Chapter 4

FLOW-BASED REACTIVE TEST SYNTHESIS

The previous chapter introduced the problem of automated test strategy synthesis based on network flow optimization for user-specified temporal logic objectives. In this chapter, we will consider an bigger class of temporal logic objectives, and propose an automated test synthesis framework rooted in automata theory and flow networks. Especially, the routing optimization will be reformulated using two flow-based optimizations: i) a min-max Stackelberg game with coupled constraint sets, and ii) a mixed-integer linear programming formulation. The second flow-based reformulation lends itself to tractable implementations. Additionally, we study how these automatically found test strategies can be used to synthesize a strategy for a dynamic test agent.

4.1 Introduction

In this chapter, we will expand the class of temporal logic objectives to include reachability, avoidance, and reaction sub-tasks that commonly occur in high-level specifications of robotic missions [81]. A test strategy is *feasible* if a well-designed system can succeed in the test. We will formalize notions of *feasibility* and *restrictiveness* of a test strategy to handle these expanded class of specifications. Furthermore, in addition to the previous chapter, we will formally present the assumptions and guarantees that the system places on its test environment.

Next, we revisit the routing problem for the expanded class of specifications, and

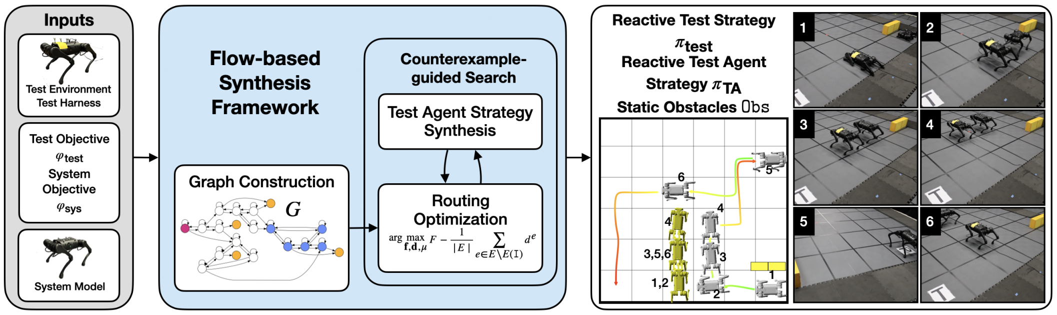


Figure 4.1: Overview of the flow-based test synthesis framework which consists of three key parts: i) graph construction, ii) routing optimization, and iii) test environment synthesis (e.g., reactive test strategy / test agent strategy, static obstacles).

use automata theory to construct a product graph representing system state evolution along with progress of the temporal logic objectives. We formulate the routing problem on this product graph, first as a special class of Stackelberg games, and then as a mixed-integer linear program. We will motivate the mixed-integer formulation from the drawbacks of the game formulation. Using the MILP formulation, we can automatically find a test strategy for different environment types: static obstacles, reactive obstacles, and dynamic test agents. Even feasible solutions of the MILP return test strategies that satisfy the temporal logic objectives, and optimal solutions are guaranteed to not be overly-restrictive. Moreover, this routing optimization is proven to be NP-hard in the size of the product graph, thus supporting the MILP formulation. Finally, given a test agent, we are able to match the solution of the MILP to synthesize a strategy for the test agent. We use a simple counerexample-guided approach to ensure that the MILP solutions are dynamically feasible for the test agent.

Finally, the test synthesis framework is demonstrated on simulated grid world settings and on hardware with a pair of quadrupedal robots. For all experiments, our framework synthesizes test strategies that place the fewest possible restrictions on the system over the course of the test either by obstacle placement or a dynamic agent. In experiments with reactive obstacles and dynamic agents, the reactive test strategy results in a different test execution depending on system behavior. Despite this, the system is always routed through the test objective (e.g., being put in low-fuel state or having to walk over challenging terrain).

This chapter is adapted from:

J. B. Graebener*, A. S. Badithela*, D. Goktas, W. Ubellacker, E. V. Mazumdar, A. D. Ames, R. M. Murray (2024). “Flow-Based Synthesis of Reactive Tests for Discrete Decision-Making Systems with Temporal Logic Specifications”. arXiv preprint https://arxiv.org/abs/2404.09888 (In submission to Transactions on Robotics).

A. Badithela*, J. B. Graebener*, W. Ubellacker, E. V. Mazumdar, A. D. Ames, R. M. Murray. (2023). “Synthesizing Reactive Test Environments for Autonomous Systems: Testing Reach-Avoid Specifications with Multi-Commodity Flows.” In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 12430–12436. DOI: 10.1109/ICRA48891.2023.10160841.

4.2 Related Work

Following the previous chapter, we synthesize test environments from LTL specifications. We generate tests without specific knowledge of the system controller such that the test environment *adapts* or *reacts* to system behavior at runtime. Our test synthesis framework requires knowledge of a nondeterministic model of the system, but is agnostic to the high-level controller of the system, and is completely black-box to models and controllers at lower levels of abstraction.

Reactive, specification-based testing over discrete logics has been studied in [82–86]. In [82], reactive synthesis [87] is used to find a test strategy from LTL specifications of the system and a user-defined fault model, with guarantees that the resulting test trace will show the fault if the system implementation is faulty with respect to the fault model. However, this method requires fault models to be carefully specified over the output states of the system. Though very beneficial for specifying and catching sub-system level faults, it becomes intractable for specifying complex system-level faults, especially when the set of output states is large. The test synthesis framework in this chapter is also specification-based and adaptive to system behavior, but we specify desired test behavior in terms of system and test objectives. Additionally, the procedure in [82] does not account for the freedom of the system to satisfy its own requirements. In this chapter, we will synthesize reactive test strategies that demonstrate the test objective while placing minimal restrictions on the system. The automata-theoretic tools used in this chapter build on concepts used in correct-by-construction synthesis and model checking [60, 88], and will be covered in the next section.

Game-based formulations of testing either presume entirely cooperative or entirely adversarial settings. In [89], testing of reactive systems was formalized as a game between two players, where the tester and the system try to reveal and hide faults, respectively. In [90], the test strategy is found by optimizing for reachability and coverage metrics over a game modeling the system and the tester. Test case generation in cooperative settings is studied in [91, 92]. However, the test synthesis problem considered here is neither fully *adversarial* nor fully *cooperative*; a well-designed system is cooperative with the test environment in realizing the system objective, but since the system is agnostic to test objective, it need not cooperate with the test environment in realizing it.

In this chapter, in addition to static obstacles that restrict the system throughout the test, we will also consider reactive obstacles, and a dynamic test agent that is reactive to system behavior at runtime. Leveraging network flows, we will first pose the test synthesis problem as a Stackelberg game, and then present a more efficient formulation as an MILP. In recent years, network flow optimization frameworks with tight convex relaxations have led to massive computational speed-ups in solving robot motion planning problems [93, 94]. Network flow-based mixed integer programs have also been to synthesize playable game levels in video games [95], which was then applied to construct playable scenarios in robotics settings [96].

4.3 Preliminaries

In this section, we will revisit concepts from automata theory, and build on the background of automata theory and flow networks introduced in Chapter 3.

Consider the finite transition system TS, introduced in previous chapter. Once again, we use LTL to describe the system and test objectives. However, we place the following additional constraint on the system, requiring it to have at least one terminal state, to simplify the test objective that we synthesize for.

Definition 4.1 (System)**.** The *system under test* is modeled as a finite transition system T_{sys} with a single initial state, that is, $|T_{\text{sys}}.S_0|= 1$. Furthermore, at least one of the system states is terminal (i.e., no outgoing edges).

The system designers provide the states S, actions A, transitions δ, and a set of possible initial conditions S_0, set of atomic propositions, AP_{sys} and a corresponding label function $L_{\text{sys}} : S \to 2^{AP_{\text{sys}}}$. We require a unique initial condition $s_0 \in S_0$ to synthesize the test. If the test designer wishes to select an initial condition, then they can synthesize the test for each $s_0 \in S_0$ and choose accordingly. In addition to AP_{sys}, the test designer can choose additional atomic propositions AP_{test} and define a corresponding labeling function $L : S \to 2^{AP}$, where $AP := AP_{\text{sys}} \cup AP_{\text{test}}$. For test synthesis, the system model is $T_{\text{sys}} = (S, A, \delta, \{s_0\}, AP, L)$ is defined for the specific initial condition s_0 chosen by the test designer. The terminal state is used for defining test termination when the system satisfies its objective.

