Minimally Constrained Testing for Autonomy with Temporal Logic Specifications

Apurva Badithela, Josefine Graebener, Richard M. Murray

***Abstract*—In this paper, we study automated test generation for discrete decision-making modules in autonomous systems. First, we consider a subset of Linear Temporal Logic to represent formal requirements on the system and the test environment. The system specification captures requirements for the system under test while the test specification captures basic attributes of the test environment known to the system, and additional structure provided by a test engineer, which is unknown to the system. Second, a game graph representing the high-level interaction between the system and the test environment is constructed from transition systems modeling the system and the test environment. We provide an algorithm that finds the projection of the acceptance conditions of the system and test specifications on the game graph. Finally, to ensure that the system meets the test specification in addition to satisfying the system specification, we present a framework to construct a minimally constrained test. Specifically, we formulate this as a multi-commodity network flows problem, and present two optimizations to solve for the minimally constrained test. We conclude with future directions on applying these algorithms to constrain test environments in self-driving applications.**

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 refers to simulation-based falsification, where inputs to a model of the system are found which result in system violating its requirements [8, 6, 1]. Our notion of testing in this work is complementary to falsification — we seek to observe certain desired behavior during a test execution that is successful with respect to the system's specifications. As illustrated in Figure 1, the system is required to park in a spot, while the test is setup such that the system needs more than one attempt to pull into the parking spot. We characterize the mission requirements on the system as a system specification, and characterize the desired behavior observed during the test via a test specification.

We begin with a test environment that has little influence on the system. However, we might not be able to find a strategy for such a test environment to realize the test specification. Thus, we seek to constrain the test (by constraining actions that the system can take) such that: a) the system can still satisfy its requirements, and b) the test specification is satisfied in a successful execution. Additionally, we seek to minimally constrained tests as these might translate to more flexibility for the autonomous system to satisfy its requirements. The contributions are the following:

- Transforming the problem of finding a test environment consistent with specifications as a graph partitioning problem,
- Defining the notion of minimally constrained test environment and mapping these constraints to the graph partitioning problem,
- Presenting a convex optimization to solve for a minimally node-constrained test (under a special case), and
- Presenting a convex-concave min-max optimization to solve for a minimally edge-constrained test (for the general case).

II. Preliminaries

A. Temporal Logic, Transition Systems, and Automata

Linear Temporal Logic (LTL) can describe temporal properties on a trace of propositional formulas [2]. The syntax of LTL comprises of both logical ($\wedge$ *and*, $\vee$ *or*, and $\neg$ *negation*), and temporal operators ($\bigcirc$ *next*, $\square$ *always*, $\Diamond$ *eventually*, and $\mathcal{U}$ *until*) operators. A Nondeterministic Büchi Automaton (NBA) is a tuple $\mathcal{B} = (Q, 2^{AP}, \delta, Q_0, F)$, where Q represents the states, AP is the set of atomic propositions, δ represents the transition function, $Q_0 \subseteq Q$ represents the initial states, and $F \subseteq Q$ is the set of acceptance states. A transition system is a tuple $\mathcal{T} = (S, A, \rightarrow, I, AP, L)$ where S is a set of states, A is the set of actions, $\rightarrow: S \times A \rightarrow S$ is the transition relation, $I \subseteq S$ is the set of initial states, AP is the set of atomic propositions, and $L : S \rightarrow 2^{AP}$ is a labeling function that indicates the set of atomic propositions that evaluate to *true* at a particular state.

Definition 1 (Product Automaton)**.** The product of a transition system $\mathcal{T} = (S, A, \rightarrow, I, AP, L)$ and a NBA $\mathcal{B} = (Q, 2^{AP}, \rightarrow, Q_0, F)$, is the tuple $\mathcal{T} \otimes \mathcal{B} = (S', A, \rightarrow', I', AP', L')$, where:

- $S' = S \times Q$,
- $\forall s, t \in S$, $\forall q, p \in Q$ such that $s \xrightarrow{a} t$ and $q \xrightarrow{L(t)} p$, then, $(s, q) \underset{a}{\rightarrow}' (t, p)$,
- $I' = \{(s_0, q) : s_0 \in I,\ \exists q_0 \in Q_0 \text{ s.t. } q_0 \xrightarrow{L(s_0)} q\}$,
- $AP' = Q$, and
- $L' : S \times Q \rightarrow 2^Q$ such that $L'((s, q)) = \{q\}$.

