transition system $\mathcal{T}$. This is a mapping where multiple states in $\mathcal{G}$ will map to a single state in $\mathcal{T}$. Additionally, to map a state $g = (s, (q_{\text{sys}}, q_{\text{test}})) \in \mathcal{G}.S$ to its corresponding state in $\mathcal{B}_\pi$ we define the projection,

$$P_{\mathcal{G}\to\mathcal{B}_\pi}(g) = (q_{\text{sys}}, q_{\text{test}}) \in \mathcal{B}_\pi.Q. \qquad (9)$$

We use the projection defined in equation (9) to determine the state of the test execution $q \in \mathcal{B}_\pi.S$. When the system enters a state $g \in \mathcal{G}.S$ with an active cut, the corresponding transition from state $P_{\mathcal{G}\to\mathcal{T}}(g)$ in $\mathcal{T}$ will be constrained. The test environment will accumulate constraints on $\mathcal{T}$ until the test transitions to a $g' \in \mathcal{G}.S$ mapping to $q' \in \mathcal{B}_\pi.Q$, such that $q \neq q'$. The obstacles that were placed previously will be removed and new obstacles will be placed according to the set of active cuts on $\mathcal{G}$ corresponding to q'. This procedure is outlined in Algorithm 2.

This makes our framework reactive to the system state during the test execution, where finding static constraints on $\mathcal{G}$ results in a reactive test strategy that constrains the system actions according to the observed behavior during the test. Thus, test executions are not open-loop and constraints to system actions are made visible according to the system state.

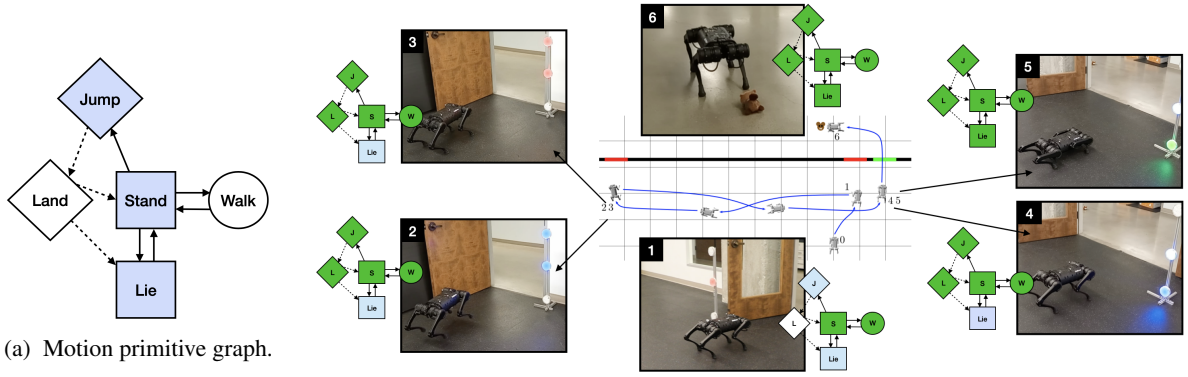
Algorithm 2: Reactive Test Synthesis

procedure REACTIVE TEST($\mathcal{T}, \varphi_{\text{sys}}, \varphi_{\text{test}}$)
  $\mathcal{G}, \mathcal{G}_{\text{sys}}, \mathcal{B}_\pi, \mathcal{B}_{\text{sys}}, \mathcal{B}_{\text{test}} \leftarrow$ AUTOMATA($\mathcal{T}, \varphi_{\text{sys}}, \varphi_{\text{test}}$)
  $E^{\text{cuts}}_{\mathcal{G}} \leftarrow$ CONSTRAINTS($\mathcal{T}, \mathcal{G}, \mathcal{G}_{\text{sys}}, \mathcal{B}_\pi, \mathcal{B}_{\text{sys}}, \mathcal{B}_{\text{test}}$)
  $g = (s, (q_{\text{sys}}, q_{\text{test}})) \leftarrow \mathcal{G}.I$, and $q = (q_{\text{sys}}, q_{\text{test}})$
  $q_{\text{prev}} \leftarrow P_{\mathcal{G}\to\mathcal{B}_\pi}(g)$ ▷ Initialize previous q.
  $E^{\text{cuts}}_{\mathcal{T}} \leftarrow \emptyset$ ▷ Initialize empty set of active cuts.
  $E \leftarrow \mathcal{T}.E$ ▷ Original transitions from $\mathcal{T}$.
  while not $q_{\text{sys}} \in \mathcal{B}_{\text{sys}}.F$ **do**
    if $q \neq q_{\text{prev}}$ **then**
      $E^{\text{cuts}}_{\mathcal{T}} \leftarrow \emptyset$ ▷ Reset all active cuts.
    if outgoing_edge$_{\mathcal{G}}(g) \in E^{\text{cuts}}_{\mathcal{G}}$ **then** ▷ Add cut.
      $E^{\text{cuts}}_{\mathcal{T}} \leftarrow E^{\text{cuts}}_{\mathcal{T}} \cup$ outgoing_edge$_{\mathcal{T}}(P_{\mathcal{G}\to\mathcal{T}}(g))$
    $\mathcal{T}.E \leftarrow E \setminus E^{\text{cuts}}_{\mathcal{T}}$ ▷ Update available transitions.
    $s \leftarrow$ system_step$_{\mathcal{T}}(s)$ ▷ System next step.
    $g \leftarrow$ advance_test$_{\mathcal{G}}(g, s)$ ▷ Update state in $\mathcal{G}$.
    $q_{\text{prev}} \leftarrow q$
    $q \leftarrow P_{\mathcal{G}\to\mathcal{B}_\pi}(g)$ ▷ Update state in $\mathcal{B}_\pi$.