Assumption 4.1. Except for sink states, transitions between states of the system are bidirectional: $\forall(s, s') \in T_{\text{sys}}.E$ where s' is not a terminal state, we also have $(s', s) \in T_{\text{sys}}.E$.

This assumption is for a simpler presentation, and the framework can be extended to transition systems without this assumption (see Remark 4.8).

Definition 4.2 (Test Environment)**.** The *test environment* consists of one or more of the following: static obstacles, reactive obstacles, and dynamic test agents. A *static obstacle* on $(s, s') \in T_{\text{sys}}.E$ is a restriction on the system transition (s, s') that remains in place for the entire duration of the test. A *reactive obstacle* on $(s, s') \in T_{\text{sys}}.E$ is a temporary restriction on the system transition (s, s') that can be enabled/disabled over the course of the test. A *dynamic test agent* can occupy states in $T_{\text{sys}}.S$, thus restricting the system from entering the occupied state.

The desired test behavior can be captured via sub-tasks that are defined over atomic propositions AP. Table 4.1 lists the sub-task specification patterns that are considered. These specification patterns are commonly used to specify robotic missions [81]. The desired test behavior is characterized by the system and test objectives, defined over the set of atomic propositions AP that can be evaluated on system states $T_{\text{sys}}.S$.

Table 4.1: Sub-task specification patterns defined over atomic propositions.

Name	Formula	
Visit	$\bigwedge_{i=1}^{m} \Diamond p^i$	(s1)
Sequenced Visit	$\Diamond(p^0 \wedge (\Diamond(p^1 \wedge \ldots \Diamond p^m)))$	(s2)
Safety	$\Box \neg p$	(s3)
Instantaneous Reaction	$\Box(p \rightarrow q)$	(s4)
Delayed Reaction	$\Box(p \rightarrow \Diamond q)$	(s5)

Definition 4.3 (Test Objective)**.** The *test objective* φ_{test} comprises of at least one visit or sequenced visit sub-task or a conjunction of these sub-tasks. The Büchi automaton $\mathcal{B}_{\text{test}}$ corresponds to the test objective φ_{test}.

Definition 4.4 (System Objective)**.** The *system objective* φ_{sys} contains at least one visit or sequenced visit sub-task. In addition, it can also contain some conjuction of safety, instantaneous and/or delayed reaction, and visit and/or sequenced visit sub-tasks. The Büchi automaton $\mathcal{B}_{\text{sys}}$ corresponds to the system objective φ_{sys}. We say that the *system reaches its goal*, or the *system execution is successful*, if the system trace is accepted $\mathcal{B}_{\text{sys}}$.

Typically, some aspects of a test are not revealed to the system until test time such as testing the persistence of a robot or prompting it to exhibit a difficult maneuver by placing obstacles in its path. This is formalized as a test objective and is not known to the system. In contrast, the system is aware of the system objective, which captures its requirements. For example, to test for *safety*, the system should know to avoid unsafe areas (4.3). To test a *reaction*, $\square(p \rightarrow q)$, the system needs to be aware of the reaction requirement (4.4), and the test objective needs to contain the corresponding visit requirement $\diamondsuit p$ to trigger the reaction. Furthermore, the test objective can contain standalone reachability (visit and/or sequenced visit) sub-tasks that are not associated with a system reaction sub-task, but require the system to reach/visit certain states. The test objective is accomplished by restricting system actions in reaction to the system state via the test harness.

In addition to the system objective, the system must interact safely with the test environment. The system must also obey the initial condition set by the test designer. For each obstacle/agent of the test environment, the system controller must respect the corresponding restrictions on its actions (i.e., cannot crash into obstacles/agents). Furthermore, for a valid system implementation, all lower-level planners and controllers of the system must simulate transitions on T.

Definition 4.5 (System Guarantees)**.** The system guarantees are a conjunction of the system objective, initial condition, safe interaction with the test environment, and a system implementation respecting the model T_{sys}.

Definition 4.6 (System Assumptions)**.** The system *assumes* that the test environment satisfies the following conditions:
A1. The test environment can consist of: i) static obstacles (e.g., wall), ii) reactive obstacles (e.g., door), and iii) test agents whose dynamics are provided to the system.
A2. The test environment will not take any action that will inevitably lead to unsafe behavior (e.g., not restricting a system action after the system has committed to it, test agents not colliding into the system).
A3. The test environment will not take any action that will inevitably block all paths for the system to reach its goal (e.g., restrictions will not completely the enclose the system or block it from progressing to its goal).
A4. If the system and test environment are in a livelock, the system will have the option to break the livelock and take a different path toward its goal.

A *correct system strategy* satisfies the system guarantees when the test environment satisfies the system assumptions. Therefore, a correct system strategy would result in a successful system execution. The system's specification cannot be expressed as an LTL formula. This is because, in an LTL synthesis setting, the system can assume that the test harness can behave in a worst-case manner and will never synthesize a satisfying controller. However, the system can assume that the test harness will always ensure that a path to achieving the system specification remains. This existence of a satisfying path cannot be easily captured in an LTL assumption.

Now, we will cover background on Büchi automata from the system and test objectives, and its usefulness for constructing a product graph that tracks both the evolution of the system state as well as the automaton state as it makes progress in satisfying its specifications.

Definition 4.7 (Deterministic Büchi Automaton)**.** A *non-deterministic Büchi automaton* (NBA) [60, 97] is a tuple $\mathcal{B} := (Q, \Omega, \delta, Q_0, F)$, where Q denotes the states, $\Omega := 2^{AP}$ is the set of alphabet for the set of atomic propositions AP, $\delta : Q \times \Sigma \to Q$ denotes the transition function, $Q_0 \subseteq Q$ represents the initial states, and $F \subseteq Q$ is the set of acceptance states. The automaton is a *deterministic Büchi automaton* (DBA) iff $|Q_0| \leq 1$ and $|\delta(q, A)| \leq 1$ for all $q \in Q$ and $A \in \Omega$.

Remark 4.1. We use *deterministic* Büchi automata since each input word corresponding to a test execution should have a unique run on the automaton. While there are several different automata representations, deterministic Büchi automata are a natural choice for many LTL specifications.

Since the objectives are reach-avoid specifications and do not encode behaviors that occur "infinitely often", *deterministic finite automata* (DFAs) [60] would have sufficed. The intuition behind this is that we are only using the automata for tracking the automaton state on the product graph (see the following paragraphs on graph construction). Using Büchi automata was an implementation choice, and to leave the possibility for expanding to objectives that can only be characterized by DBAs and not DFAs. The tool Spot [98] was used to construct a deterministic Büchi automaton from an LTL formula, which had excellent documentation that made it easy to access and use. LTL to DFA tools are not as common, though there are a few tools that translate LTL formulas on finite traces (LTLf) to DFAs such as LTL_f2DFA [99] and Lisa [100]. Both tools rely on MONA [101], which translates LTL formulas to finite-state automata; Lisa additionally also depends on the DBA conversion from Spot to return a DFA.

To track progress with respect to the system and test objectives, we introduce the specification product, which will be used to construct the product graph.

Definition 4.8 (Specification Product)**.** A *product* of two Büchi automata, $\mathcal{B}_1$ and $\mathcal{B}_2$ over the alphabet Ω, is defined as $\mathcal{B}_1 \otimes \mathcal{B}_2 := (Q, \Omega, \delta, Q_0, F)$, with states $Q := \mathcal{B}_1.Q \times \mathcal{B}_2.Q$, initial state $Q_0 := \mathcal{B}_1.Q_0 \times \mathcal{B}_2.Q_0$, acceptance states $F := \mathcal{B}_1.F \times \mathcal{B}_2.F$. The transition relation δ is defined as follows, for all $(q_1, q_2) \in Q$, for all $A \in \Omega$, $\delta((q_1, q_2), A) = (q_1', q_2')$ if $\mathcal{B}_1.\delta(q_1, A) = q_1'$ and $\mathcal{B}_2.\delta(q_2, A) = q_2'$. The *specification product* is the product $\mathcal{B}_\pi := \mathcal{B}_{\text{sys}} \otimes \mathcal{B}_{\text{test}}$, where $\mathcal{B}_{\text{sys}}$ is the Büchi automaton corresponding to the system specification, and $\mathcal{B}_{\text{test}}$ is the Büchi automaton corresponding to the test objective. The states $(q_{\text{sys}}, q_{\text{test}}) \in \mathcal{B}_\pi.Q$, where $q_{\text{sys}} \in \mathcal{B}_{\text{sys}}$ and $q_{\text{test}} \in \mathcal{B}_{\text{test}}$, capture the event-based progression of the test and are referred to as history variables.