A. Badithela is a PhD candidate in Control and Dynamical Systems, Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA 91106, USA apurva@caltech.edu

J. Graebener is a PhD candidate in Space Engineering, Graduate Aerospace Laboratories, California Institute of Technology, Pasadena, CA 91106, USA jgraeben@caltech.edu

R.M. Murray is with the Control and Dynamical Systems, Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA 91106, USA murray@cds.caltech.edu

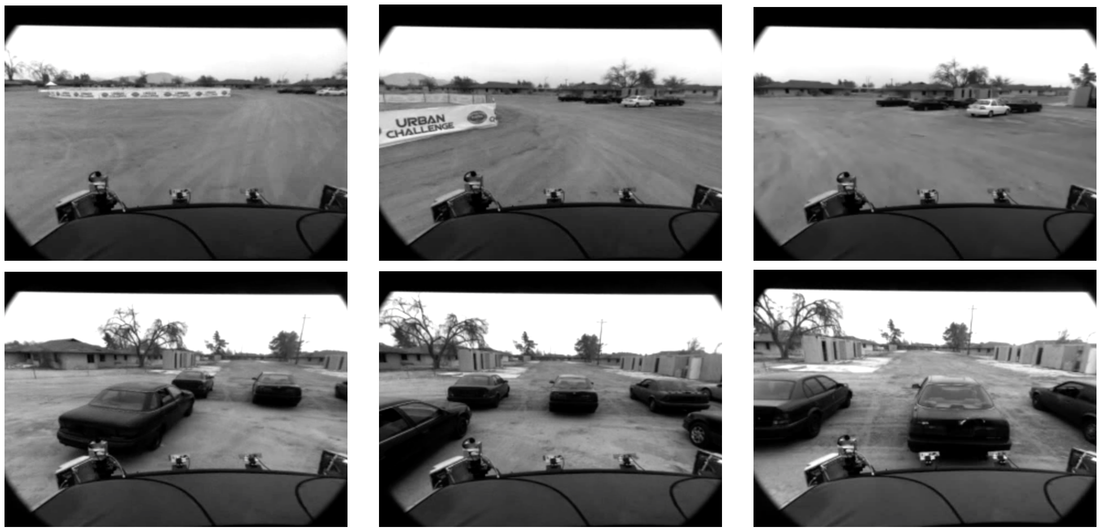

Fig. 1: Caltech's entry Alice in the 2007 DARPA Urban Challenge undergoing a qualifying test. *Top row:* Alice entering the lot and navigating to parking spot, *Bottom row:* Multiple attempts to pull-in to spot before successful parking. Alice's requirement, or *system specification*, is to safely park in a parking spot. Informally, one possible test specification could be that Alice should require more than one attempt in entering the parking spot to successfully park. Given an empty parking lot as the original test environment, how should obstacles be placed such that Alice requires more than one attempt (satisfying the *test specification*) to successfully park (satisfying the *system specification*)? The locations of obstacles was manually decided in this test; how can we automate finding constraints on the test environment?

B. System and Test Environment

The system specification and the test specification represent requirements on the system under test and the test environment, respectively. In addition to capturing the assumptions of the system, the test specification captures additional requirements that the test environment must be designed for in order to observe the behavior $\square\psi^{\mathrm{s}}_{\mathrm{test}} \wedge \square\lozenge\psi^{\mathrm{f}}_{\mathrm{test}}$ on a test execution.

Definition 2 (System and Test Specification [5])**.** A system specification is the an LTL formula,

$$\begin{aligned}\varphi_{\mathrm{sys}} = (\varphi^{\mathrm{init}}_{\mathrm{test}} \wedge \square\varphi^{\mathrm{s}}_{\mathrm{test}} \wedge \square\lozenge\varphi^{\mathrm{f}}_{\mathrm{test}}) \rightarrow \\ (\varphi^{\mathrm{init}}_{\mathrm{sys}} \wedge \square\varphi^{\mathrm{s}}_{\mathrm{sys}} \wedge \square\lozenge\varphi^{\mathrm{f}}_{\mathrm{sys}}),\end{aligned} \tag{1}$$