IV. EXPERIMENTAL RESULTS

We implemented and validated this framework on simulated grid world examples and hardware experiments. For the examples in this paper, we use a regularization parameter of $\lambda = 1$ and initialize the optimization with flows satisfying the conservation and capacity constraints and no cuts on the virtual product graph. Results on additional grid world and road network examples can be found at this repository[1].

[1] https://github.com/abadithela/Flow-Constraints



(a) Motion primitive graph.

(b) Snapshots of the hardware test execution on the Unitree A1 quadruped.

Fig. 3: Resulting test execution on the Unitree A1 quadruped generated by this framework.

1) Robot in a Corridor: The agent under test in the running example is controlled by a grid world controller synthesized using TuLiP (Temporal Logic and Planning Toolbox) [30]. The algorithms presented in section III-D result in a test execution during which the agent visits the two pre-determined key locations before reaching one of the goal states at the end of the corridor. The resulting test execution is shown in Figure 2d.

2) Hardware Experiments with Quadruped: Next we will find a test strategy to test an actual robotic system, the Unitree A1 quadruped. The quadruped is controlled using a motion primitive layer with behaviors for lying down, standing, walking, and jumping. The underlying dynamics of the transitions between primitives are abstracted away from the higher-level autonomy as described in [31], and can be commanded directly. The autonomy layer is provided by a TuLiP controller generated on an abstraction of the transition system of the quadruped, consisting of grid world locations and states corresponding to the available motion primitives. We find test strategies and execute the resulting test for two test specifications inspired by search and rescue missions.

a) Beaver Rescue: The quadruped's task is to rescue the beaver from the hallway and return it to the lab. The system specification is given as $\varphi_{\text{sys}} = \Diamond\text{goal}$, where *goal* corresponds to the quadruped and the beaver reaching the safe location in the lab. The test specification is given as $\varphi_{\text{test}} = \Diamond\text{door}_1 \wedge \Diamond\text{door}_2$, ensuring that the quadruped will use different doors on the way to the beaver and back into the lab. The resulting test execution first shows the quadruped using door_2 to exit the lab into the hallway, then after it reaches the beaver, door_2 is shut and the quadruped walks to door_1 to finally return to the lab. The reactive aspect here can be observed as follows — if the quadruped chose to enter the hallway through door_1, then the resulting test execution would constrain access to door_1 when the quadruped is attempting to re-enter the lab with the beaver. Snapshots of this test execution can be seen in Figure 1.

b) Motion Primitive Testing: In this example, we test the motion primitives of the quadruped shown in Figure 3a. The goal for the quadruped is reaching the beaver in the hallway. The test specification is given as $\varphi_{\text{test}} = \Diamond\text{jump} \wedge \Diamond\text{lie} \wedge \Diamond\text{stand}$, which ensures that each motion primitive is tested at least once. The test setup includes lights at different heights, which correspond to the motion primitive which might unlock the door. The light starts in blue and after the motion primitive has been executed, the light will turn red (if the door remains locked) or green (if the door is unlocked). Our framework will decide whether the doors will be locked or unlocked according to which motion primitives have already been observed during the test. This is where the reactivity of this framework becomes apparent, if the quadruped chose a different set of doors and motion primitives, the resulting test execution would have been different. Snapshots of the test execution are shown in Figure 3b.

V. Conclusions and Future Work

We introduced a bilevel optimization framework to find reactive test environments that constrain the system under test to satisfy the test specification while also ensuring that a correctly designed system can satisfy its specification. We defined projection functions to map the optimization result into constraints on system actions. We implemented this approach to test high-level decision-making in hardware and executed the resulting reactive test strategy. For future work, we would like to prove that our algorithm is sound and complete, and provide sub-optimality guarantees on the generated test environment. The approach outlined in this paper requires reactive placement of obstacles during a test, which can be challenging for real-world use cases. Therefore, we aim to extend this framework to include dynamic test agents to constrain the system actions, and find cost-effective test environments, for example by minimizing the number of test agents.