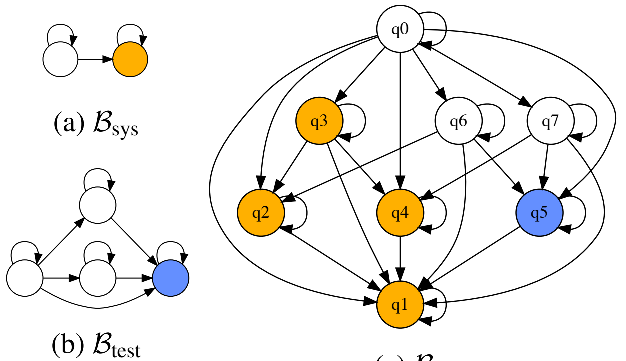
The system reaching its goal would typically mark the end of a test execution. However, the test engineer can also decide to terminate the test if the system appears to be stuck or enters an unsafe state. We assume that the test engineer gives the system a reasonable amount of time to complete the test. Upon test termination in state s_n, we augment the trace σ with the infinite suffix s_n^ω for evaluation purposes.

Remark 4.2. As tests have a defined start and end point, we need to bridge the gap between the finiteness of test executions and the infinite traces that are needed to evaluate LTL formulae. Augmenting the trace with the infinite suffix allows us to leverage useful tools available for LTL.

Remark 4.3. The states of the specification product automaton track the states of the individual Büchi automata, $\mathcal{B}_{\text{sys}}$ and $\mathcal{B}_{\text{test}}$, in the form of the Cartesian product to remember accepting states of the individual automata, which will be necessary for our framework (see Definitions 4.8, 4.15).

We use the synchronous product operator to construct a product of a transition system and a Büchi automaton. In particular, we will use this operator to construct the virtual product graph and the system product graph (see Section 4.4).

Definition 4.9 (Synchronous Product)**.** The *synchronous product* of a DBA $\mathcal{B}$ and a FTS T_{sys}, where the alphabet of $\mathcal{B}$ is the labels of T_{sys}, is the transition system


(a) $\mathcal{B}_{\text{sys}}$
(b) $\mathcal{B}_{\text{test}}$
q0
q3
q6
q7
q2
q4
q5
q1

(c) $\mathcal{B}_\pi$

Figure 4.2: Automata for Example 4.2. Yellow ● and blue ● nodes in $\mathcal{B}_{\text{sys}}$ and $\mathcal{B}_{\text{test}}$ are the respective accepting states. In the product $\mathcal{B}_\pi$, we continue to track these states for the system and test objectives. States in the product $\mathcal{B}_\pi$ that are accepting to both objectives (e.g., q1) are also shaded yellow.

$P \coloneqq T_{\text{sys}} \otimes \mathcal{B}$, where:

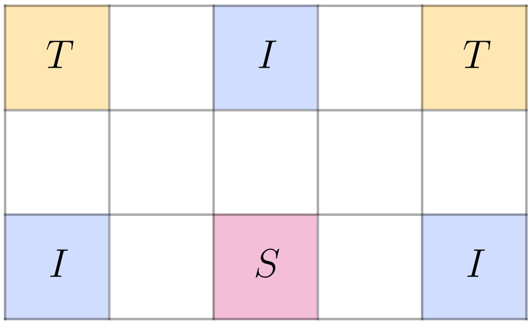
$$\begin{aligned}
&P.S \coloneqq T_{\text{sys}}.S \times \mathcal{B}.Q,\\
&P.\delta((s,q),a) \coloneqq (s',q') \text{ if } \forall s,s' \in T_{\text{sys}}.S, \forall q,q' \in B.Q,\\
&\quad \exists a \in T_{\text{sys}}.A, \text{ s.t. } T_{\text{sys}}.\delta(s,a) = s' \text{ and } \mathcal{B}.\delta(q, T_{\text{sys}}.L(s')) = q',\\
&P.S_0 \coloneqq \{(s_0,q) \,|\, s_0 \in T_{\text{sys}}.S_0,\ \exists q_0 \in \mathcal{B}.Q_0 \text{ s.t.}\\
&\quad \mathcal{B}.\delta(q_0, T_{\text{sys}}.L(s_0)) = q\},\\
&P.AP \coloneqq \mathcal{B}.Q,\\
&P.L((s,q)) \coloneqq \{q\}, \quad \forall (s,q) \in P.S.
\end{aligned}$$

We denote the transitions in P as

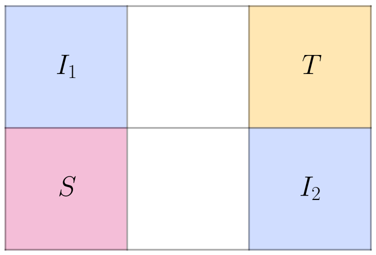
$$P.E \coloneqq \{(s,s') \,|\, s,s' \in P.S \text{ if } \exists a \in P.A \text{ s.t. } P.\delta(s,a) = s'\}. \tag{4.6}$$

An infinite sequence on P corresponds to a state-history trace $\vartheta = (s_0,q_0),\dots,(s_n,q_n)^\omega$. We refer to $(s,q) \in P.S$ as the state-history pair and define the corresponding path to be the finite prefix: $\vartheta_{fin} = (s,q)_0,(s,q)_1,\dots,(s,q)_n$.

Example 4.1. The system under test can transition (N-S-E-W) on the grid world as illustrated in Fig. 4.3a. The initial condition of the system is marked by S, and the system is required to visit one of the goal states marked by T, $\varphi_{\text{sys}} = \Diamond T$. The test objective is to observe the system visit at least one of the I states before the system reaches its goal, encoded as $\varphi_{\text{test}} = \Diamond I$.



(a) Example 4.1.



(b) Example 4.2.

Figure 4.3: Grid world layouts for examples.

Example 4.2. In this example, the system under test can transition (N-S-E-W) on the grid world as illustrated in Fig. 4.3b. The initial condition of the system is marked by S, and the system objective is to visit T, $\varphi_{\text{sys}} = \diamondsuit T$. The test objective is to observe the system visit states I_1 and I_2: $\varphi_{\text{test}} = \diamondsuit I_1 \wedge \diamondsuit I_2$.

4.4 Problem Statement

For the improved class of specifications, the test environment synthesis problem can be stated as follows. Assuming that a test engineer specifies the desired test behavior (i.e., system and test objectives), we seek to synthesize a reactive test strategy under which every successful system execution will also be a successful test execution — every system trace that satisfies the system objective will also satisfy the test objective. The reactive test strategy restricts system actions that are available from the test harness. Formally, a reactive test strategy is defined as follows.

Definition 4.10 (Reactive Test Strategy)**.** A *reactive test strategy* $\pi_{\text{test}} : (T_{\text{sys}}.S)^* T_{\text{sys}}.S \rightarrow 2^{A_H}$ defines the set of restricted system actions at the current system state based on the prefix of the system trace up to the current state. For some finite prefix $\sigma_{0:k} = s_0, \ldots, s_k$ starting from the system initial state $s_0 \in T_{\text{sys}}.S_0$, $\pi_{\text{test}}(\sigma_{0:k}) \subseteq H(s_k)$ is the set of restricted system actions from state s_k. A test environment *realizes* the strategy π_{test} if it restricts system actions according to π_{test}. The resulting system trace is denoted as $\sigma(\pi_{\text{sys}} \times \pi_{\text{test}})$.

More concretely, for some finite prefix $s_0, \ldots, s_i$ of a system execution σ starting from an initial state $s_0 \in T.S_0$, $\pi_{\text{test}}(s_0, \ldots, s_i) \subseteq H(s_i)$ denotes the set of actions unavailable to the system from state s_i of execution σ. These actions can be restricted by the test environment via static and reactive obstacles, and a dynamic test agent. The reactive test strategy must be such that it respects the system assumptions **A1–A4**. In turn, a correct system strategy must choose from actions available following the restrictions placed by the test environment. Formally, suppose $\Sigma_{\text{fin}} := (T_{\text{sys}}.S)^* T_{\text{sys}}.S$ be the set of all possible finite trace prefixes for the system, and at each time step $k \geq 0$, the system strategy $\pi_{\text{sys}} : \Sigma_{\text{fin}} \to T_{\text{sys}}.A \setminus \pi_{\text{test}}(\Sigma_{\text{fin}})$ must pick from unrestricted system actions from its current state.