A test specification is the LTL formula,

$$\begin{aligned}\varphi_{\mathrm{test}} = (\varphi^{\mathrm{init}}_{\mathrm{sys}} \wedge \square\varphi^{\mathrm{s}}_{\mathrm{sys}} \wedge \square\lozenge\varphi^{\mathrm{f}}_{\mathrm{sys}}) \rightarrow \Big((\varphi^{\mathrm{init}}_{\mathrm{test}} \wedge \square\varphi^{\mathrm{s}}_{\mathrm{test}} \\ \wedge \square\lozenge\varphi^{\mathrm{f}}_{\mathrm{test}}) \wedge \square\psi^{\mathrm{s}}_{\mathrm{test}} \wedge \square\lozenge\psi^{\mathrm{f}}_{\mathrm{test}}\Big),\end{aligned} \tag{2}$$

where $\varphi^{\mathrm{init}}_{\mathrm{sys}}$ is the initial condition of that the system needs to satisfy, $\square\varphi^{\mathrm{s}}_{\mathrm{sys}}$ encode system dynamics and safety requirements on the system, and $\square\lozenge\varphi^{\mathrm{f}}_{\mathrm{sys}}$ specifies recurrence goals for the system. Likewise, $\varphi^{\mathrm{init}}_{\mathrm{test}}$, $\square\varphi^{\mathrm{s}}_{\mathrm{test}}$, and $\square\lozenge\varphi^{\mathrm{f}}_{\mathrm{test}}$ represent assumptions the system has on the test environment. Additionally, the propositional formulas $\psi^{\mathrm{s}}_{\mathrm{test}}$ and $\square\lozenge\psi^{\mathrm{f}}_{\mathrm{test}}$ represent the safety and recurrence formulas of the test specification that is not known to the system. Note that the system and test specifications are in a form known as $GeneralReactivity(1)$ or $GR(1)$ formulas, for which efficient synthesis techniques are well-known [3].

Remark 1. In this paper, we consider a subset of the formulas described in equations (1) and (2) in which the subformulas representing initial conditions, safety and progress requirements are sets of atomic propositions in the set 2^{AP}.

C. Network Flows

We build on the concept of network flows to define a minimally constrained test. An extension of network flows to the game setting is known as a flow game [7].

Definition 3 (Network Flow)**.** A network flow 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.

D. Example

We will use the following as a running example throughout this paper. Consider a maze in a grid world setting. The system under test is an agent starting at its initial position on the bottom left corner of the grid with the intention of reaching its goal position in the top right corner. The dynamics are given as simple grid world dynamics enabling horizontal and vertical transitions to neighboring grid cells. The test behavior that we want to observe is that the system passes through specific grid cells, defined by the test specification. We then constrain the environment by placing obstacles on the grid cells by solving optimization (3).

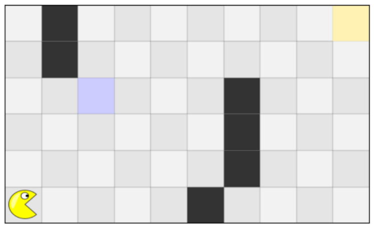

Fig. 2: Example maze layout, the agent (pacman) wants to reach its goal state (gold). The test specification wants to route its path through the intermediate state (blue).

III. Algorithms

A. Projecting Acceptance States from Büchi Automata

Suppose that we're given the system and test environment transition systems, $\mathcal{T}_{\text{sys}}$ and $\mathcal{T}_{\text{test}}$, and the system and test specifications, φ_{sys} and φ_{test}, respectively. Construct the product transition system $\mathcal{G} = \mathcal{T}_{\text{sys}} \times \mathcal{T}_{\text{test}}$, and construct the Büchi automata, $\mathcal{B}_{\text{sys}}$ and $\mathcal{B}_{\text{test}}$, corresponding to the system and test specifications, respectively. The start nodes S_n, intermediate nodes I_n, and target nodes T_n are vertices on $\mathcal{G}$ are of special importance. The start nodes S_n correspond to initial conditions of a test. The target nodes T_n correspond to states when the system specification φ_{sys} is satisfied, and the intermediate nodes I_n correspond to states when the test specification φ_{test} is satisfied. Algorithm 1 details how S_n, I_n, and T_n are determined.

Remark 2. The product transition system $\mathcal{G}$ is a 2-player turn-based game graph. For the system, the objective is to find a strategy that reaches (or repeatedly reaches) T_n on $\mathcal{G}$. Likewise, the test environment's objective is to find a strategy that passes through I_n before the state(s) T_n are reached.