Acknowledgments

The authors would like to acknowledge Mani Chandy, Tichakorn Wongpiromsarn, Qiming Zhao, Michel Ingham, Joel Burdick, Leonard Schulman, Shih-Hao Tseng, Ioannis Filippidis, and Ugo Rosolia for insightful discussions.

REFERENCES

[1] S. Sankaranarayanan and G. Fainekos, "Falsification of temporal properties of hybrid systems using the cross-entropy method," in *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*, 2012, pp. 125–134.

[2] J. Kapinski, J. V. Deshmukh, X. Jin, H. Ito, and K. Butts, "Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques," *IEEE Control Systems Magazine*, vol. 36, no. 6, pp. 45–64, 2016.

[3] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2011, pp. 254–257.

[4] G. Chou, Y. E. Sahin, L. Yang, K. J. Rutledge, P. Nilsson, and N. Ozay, "Using control synthesis to generate corner cases: A case study on autonomous driving," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2906–2917, 2018.

[5] T. Dang and T. Nahhal, "Coverage-guided test generation for continuous and hybrid systems," *Formal Methods in System Design*, vol. 34, no. 2, pp. 183–213, 2009.

[6] M. Hekmatnejad, B. Hoxha, and G. Fainekos, "Search-based test-case generation by monitoring responsibility safety rules," *arXiv preprint arXiv:2005.00326*, 2020.

[7] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Falsification of ltl safety properties in hybrid systems," *International Journal on Software Tools for Technology Transfer*, vol. 15, no. 4, pp. 305–320, 2013.

[8] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 167–170.

[9] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[10] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh, M. Vazquez-Chanlatte, and S. A. Seshia, "Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems," in *International Conference on Computer Aided Verification*. Springer, 2019, pp. 432–442.

[11] "Technical Evaluation Criteria," https://archive.darpa.mil/grandchallenge/rules.html.

[12] "DARPA Urban Challenge," https://www.darpa.mil/about-us/timeline/darpa-urban-challenge.

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

[14] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Transactions on Automatic Control*, vol. 57, no. 11, pp. 2817–2830, 2012.

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

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

[17] A. Censi, K. Slutsky, T. Wongpiromsarn, D. Yershov, S. Pendleton, J. Fu, and E. Frazzoli, "Liability, ethics, and culture-aware behavior specification using rulebooks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8536–8542.

[18] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *arXiv preprint arXiv:1708.06374*, 2017.

[19] T. Wongpiromsarn, K. Slutsky, E. Frazzoli, and U. Topcu, "Minimum-violation planning for autonomous systems: Theoretical and practical considerations," in *2021 American Control Conference (ACC)*. IEEE, 2021, pp. 4866–4872.

[20] J. B. Graebener, A. Badithela, and R. M. Murray, "Towards better test coverage: Merging unit tests for autonomous systems," in *NASA Formal Methods: 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24–27, 2022, Proceedings*, 2022, pp. 133–155.

[21] 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, no. 3, pp. 911–938, 2012.

[22] M. Kloetzer and C. Belta, "Dealing with nondeterminism in symbolic control," in *Hybrid Systems: Computation and Control: 11th International Workshop, HSCC 2008, St. Louis, MO, USA, April 22-24, 2008. Proceedings 11*. Springer, 2008, pp. 287–300.

[23] J. Tumova and D. V. Dimarogonas, "Synthesizing least-limiting guidelines for safety of semi-autonomous systems," in *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 5714–5719.

[24] A. Badithela, J. B. Graebener, and R. M. Murray, "Minimally constrained testing for autonomy with temporal logic specifications," 2022.

[25] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[26] J. R. Büchi, *On a Decision Method in Restricted Second Order Arithmetic*. New York, NY: Springer New York, 1990, pp. 425–435. [Online]. Available: https://doi.org/10.1007/978-1-4613-8928-6_23

[27] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[28] V. V. Vazirani, *Approximation algorithms*. Springer, 2001, vol. 1.

[29] I. Tsaknakis, M. Hong, and S. Zhang, "Minimax problems with coupled linear constraints: computational complexity, duality and solution methods," *arXiv preprint arXiv:2110.11210*, 2021.

[30] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray, "Tulip: a software toolbox for receding horizon temporal logic planning," in *Proceedings of the 14th international conference on Hybrid systems: computation and control*, 2011, pp. 313–314.

[31] W. Ubellacker, N. Csomay-Shanklin, T. G. Molnar, and A. D. Ames, "Verifying safe transitions between dynamic motion primitives on legged robots," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 8477–8484.