By use of obstacles and/or test agents, the test environment externally blocks system transitions, and the system must correctly observe these obstacles and recognize the corresponding actions to be unsafe. We assume that the system can observe all restricted actions on its current state before it commits to an action, and therefore, a correct system strategy π_{sys} must choose from the available actions at each time step. Depending on the implemnentation, the system might have to re-plan its strategy π_{sys} in response to obstacles placed by the test environment.

Definition 4.11 (Feasibility of a Test Strategy)**.** Given a test environment, system T_{sys}, system and test objectives, φ_{sys} and φ_{test}, a reactive test strategy π_{test} is said to be *feasible* iff: i) the test environment can realize π_{test}, ii) there exists a correct system strategy π_{sys}, and iii) any execution corresponding to a correct π_{sys} satisfies the system and test objectives: $\sigma(\pi_{\text{sys}} \times \pi_{\text{test}}) \vDash \varphi_{\text{test}} \wedge \varphi_{\text{sys}}$.

The reactive test strategy does not help the system in achieving the system objective φ_{sys}; it only restricts the system such that the test objective can be realized. The system can choose an incorrect strategy π_{sys}, and in such a case, we cannot provide any guarantees. Furthermore, in routing the system to the test objective, it would be ideal if the test strategy does not overly restrict the system for the system to demonstrate decision-making when given the freedom to choose from multiple possible actions, including those that might be unsafe or lead the system to a livelock. For this reason, we will revisit the notion of the restrictiveness of tests defined over test executions. Given any system trace σ, every finite prefix $\sigma_{0:k} = s_0, \ldots, s_k$ maps to some history variable $q \in \mathcal{B}_\pi.Q$. Therefore, we can track this history variable along with the evolution of the system in a state-history trace $\vartheta = (s_0, q_0), (s_1, q_1), \ldots$, where the history variable q_k corresponds to the finite prefix $\sigma_{0:k}$. From now on, we refer to ϑ as the test execution, and proceed to define the restrictiveness of a test strategy in terms of the number of possible test executions. In the previous chapter, restrictiveness of a test was only defined on the system trace. Since we now have a broader class of specifications and use Büchi automata to track temporal events, we will define restrictiveness over test executions.

Definition 4.12 (Restrictiveness of a Test Strategy)**.** State-history traces ϑ_1 and ϑ_2 are *unique* if they do not share any consecutive state-history pairs — any two state-history pairs (s, q) and (s', q') do not appear in consecutive time steps in both ϑ_1 and ϑ_2. For a feasible π_{test}, let Σ be the set of all executions corresponding to correct system strategies, and let Θ be the set of all state-history traces corresponding to Σ. Let $\Theta_u \subseteq \Theta$ be a set of unique state-history traces. A test strategy π_{test} is *not overly restrictive* if the cardinality of Θ_u is maximized.

Remark 4.4. The set of all state history traces Θ can be infinite. However, the set Θ_u is finite because: i) the system has a finite number of states and the specification product has a finite number of history variables, and ii) every state-history trace in Θ_u is *unique* with respect to any other trace in Θ_u.

Problem 4.1 (Finding a Test Strategy)**.** Given a high-level abstraction of the system model T, test harness H, system objective φ_{sys}, test objective φ_{test}, find a feasible, reactive test strategy π_{test} that is not overly-restrictive.

To realize the test strategy, the test environment can place obstacles and use dynamic agents to restrict actions of the system. In the case of dynamic agents, the agent strategy must be found such that it *simulates* the restrictions set forth by the test strategy.

Problem 4.2 (Reactive Test Agent Strategy Synthesis)**.** Given a high-level abstraction of the system model T, test harness H, system objective φ_{sys}, test objective φ_{test}, and a test agent modeled by transition system T_{TA}. Find the test agent strategy π_{TA} and the set of static obstacles $\texttt{Obs}_{\text{static}}$ that: i) satisfy the system's assumptions on its environment, and ii) realize a reactive test strategy π_{test} that is not overly-restrictive and feasible. If the test agent cannot realize at least one reactive test strategy π_{test} that is not overly-restrictive, then find the π_{TA} that realizes the best possible π_{test}.

4.5 Graph Construction

To reason about executions of the system in relation to the system and test objectives, automata theory is leveraged to construct the following product graphs.

Definition 4.13 (Virtual Product Graph and System Product Graph)**.** A *virtual product graph* is the product transition system $G := T_{\text{sys}} \otimes \mathcal{B}_\pi$. Similarly, the system product graph is defined as $G_{\text{sys}} := T_{\text{sys}} \otimes \mathcal{B}_{\text{sys}}$.

The virtual product graph G tracks the test execution in relation to both the system and test objectives while the system product graph G_{sys} tracks the system objective. We will find the restrictions on system actions on G, while G_{sys} represents the system's perspective concerning the system objective during the test execution. For each node $u = (s, q) \in G.S$, we denote the corresponding state in $s \in T_{\text{sys}}.S$ as $u.s := s$. Similarly, the state corresponding to $v \in G_{\text{sys}}.S$ is denoted by $v.s := s$. For practical implementation, we remove nodes on the product graphs that are not reachable from the corresponding initial states, $G.S_0$ or $G_{\text{sys}}.S_0$.

Definition 4.14 (Projections)**.** States from G to G_{sys} are related via the *projection* map $\mathcal{P}_{G\to G_{\text{sys}}} : G.S \to G_{\text{sys}}.S$ as

$$\mathcal{P}_{G\to G_{\text{sys}}}((s, (q_{\text{sys}}, q_{\text{test}}))) = (s, q_{\text{sys}}). \tag{4.7}$$

These projections help us to reason about how restrictions found on G map to the system T_{sys} and the system product graph G_{sys}. We can now define the edges on G that we can restrict with the test harness as follows,

$$\begin{aligned} E_H = \{((s,q),(s',q')) \in G.E \,|\, &\forall s \in T_{\text{sys}}.S, \\ &\forall a \in H(s) \text{ s.t. } s' = T_{\text{sys}}.\delta(s,a)\}. \end{aligned} \tag{4.8}$$

Lemma 4.1. *For every path $(s, q_{\text{sys}})_0, (s, q_{\text{sys}})_1, \ldots, (s, q_{\text{sys}})_n$ on G_{sys}, there exists at least one corresponding path on G.*

Proof. Suppose there exists some $q_{\text{test}\,0}, \ldots, q_{\text{test}\,n} \in \mathcal{B}_{\text{test}}.Q$ such that $(s, (q_{\text{sys}}, q_{\text{test}}))_0, \ldots, (s, (q_{\text{sys}}, q_{\text{test}}))_n$ is a path on G. Then, by construction, there exists a path on G_{sys} where $(s, (q_{\text{sys}}, q_{\text{test}}))_k$ maps to $(s, q_{\text{sys}})_k$ for all $0 \le k \le n$. □

Paths on the virtual product graph G correspond to possible test executions. This is illustrated in Figures 4.4 and 4.5 for the example 4.2. We identify the nodes on G that capture the acceptance conditions for the system and test objectives.

Definition 4.15 (Source, Intermediate, and Target Nodes)**.** The *source node* S represents the initial condition of the system. The *intermediate nodes* I correspond to system states in which the test objective acceptance conditions are met. Finally, the *target nodes* T represent the system states for which the acceptance condition for the system objective is satisfied. Formally, these nodes are defined as follows,

$$\begin{aligned} &\mathtt{S} := \{(s_0, q_0) \in G.S \,|\, s_0 \in T_{\text{sys}}.S_0, q_0 \in \mathcal{B}_\pi.Q_0\}, \\ &\mathtt{I} := \{(s, (q_{\text{sys}}, q_{\text{test}})) \in G.S \,|\, q_{\text{test}} \in \mathcal{B}_{\text{test}}.F,\ q_{\text{sys}} \notin \mathcal{B}_{\text{sys}}.F\}, \\ &\mathtt{T} := \{(s, (q_{\text{sys}}, q_{\text{test}})) \in G.S \,|\, q_{\text{sys}} \in \mathcal{B}_{\text{sys}}.F\}. \end{aligned}$$

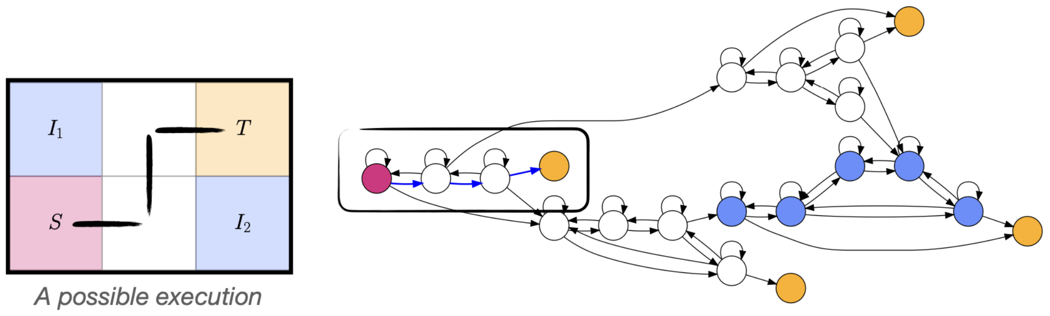


Figure 4.4: A possible execution of the system for Example 4.2 as illustrated on the transition system T_{sys} and the corresponding product graph G.