Algorithm 1 Project Acceptance States on $\mathcal{G}$

```
1:  procedure $P(\mathcal{T}_{\text{sys}}, \varphi_{\text{sys}}, \mathcal{T}_{\text{test}}, \varphi_{\text{test}})$
2:      $\mathcal{G} \leftarrow \mathcal{T}_{\text{sys}} \times \mathcal{T}_{\text{test}}$    ▷ Game graph
3:      $\mathcal{B}_{\text{sys}} \leftarrow BA(\varphi_{\text{sys}}) = (Q_{\text{sys}}, 2^{AP}, \rightarrow_{\text{sys}}, Q_{0,\text{sys}}, F_{\text{sys}})$
4:      $\mathcal{B}_{\text{test}} \leftarrow BA(\varphi_{\text{test}}) = (Q_{\text{test}}, 2^{AP}, \rightarrow_{\text{test}}, Q_{0,\text{test}}, F_{\text{test}})$
5:      $\mathcal{G} \otimes \mathcal{B}_{\text{sys}} = (S'_{\text{sys}}, A_{\text{sys}}, \rightarrow'_{\text{sys}}, I'_{\text{sys}}, AP'_{\text{sys}}, L'_{\text{sys}})$
6:      $\mathcal{G} \otimes \mathcal{B}_{\text{test}} = (S'_{\text{test}}, A_{\text{test}}, \rightarrow'_{\text{test}}, I'_{\text{test}}, AP'_{\text{test}}, L'_{\text{test}})$
7:      $P_{\mathcal{G}}(I'_{\text{sys}}) = \{s \in S_{\mathcal{G}} | \exists q_{\text{sys}} \in Q_{\text{sys}} \text{ s.t. } (s, q_{\text{sys}}) \in I'_{\text{sys}}\}$
8:      $P_{\mathcal{G}}(I'_{\text{test}}) = \{s \in S_{\mathcal{G}} | \exists q_{\text{test}} \in Q_{\text{test}} \text{ s.t. } (s, q_{\text{test}}) \in I'_{\text{test}}\}$
9:      $S_n \leftarrow P_{\mathcal{G}}(I'_{\text{sys}}) \cap P_{\mathcal{G}}(I'_{\text{test}})$    ▷ Start nodes on $\mathcal{G}$
10:     $I_n \leftarrow \{s \in S_{\mathcal{G}} | \exists q_{\text{test}} \in Q_{\text{test}}, L'_{\text{test}}((s, q_{\text{test}})) \subseteq F_{\text{test}}\}$    ▷ Intermediate nodes on $\mathcal{G}$
11:     $T_n \leftarrow \{s \in S_{\mathcal{G}} | \exists q_{\text{sys}} \in Q_{\text{sys}}, L'_{\text{sys}}((s, q_{\text{sys}})) \subseteq F_{\text{sys}}\}$    ▷ Target nodes on $\mathcal{G}$
12:     return $S_n, I_n, T_n$
13: end procedure
```

Consider the product transition system $\mathcal{G} = (V, E)$ with sets of vertices: *start* S_n, *intermediate* I_n, and *target* T_n defined prior. For every edge $e \in E$, let $d_e \in \mathbb{B}$ be an indicator variable for whether edge e is cut, and likewise, for every vertex $v \in V$, let $d_v \in \mathbb{B}$ be an indicator variable for whether vertex v is removed from $\mathcal{G}$. The cuts d_e and d_v represent *node constraints* and *edge constraints* on $\mathcal{G}$, respectively. Let $Paths(S_n, T_n)$ denote all paths on $\mathcal{G}$ from source vertices S_n to target vertices T_n, and for some path $p \in Paths(S_n, T_n)$, let f_p denote the flow along path p. Now, we can properly define a minimally constrained test.

Definition 4 (Minimally Constrained Test)**.** A minimally constrained test, with respect to *node constraints*, is such that $\sum_{I_n \cap p = \emptyset} f_p = 0$ while $\frac{\sum_{v \in V} d_v}{\sum_{I_n \cap p \neq \emptyset} f_p}$ is minimized. Similarly, a minimally constrained test, with respect to *edge constraints*, is such that $\sum_{I_n \cap p = \emptyset} f_p = 0$ while $\frac{\sum_{e \in E} d_e}{\sum_{I_n \cap p \neq \emptyset} f_p}$ is minimized. Variable change from flow on paths to flow along edges is detailed in [9].

Problem 1. Given the transition systems $\mathcal{T}_{\text{sys}}$ and $\mathcal{T}_{\text{test}}$, and specifications φ_{sys} and φ_{test}, find the minimally constraint test with respect to node constraints and/or edge constraints.

B. Minimally Constrained Test: Node Constraints

We are given a game graph $\mathcal{G}$, which represents the allowed transitions through the maze grid, with the set of vertices V and the set of edges E. Now we find the minimum number of vertices to cut (i.e. obstacles to place) to constrain the flow through the designated intermediate vertex I. First we find a combination of paths from the source s to the goal g. If these are disjoint, i.e. do not share vertices (other than I) or have direct edges from a vertex in one path to a vertex in the other path - these vertices are designated as the set of critical vertices H. We then augment the graph by removing the vertex I, such that $\bar{\mathcal{G}} = \langle \bar{V}, E, c, s, t \rangle$, with $\bar{V} = V \backslash I$ and then find the minimum number of obstacles as follows:

$$\begin{aligned} \min_{x} \quad & \sum_{i=1}^{\bar{V}} x_i^2 \\ \text{s.t.} \quad & x_{\text{source}}^0 = 1, \quad x_{\text{goal}}^1 = 1, \\ & \sum_k x_i^k = 1, \quad \forall i \in \bar{V} \\ & x_j^2 = 0, \quad \forall j \in H \\ & x_i^0 = 1 \rightarrow x_j^1 = 0, \quad \forall i \in \bar{V}, (i,j) \in E \\ & x_i^1 = 1 \rightarrow x_j^0 = 0, \quad \forall i \in \bar{V}, (i,j) \in E \end{aligned} \tag{3}$$

where the vector X, which consists of the binary values x_i^k, $\forall i \in \bar{V}$ and $k \in \{0, 1, 2\}$. Intuitively, we partition the set of vertices V into three partitions, where 0 represents the vertices that can be transitioned before reaching I and 1 represents the vertices after reaching I. All vertices in partition 2 will be cut (obstacles will be placed) to separate partitions 0 and 1.

Remark 3. Solving for minimum node cut according to optimization (3) depends on identifying the correct combination

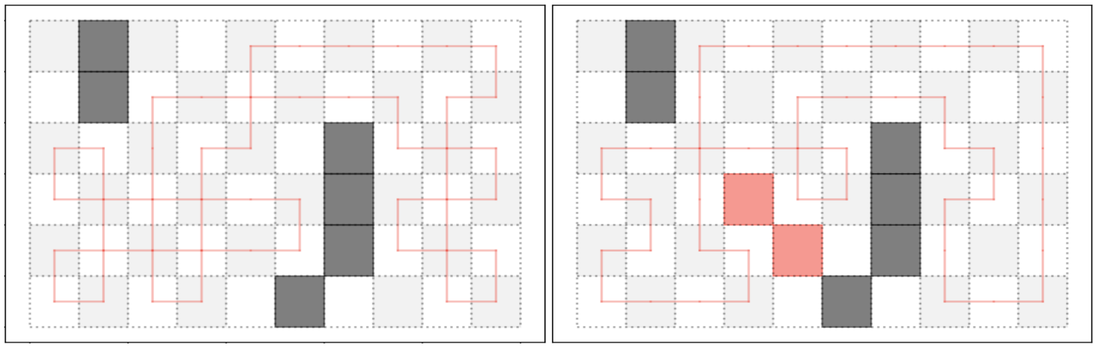
Fig. 3: Comparison of the flow through the maze from the initial position in the bottom left corner to the goal position in the top right corner without obstacles and with obstacles (red). By constraining the environment all flows now pass through the grid cell $(2, 2)$ as specified in φ_{test}.

of paths that ensure feasibility. In general, finding the right combination of paths is intractable due to the combinatorial nature of the problem.

C. Minimally Constrained Test: Edge Constraints

In this section, we address the issue of the previous remark by providing an convex-concave min-max optimization that does not require knowledge of the right combination of paths apriori. The test environment can be thought of as routing commodity-1 from S_n to I_n, and commodity-2 from I_n to T_n, while the system can be thought of as routing commodity-3 from S_n to T_n without passing through I_n. Thus, the system and the test environment can be thought of as two players with different objectives. This problem can thus be framed as a constrained min-max optimization problem with dependent feasible sets, a convex-concave min-max Stackleberg game [4]. The objective function of this game is the flow-cut gap and the max flow for commodity-3 on $\bar{\mathcal{G}}$. The resulting min-max Stackleberg game can be solved by the gradient-descent methods introduced in [4]. This convex-concave min-max optimization can be formalized as follows,