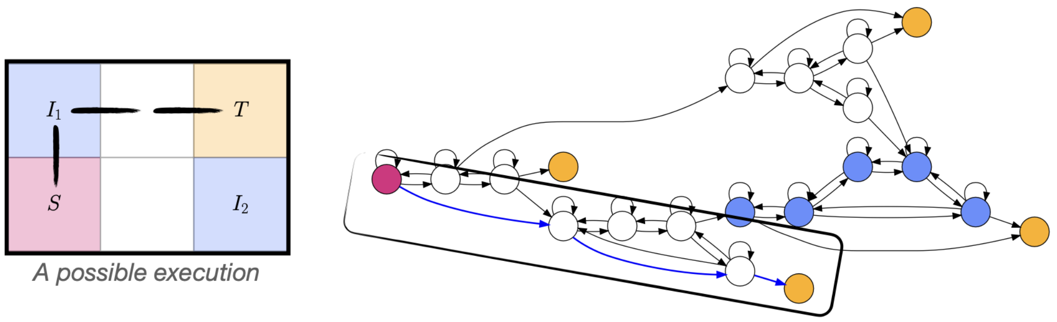


Figure 4.5: A possible execution of the system for Example 4.2 as illustrated on the transition system T_{sys} and the corresponding product graph G.

In addition, we define the set of states corresponding to the system acceptance condition on G_{sys} as $\mathtt{T}_{\text{sys}} := \{(s,q) \in G_{\text{sys}}.S \mid q \in \mathcal{B}_{\text{sys}}.F\}$.

Proposition 4.1. Every test execution corresponds to a path $\vartheta_n = (s,q)_0, \ldots, (s,q)_n$ on G where $(s,q)_0 \in \mathtt{S}$. The corresponding system trace σ_n satisfies the system objective, $\sigma \models \varphi_{\text{sys}}$ iff $(s,q)_n \in \mathtt{T}$. Furthermore, if $\sigma \models \varphi_{\text{test}}$, then the path ϑ_n contains a state-history pair $(s,q)_i \in \mathtt{I}$ for some $0 \leq i \leq n$.

Provided that there exists a path on G from $\mathtt{S}$ to $\mathtt{T}$, identifying a feasible reactive test strategy corresponds to identifying edges to cut on G. These edge cuts correspond to restricted system actions. In particular, these edge cuts are such that all paths on G from source $\mathtt{S}$ to target $\mathtt{T}$ visit the intermediate $\mathtt{I}$. Now, we will go over two new flow-based formulations to solve the routing problem.

4.6 Part I: Flow-based Optimization via Min-Max Stackelberg Games with Coupled Constraints

In the previous chapter, we covered flow-based test synthesis in which flow networks that were defined on the transition system of the system under test. The temporal logic objectives were also limited to a simple sequence of waypoints defined on the states of the system.

This new flow-based formulation has the following key advances: i) edge-cuts or restrictions are found on a product graph of the system transition system and a Büchi automaton representing the system and test objectives, ii) the new flow-based optimization no longer has exponential number of constraints, and is tractable to encode, and iii) the synthesized test is *reactive* to system behavior and no longer limited to static obstacles. Furthermore, this new flow-based reformulation represents edge-cuts as continuous variables, but the complexity still remains since the formulation becomes a min-max Stackelberg game with coupled constraints. In this part of the chapter, we will introduce the product graph and setup the flow-based reactive synthesis formulation. This work serves as a prelude to the next section in which we propose an MILP approach to solving the problem that comes with guarantees of synthesizing a test that is feasible and not overly-restrictive, and also lends to a more tractable implementation.

Stackelberg Game Formulation

Leveraging automata theory to represent product graphs leads to the intermediate node becoming analogous to the waypoints. Instead of a sequence of waypoints on $T_{\text{sys}}.S$, the nodes $\mathtt{I}$ become the waypoint that the system must be routed through. Instead of defining flows for every pair of propositions, we will define three flows: from source to intermediate ($\mathbf{f}_{\mathtt{S}\to\mathtt{I}}$), from intermediate to sink ($\mathbf{f}_{\mathtt{I}\to\mathtt{T}}$), and a bypass flow ($F_{\mathtt{S}\to\mathtt{T}}$). The test strategy synthesis problem can be seen as placing restrictions such that flows $\mathbf{f}_{\mathtt{S}\to\mathtt{I}}$ and $\mathbf{f}_{\mathtt{I}\to\mathtt{T}}$ are preserved while the bypass flow is cut.

Definition 4.16 (Constrained Min-Max Optimization with Coupled Constraints [102]**).** A *constrained min-max optimization with dependent feasible sets*, also referred to as a min-max Stackelberg game, between the lead player X with strategy space $\mathcal{X}$ and a follower player Y with strategy space $\mathcal{Y}$ can be represented as the following optimization:

$$\begin{aligned} \min_{x\in\mathcal{X}} \max_{y\in\mathcal{Y}} \quad & f(x,y) \\ \text{s.t.} \quad & g(x,y) \geq 0, \end{aligned} \tag{4.9}$$

where $f(x,y) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ is the objective function and $g(x,y) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^k$ represents the constraints.

In this Stackelberg formulation, the outer (min) player is the test environment controls the flow variables $\mathbf{f}_{\mathtt{S}\to\mathtt{I}}$ and $\mathbf{f}_{\mathtt{I}\to\mathtt{T}}$, edge cuts $\mathbf{d}$, and the auxiliary variable t. Let $0 < t \leq 1$ be an auxiliary variable defined as: $t = \frac{1}{F}$, given that $F > 0$. Hereafter, all flow variables are normalized with respect to the total flow by multiplying with t. The objective function is such that the tester maximizes the total flow $F = \min\{F_{\mathtt{S}\to\mathtt{I}}, F_{\mathtt{I}\to\mathtt{T}}\}$, 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 briefly described. A lot of these constraints are formally explained in the following section with MILPs, and are intuitively described here for brevity. The objective, which is given as, $t + \gamma F_{\mathtt{S}\to\mathtt{T}}$, where $\gamma > 0$ is a regularization parameter that penalizes the test environment for any non-zero bypass flow through the network. The auxiliary variable t is useful because the outer (min) player can minimize this term, thus maximizing the total flow value F. This objective also works for the inner (max) player, which in the worst-case, seeks to take a bypass path.

Table 4.2: List of outer player constraints used in Optimization 4.21 with normalized flows.

Outer Player Constraints	**Equation** $k \in \{\mathtt{S} \to \mathtt{I},\ \mathtt{I} \to \mathtt{T}\}$	
Capacity (exact)	$\forall e \in E, \quad d^e \in \{0, t\}, \quad 0 \leq f_k^e \leq t.$	(oc10)
Capacity (approx.)	$\forall e \in E, \quad 0 \leq d^e \leq t, \quad 0 \leq f_k^e \leq t.$	(oc11)
Conservation	$\forall v \in S \setminus \{\mathtt{S}, \mathtt{T}\}, \sum_{\substack{u:(u,v) \\ \in E}} f_k^{(u,v)} = \sum_{\substack{u:(v,u) \\ \in E}} f_k^{(v,u)}.$	(oc12)
Cut	$\forall e \in E, \quad d^e + f_k^e \leq t.$	(oc13)
Minimum Total Flow	$1 \leq F_{\mathtt{S}\to\mathtt{I}}$ and $1 \leq F_{\mathtt{I}\to\mathtt{T}}$	(oc14)
Feasibility	$F_{\mathcal{G}_{\text{sys}}}(q) \geq 1 \forall q \in \mathcal{B}_\pi.Q$	(oc15)
Static Obstacles	$d^{(i,j)} = d^{(k,l)}$ if $i.s = k.s$ and $j.s = l.s$	(oc16)

The three flows satisfy standard network flow constraints concerning capacity and conservation. The only difference: i) all standard flow constraints are normalized by multiplying through t, and ii) the cut variable $\mathbf{d}$ is relaxed as opposed to being restricted to vectors with binary elements. Furthermore, the cut variable restricts flows as follows, for all $k \in \{\mathtt{S} \to \mathtt{I}, \mathtt{I} \to \mathtt{T}, \mathtt{S} \to \mathtt{T}\}$, the cut constraints are:

$$\forall e \in \mathcal{G}.E, \quad d^e + f_k^e \leq t.$$

Table 4.3: List of inner player constraints used in Optimization 4.21 with normalized flows.