$$\begin{aligned}
\underset{f_e^1, f_e^2, F, d_e}{\text{argmin}} \ \underset{f_e^3}{\text{argmax}} \quad & \sum_{e \in E} \frac{d_e}{F} + \lambda \sum_{v:(s_3,v)\in E} \frac{f_e^3}{F} \\
\text{s.t.} \quad & 1 \leq \sum_{v:(s_1,v)\in E} \frac{f_e^1}{F}, \ 1 \leq \sum_{v:(s_2,v)\in E} \frac{f_e^2}{F}, \\
& 0 \leq \frac{d_e}{F} \leq \frac{1}{F}, \forall e \in E \\
& 0 \leq \frac{f_e^k}{F} \leq \frac{1}{F}, \forall e \in E, \ \forall k \in \{1,2,3\}, \\
& \frac{d_e}{F} + \frac{f_e^k}{F} \leq \frac{1}{F}, \forall e \in E, \ \forall k \in \{1,2,3\},
\end{aligned} \tag{4}$$

where $f_e^k \geq 0$ represents the flow along edge $e \in E$, and is defined for every $e \in E$ and for each of the three commodities $k = 1, 2, 3$. The variable d_e represents whether edge $e \in E$ is cut ($d_e = 1$) or not ($d_e = 0$), and the auxiliary variable $F \geq 0$ represents the maximum of the commodity-1 flow and commodity-2 flow. The regularization parameter λ places weight on commodity-3 flow. The *max*-player is the system, which tries to maximize commodity-3 flow from S_n to T_n, while the *min*-player is the test environment, which optimizes to cut off commodity-3 flow while achieving a sparse cut of edges. The conservation and capacity constraints are not listed above but apply to each vertex and edge respectively. Note that a similar formulation could be extended to minimally constrained tests with node constraints.

IV. Conclusions and Future Work

We outlined the problem of finding the minimally constrained test specification as a min-max Stackleberg optimization. For future work, we would like to extend this algorithm to more general $GR(1)$ formulas, and provide sub-optimality guarantees on the generated test environment. We want to illustrate minimally constrained tests identified by our algorithms on both gridworld and road network examples, with both static and dynamic test environments. Furthermore, we aim to extend this framework to include dynamic test agents to find the minimum number of test agents required to constrain a test environment for a test specification.

Relevance to workshop theme: The framework presented in this work-in-progress paper are for automated generation of test environments. In particular, the cuts (node cuts or edge cuts) represent constraints that the test environment places on the system under test. In the test environment, these constraints can be realized by placing obstacles, both *physical* and *virtual*, to prompt the system to successfully pass the test in order to meet its requirements. In the physical space, these obstacles restrict regions of space that the robot can physically occupy. The virtual constraints make up a test harness which allows for testing whether the system can satisfy its requirements even when some software functionality is unavailable. This inspires the question of what infrastructure, both *hardware* and *software* test harnesses, is necessary for testing certain classes of requirements used for specifying planning and control modules in robotics?

Acknowledgments

The authors would like to acknowledge Professor Eric Mazumdar, Professor Mani Chandy, Professor Tichakorn Wongpiromsarn, Professor Joel Burdick, Professor Leonard Schulman, Dr. Shih-Hao Tseng, Dr. Ioannis Filippidis, and Dr. Ugo Rosolia for insightful discussions. We acknowledge funding from AFOSR Test and Evaluation Program, grant FA9550-19-1-0302 and National Science Foundation award CNS-1932091.

References

[1] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram 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*, pages 254–257. Springer, 2011.

[2] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[3] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Saar. Synthesis of reactive (1) designs. *Journal of Computer and System Sciences*, 78(3):911–938, 2012.

[4] Denizalp Goktas and Amy Greenwald. Convex-concave min-max stackelberg games. *Advances in Neural Information Processing Systems*, 34, 2021.

[5] Josefine B Graebener, Apurva Badithela, and Richard 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*, pages 133–155, 2022.

[6] James Kapinski, Jyotirmoy V Deshmukh, Xiaoqing Jin, Hisahiro Ito, and Ken 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*, 36 (6):45–64, 2016.

[7] Orna Kupferman, Gal Vardi, and Moshe Y Vardi. Flow games. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[8] Sriram Sankaranarayanan and Georgios 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*, pages 125–134, 2012.

[9] Vijay V Vazirani. *Approximation algorithms*, volume 1. Springer, 2001.