Inner Player Constraints	**Equation**	
Cut	$\forall e \in E, \quad d^e + f^e_{\mathtt{S}\to\mathtt{T}} \le t.$	(ic17)
Capacity (approx.)	$\forall e \in E, \quad 0 \le f^e_{\mathtt{S}\to\mathtt{T}} \le t.$	(ic18)
Conservation	$\forall v \in S \setminus \{\mathtt{S},\mathtt{T}\}, \sum_{u\in V} f^{(u,v)}_{\mathtt{S}\to\mathtt{T}} = \sum_{u:(v,u)\in E} f^{(v,u)}_{\mathtt{S}\to\mathtt{T}}.$	(ic19)
No $\mathtt{I}$ Flow	$f^{(u,v)}_{\mathtt{S}\to\mathtt{T}} = 0$ if $u \in \mathtt{I}$ or $v \in \mathtt{I}$.	(ic20)

That is, despite having multiple flows, they do not compete for capacity; a cut d^e will equally restrict all flows. Note that the above cut constraint for the case of bypass flow $k = \mathtt{S} \to \mathtt{T}$ is the coupling constraint between inner and outer players; the test environment controls the cut and the inner player controls bypass flow, but they must together respect this cut constraint. The constraints on the outer and inner players for the game-based network flow optimization are given in Tables 4.2 and 4.3. The constraint on minimum total flows in equation (ic(4.14)) is applied to normalized total flow values $F_{\mathtt{S}\to\mathtt{I}}$ and $F_{\mathtt{I}\to\mathtt{T}}$, and implies that the total flow $F > 0$. If this constraint is violated, the constraints are infeasible and no solution is returned. A detailed approach on these network flow constraints (e.g., feasibility constraints in Eq. (ic(4.15))) is given in the next section when we re-use these constraints to setup the final MILP formulation to solve the routing problem. The game based network flow optimization formulation is given below.

MCF-OPT(γ):

$$\min_{\substack{\mathbf{f}_{\mathtt{S}\to\mathtt{I}}\mathbf{f}_{\mathtt{I}\to\mathtt{T}},\mathbf{d},t,\\ \mathbf{f}_{\mathtt{S}_{\mathsf{sys}}\to\mathtt{T}_{\mathsf{sys}}}(q),\forall q\in\mathcal{B}_\pi.Q}} \; \max_{\mathbf{f}_{\mathtt{S}\to\mathtt{T}}} \quad t + \gamma \sum_{v:e=(\mathtt{S},v)\in\mathcal{G}.E} f^e_{\mathtt{S}\to\mathtt{T}} \tag{4.21}$$
$$\text{s.t.} \quad (\text{oc}(4.11))\text{-}(\text{oc}(4.15)),\ (\text{ic}(4.17))\text{-}(\text{ic}(4.20)).$$

This optimization is in the form of a min-max Stackelberg game with dependent constraint sets studied in [103]. However, there are no known polynomial-time solutions to solve this optimization since its value function is not convex. Given below, the value function outputs the optimal value of the inner optimization problem in Optimization (4.21) for any choice of $(\mathbf{f}, \mathbf{d}, t)$. The flow values F and $F_{\mathtt{S}\to\mathtt{T}}$

are functions of the edge cuts $\mathbf{d}$ — they represent max-flow on their corresponding flow networks with the capacities reduced by $\mathbf{d}$.

$$V(\mathbf{f},\mathbf{d},t) = \max_{\mathbf{f}_{\mathtt{S}\to\mathtt{T}}} \quad t + \gamma F_{\mathtt{S}\to\mathtt{T}}$$
$$\text{s.t.} \quad (\text{ic}(4.17)) - (\text{ic}(4.20)).$$

The *value function* is defined over the space of outer player variables that satisfy constraints (oc(4.17)), (oc(4.18)), (oc(4.19)), (oc(4.20)). If the value function is convex in the outer player variables, then there exist efficient algorithms to converge to the Stackelberg equilibrium [102]. However, our value function is not convex because the inner problem corresponds to solving a max flow problem parameterized by cuts $\mathbf{d}$, which is not convex. Note that this problem complexity is despite the fact that the objective and constraints are all affine and defined over continuous valued domains. The beaver rescue and motion primitive hardware experiments are derived from solutions to **MCF-OPT**(γ) for $\gamma = 1000$, which is solved using Pyomo [104]. Also note that all Stackelberg equilibria for **MCF-OPT**(γ) need not correspond to bypass flow value $F_{\mathtt{S}\to\mathtt{T}}$ being zero. This shortcoming is handled in the MILP formulation presented in the next section, and is largely driven by insights from taking the dual of the inner maximization, as we shall see below.

However, this approach did not scale to solving medium-sized examples that we will see later in this chapter. In an effort to address this, the first attempt was to take the Lagrange dual of the inner maximization, and solve a minimization instead of a min-max game. In traditional max-flow problems, the Lagrange dual of the max flow linear program is the minimum cut linear program with Lagrange multipliers corresponding to edge cuts and partitions [105]. However, in our case, since the outer player modifies edge cuts $\mathbf{d}$, the Lagrange multipliers correspond to paritions on the modified graph but do not inform the actual partition that we seek. The Lagrangian $\mathcal{L}$ associated with the inner player is:

$$\mathcal{L}\left(\mathbf{f},\mathbf{d},t,\mathbf{f}_{\mathtt{S}\to\mathtt{T}},\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\nu}\right) = t+\gamma F_{\mathtt{S}\to\mathtt{T}}+\sum_{v\in V\setminus\{\mathtt{S},\mathtt{I},\mathtt{T}\}}\mu^i\left(\sum_{u\in V\setminus\mathtt{I}}\mathbf{f}^{(u,v)}_{\mathtt{S}\to\mathtt{T}}-\sum_{u\in V\setminus\mathtt{I}}\mathbf{f}^{(v,u)}_{\mathtt{S}\to\mathtt{T}}\right)$$
$$+\sum_{\substack{(u,v)\in E\setminus E(\mathtt{I})\\ u\in\mathtt{T},v\in\mathtt{S}}}\nu^{(u,v)}\mathbf{f}^{(u,v)}_{\mathtt{S}\to\mathtt{T}}+\sum_{e\in E\setminus E(\mathtt{I})}\lambda^e\left(t-d^e-\mathbf{f}^e_{\mathtt{S}\to\mathtt{T}}\right), \quad (4.22)$$

where $\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\nu}$ are the Lagrange variables; $\boldsymbol{\lambda}$ is associated with edge-cuts and $\boldsymbol{\mu}$ represents the partition of nodes. Finding the optimal Lagrange multipliers results in following dual problem:

$$
\begin{aligned}
\min_{\boldsymbol{\lambda}\in\mathbb{R}_+^{|E|},\boldsymbol{\mu}\in\mathbb{R}_+^{|V|}} \quad & t+\gamma\sum_{e\in E}\lambda^e(t-d^e) \\
\text{s.t.} \quad & \mu^{\mathtt{S}}-\mu^{\mathtt{T}}\geq 1, \\
& \lambda^{(u,v)}-\mu^u+\mu^v\geq 0,\ \forall(u,v)\in E\setminus E(\mathtt{I}).
\end{aligned}
\tag{4.23}
$$

This dual corresponds to the dual of the max-flow problem on the graph (without nodes $\mathtt{I}$) with edge capacities $t-\mathbf{d}$. In the canonical max-flow problem, $\boldsymbol{\mu}$ represents node partitioning corresponding to minimum cut. In our case, the solution to this dual problem returns optimal Lagrange multipliers $\boldsymbol{\mu}$ as a function of the outer player variable $\mathbf{d}$. Therefore, the choice of $\mathbf{d}$ by the outer player affects $\boldsymbol{\mu}$. The partition of the graph $G\setminus\mathtt{I}$, characterized by $\boldsymbol{\mu}^*(\mathbf{d}^*)$, will be the optimal lagrange multiplier at the equilibrium $\mathbf{d}^*$. Due to strong duality, we can rewrite the **MCF-OPT**(γ) equivalently as follows:

OPT-MIN(γ)

$$
\begin{aligned}
\min_{\mathbf{f},\mathbf{d},t,\boldsymbol{\lambda}\in\mathbb{R}_+^{|E|},\boldsymbol{\mu}\in\mathbb{R}_+^{|V|}} \quad & t+\gamma\sum_{e\in E}\lambda^e(t-d^e) \\
\text{s.t.} \quad & \mu^{\mathtt{S}}-\mu^{\mathtt{T}}\geq 1, \\
& \lambda^{(u,v)}-\mu^u+\mu^v\geq 0,\ \forall(u,v)\in E\setminus E(I), \\
& (\text{oc}(4.11))\text{-}(\text{oc}(4.15)).
\end{aligned}
\tag{4.24}
$$

Despite being a single minimization, **OPT-MIN**(γ) has the structure of a *bilinear* program, which is also a consequence of our min-max Stackelberg game not being convex-concave. In general, solving a bilinear program is NP-hard in the problem data [106]. For zero bypass flow, the second bilinear term must be zero. This was also empirically observed when solving **OPT-MIN**(γ). At zero bypass flow, the term λ becomes equal to the unnormalized cut value d; if $d^{(u,v)}=t$, then $\lambda^{(u,v)}$ must equal 1 to partition nodes u and v into separate groups, and if $d^{(u,v)}=0$, then $\lambda^{(u,v)}=0$ to ensure that the nodes remain in the same group. First, we can recognize that the dual variables $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ can take integer values [105]. This allows us to formulate the problem as a mixed-integer linear program, where we can encode zero bypass flow as a constraint by setting $\lambda^e=d^e$ for edges $e\in E\setminus E(\mathtt{I})$. Furthermore, this formulation also allows the use of the unnormalized flow and capacity values, removing the need for the auxiliary parameter t.

4.7 Part II: Flow-based Optimization via Mixed-Integer Linear Programming

Once again, we revisit a network flow formulation to solve the routing optimization for the expanded class of specifications. Recall that $Paths(\mathtt{S}, \mathtt{T})$ on the graph G are the set of possible test executions. Therefore, the set of edge cuts on G must be such that all $Paths(\mathtt{S}, \mathtt{T})$ are routed to visit at least one node in the intermediate set $\mathtt{I}$. The set of edge cuts to achieve this need not be unique, and therefore, we also require that resulting test strategy to not be overly-restrictive. Additionally, we also minimize the cardinality of the set of edge cuts to remove unnecessary restrictions. In comparison to the previous chapter, the formulation presented defines just a single flow network. Furthermore, the routing problem is solved as an MILP as opposed to a min-max game with coupled constraints. This new formulation allows us to derive guarantees that the optimal solution will provide a test strategy that solves Problem 4.1. Furthermore, this formulation easily lends itself to extensions (e.g., adding auxiliary constraints, excluding certain solutions, accommodating various types of test environments).

Consider the flow network $\mathcal{G} = (V, E, (\mathtt{S}, \mathtt{T}))$ defined based on the graph G: nodes are defined exactly as $V := G.V$, and edges $E := G.E \backslash \{(u, u) \in G.E \,|\, u \in V\}$ are the same as G with the exception of self-loops, and the source and target correspond to $\mathtt{S}$ and $\mathtt{T}$, respectively. The reason for introducing flow networks separately is to maintain a representation without self-loop transitions which are not relevant when computing the flow on a graph. Maintaining self-loops on G, however, is important since it is the product between a transition system and a Büchi automaton. For simplicity, notation for nodes (V) and edges (E) is shared between the graph G and its network $\mathcal{G}$ since self-loop transitions are also not a part of the test harness. If self-loops become important, the notation $G.E$ will be explicitly used. On $\mathcal{G}$, we introduce the flow vector $\mathbf{f} \in \mathbb{R}^{|E|}$ and a Boolean vector $\mathbf{d} \in \mathbb{B}^{E \backslash E(\mathtt{I})}$ carries the edge-cut value for each edge. For some $e \in E$, $d^e = 1$ denotes that edge e is cut and the corresponding system action is restricted, and $d^e = 0$ denotes that the edge remains. Immediately, it can be specified that all edges outside the test harness cannot be restricted:

$$\begin{aligned} &d^e \in \{0, 1\}, \quad \forall e \in E, \text{ and} \\ &d^e = 0, \quad \forall e \notin E_H. \end{aligned} \tag{c1}$$

The set $E(\mathtt{I}) = \{(u, v) \in E \,|\, u \in \mathtt{I} \text{ or } v \in \mathtt{I}\}$ is the set of edges that enter or exit from a node in set $\mathtt{I}$.

Objective. Among all possible sets of edge cuts that route test executions through $\mathtt{I}$ (corresponding to satisfying the test objective), we seek a test strategy that is not overly-restrictive. Thus, we optimize for a set of edge cuts that maximizes the total flow from $\mathtt{S}$ to $\mathtt{T}$ on $\mathcal{G}$. Since the edges have unit capacity, the set of edge cuts that maximize the total flow will also result in the largest set Θ_u (see Lemma 4.3). Thus, maximizing the flow alone is sufficient to get a test that is not overly-restrictive (see Remark 4.7). In addition, unnecessary edge-cuts can be reduced by introducing a second term of subtracting the fraction of edges that are cut from the flow value:

$$\sum_{\substack{(u,v)\in E,\\ u\in \mathtt{S}}} f^{(u,v)} - \frac{1}{|E|}\sum_{e\in E} d^e. \tag{4.25}$$

The regularize $\frac{1}{|E|}$ is chosen to avoid trade-off between the terms. Due to binary edge cuts and integer edge capacity, the maximum flow value on the graph will always be an integer. The second term, however, will always take a fractional value between 0 and 1 corresponding to none of the edges being cut to all of the edges being cut. Thus, maximizing the objective will always favor increasing the first term as much as possible, and then minimizing the number of edges that are cut.

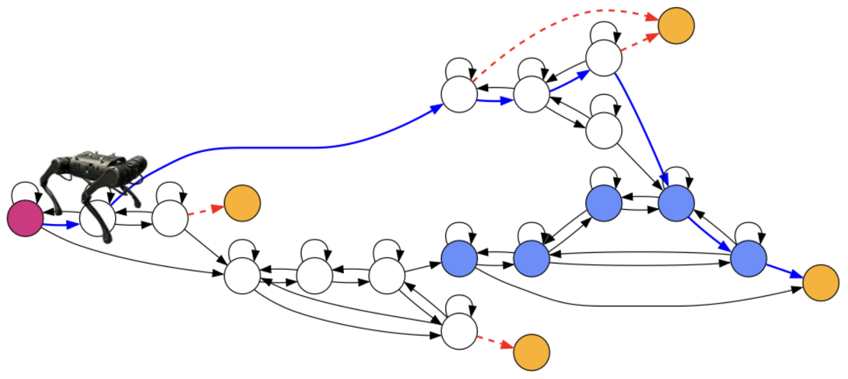
Network Flow Constraints. The flow vector $\mathbf{f}$ is subject to the standard network flow constraints:

$$\text{Flow constraints (3.1), (3.2), and (3.3) on flow network } \mathcal{G}. \tag{c2}$$

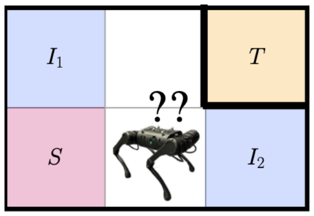
Next, an edge that is cut restricts the flow on that edge completely. However, an edge that is not cut may or may not have flow pushed through it:

$$\forall e \in E, \quad d^e + f^e \leq 1. \tag{c3}$$

Partition Constraints. In standard max-flow problems, the dual min-cut formulation has partition constraints that group nodes across a cut into two groups, one of which contains the source and the other that contains the sink [105]. The max-flow problem (and equivalently, the min-cut problem) is totally unimodular, implying that there exists an optimal integer solution. Our problem differs from the standard max-flow/min-cut problem in that we seek all $Paths(\mathtt{S},\mathtt{T})$ to be routed through $\mathtt{I}$. Alternatively, the set of edge cuts should be such that there exists a positive total flow on $\mathcal{G}$, but the network $(V\setminus \mathtt{I}, E\setminus E(\mathtt{I}), (\mathtt{S},\mathtt{T}))$ is fully partitioned. To capture this partitioning requirement, the partition conditions from standard settings is



(a) A path on G exists despite the cuts not being feasible for the system.



(b) Restrictions from the system perspective before it visits I_1 or I_2.

Figure 4.6: Illustration of why feasibility constraints are important for identifying a reactive test strategy that respects the system's assumptions. Since the system is not aware of the test objective, such a placement of constraints would lead to all paths being blocked from the system perspective.

adapted as follows. All nodes except those in $\mathtt{I}$ must be partitioned into the source $\mathtt{S}$ group or the sink $\mathtt{T}$ group. For this, introduce the variable $\boldsymbol{\mu} \in \mathbb{R}^{|V \setminus \mathtt{I}|}$ such that:

$$\begin{aligned} \mu^{\mathtt{S}} - \mu^{\mathtt{T}} &\geq 1, \\ 0 \leq \mu^{v} &\leq 1, \forall v \in V \setminus \mathtt{I}. \end{aligned} \tag{c4}$$

The partition condition is applied to edges that are not incoming or outgoing from $\mathtt{I}$:

$$d^{(u,v)} - \mu^{u} + \mu^{v} \geq 0, \; \forall (u,v) \in E \setminus E(I). \tag{c5}$$

Despite the vector $\boldsymbol{\mu}$ being real-valued, it only appears in these constraints. Therefore, the partition conditions form a block diagonal in the constraint matrix, and this block diagonal sub-matrix is totally unimodular. Therefore, we preserve the partitioning properties from standard min-cut despite adapting the constraint to our problem.

Feasibility Constraints. These constraints ensure that the synthesized test is feasible from the system's perspective (see Figure 4.6), that is, restrictions placed by the test strategy should be such that the system still has a chance of successfully navigating to the goal if it has not committed to an incorrect action (either unsafe or one that inevitably leads to livelock) up to that point. For this, the reactive edge cuts must respect system assumptions **A2** – **A4**. For every history variable $q \in \mathcal{B}_\pi.Q$, the function $\mathtt{S}_G : \mathcal{B}_\pi.Q \to G.S$ is the set of states on G when the the test execution can enter the history variable q, and is defined as follows,

$$\mathtt{S}_G(q) := \{(s, q) \in G.S \mid \forall((\bar{s}, \bar{q}), (s, q)) \in G.E,\ \bar{q} \neq q\}. \tag{4.26}$$

On the system product graph, these states map to the set:

$$\mathtt{S}_{G_\text{sys}}(q) := \{u \in G_\text{sys}.S \mid u = \mathcal{P}_{G \to G_\text{sys}}(v),\ v \in \mathtt{S}_G(q),\ \text{and}\ \exists\, Path(u, \mathtt{T}_\text{sys})\}, \tag{4.27}$$

where this set is empty if no path from the node u to $\mathtt{T}_\text{sys}$ exists on G_sys. For each $q \in \mathcal{B}_\pi.Q$, for each source in $\mathtt{s} \in \mathtt{S}_{G_\text{sys}}(q)$, define a flow network $\mathcal{G}_\text{sys}^{(q,\mathtt{s})} := (V_\text{sys}, E_\text{sys}, (\mathtt{s}, \mathtt{T}_\text{sys}))$, where nodes are $V_\text{sys} := G_\text{sys}.S$, and edges are $E_\text{sys} := G_\text{sys}.E \setminus \{(u, u) \in G_\text{sys}.E | u \in G_\text{sys}.V\}$. On graph $\mathcal{G}_\text{sys}^{(q,\mathtt{s})}$, flow vector is denoted as $\mathbf{f}_\text{sys}^{(q,\mathtt{s})}$. All such flow vectors are subject to the standard network flow constraints:

$$\begin{aligned}&\forall q \in \mathcal{B}_\pi.Q, \forall \mathtt{s} \in \mathtt{S}_{G_\text{sys}}(q),\\ &\text{Flow constraints (3.1), (3.2), and (3.3) on network } \mathcal{G}_\text{sys}^{(q,\mathtt{s})}.\end{aligned} \tag{c6}$$

The edge cut vector $\mathbf{d}$ from $\mathcal{G}$ is directly related to mapped to edges on each graph $\mathcal{G}_\text{sys}^{(q,\mathtt{s})}$. Then, it is checked whether there exists a $Path(\mathtt{s}, \mathtt{T}_\text{sys})$ on the system product graph copy $\mathcal{G}_\text{sys}^{(q,\mathtt{s})}$. Since edge-cuts/restrictions are placed reactively, only edge cuts starting from a state-history pair with history variable q applies to the graph $\mathcal{G}_\text{sys}^{(q,\mathtt{s})}$. Edges are grouped by the history variable using the mapping $\mathtt{Gr} : \mathcal{B}_\pi.Q \to 2^{G.E}$:

$$\mathtt{Gr}(q) := \{((s, q), (s', q')) \in G.E\}. \tag{4.28}$$

Then, the edge-cut values $\mathbf{d}$ are mapped onto system product graph copies, impacting the flow $f_\text{sys}^{(q,\mathtt{s}}$ by the cut constraint:

$$\begin{aligned}&\forall q \in \mathcal{B}_\pi.Q, \forall \mathtt{s} \in \mathtt{S}_{G_\text{sys}}(q), \forall (u, v) \in \mathtt{Gr}(q), \forall (u', v') \in E_\text{sys},\\ &\quad d^{(u,v)} + f_\text{sys}^{(q,\mathtt{s})^{(u',v')}} \leq 1,\ \text{ if } u'.s = u.s \text{ and } v'.s = v.s.\end{aligned} \tag{c7}$$

Once the cuts have been mapped, the check for a $Path(\mathtt{s}, \mathtt{T}_\text{sys})$ is ensured by requiring the flow value on each system product graph copy to be at least 1:

$$\sum_{(\mathtt{s},v) \in E_\text{sys}} f_\text{sys}^{(q,\mathtt{s})^{(\mathtt{s},v)}} \geq 1,\ \forall q \in \mathcal{B}_\pi.Q,\ \forall \mathtt{s} \in \mathtt{S}_{G_\text{sys}}(q). \tag{c8}$$

In the reactive obstacle setting, the feasibility constraints (c6)-(c8) group edge cuts for each history variable q, and check if there is a feasible path for the system. This feasibility check is carried out for every history variable q and every possible node $\mathtt{s} \in \mathtt{S}_G(q)$ at which the test execution enters history variable q. Since the system controller is unknown, it becomes imperative to check feasibility on every copy $\mathcal{G}_{\text{sys}}^{(q,\mathtt{s})}$.

Finally, the routing optimization is characterized as the following mixed-integer linear program (MILP) with the edge-cut vector $\mathbf{d}$ being the integer variable, and the flow $\mathbf{f}$ and partition $\boldsymbol{\mu}$ variables being continuous:

MILP-REACTIVE:

$$\begin{aligned} \max_{\substack{\mathbf{f},\mathbf{d},\boldsymbol{\mu}, \\ \mathbf{f}_{\text{sys}}^{(q,\mathtt{s})}\ \forall q \in \mathcal{B}_\pi.Q\ \forall \mathtt{s} \in \mathtt{S}_{\mathcal{G}_{\text{sys}}}(q)}} \quad & F - \frac{1}{|E|}\sum_{e \in E} d^e \\ \text{s.t.} \quad & (\text{c1})\text{-}(\text{c3}), (\text{c4})\text{-}(\text{c5}), (\text{c6})\text{-}(\text{c8}). \end{aligned} \tag{4.29}$$

Static Constraints. The feasibility constraints are simplified in test environments comprising only of static obstacles. A static obstacle is one that remains for the entire duration of the test. That is, an restriction on a system action at a particular state should always be in place regardless of the current state-history pair of the test execution. To specify this, the edges in G that correspond to the same transition of the system in $T_{\text{sys}}.E$ will have the same edge cut value:

$$d^{(u,v)} = d^{(u',v')}, \ \forall (u,v), (u',v') \in E, \ \text{if } u.s = u'.s \text{ and } v.s = v'.s. \tag{c9}$$

With Eq. (c9), a separate check for feasible paths on copies of the system product graph is not needed. By the projection map $\mathcal{P}_{G \to G_{\text{sys}}}$, a path on G implies a path on G_{sys}, and since restrictions do not change with q, the path on G_{sys} always remains. Therefore, the MILP formulation for static obstacles can be simplified to be:

MILP-STATIC:

$$\begin{aligned} \max_{\mathbf{f},\mathbf{d},\boldsymbol{\mu}} \ & F - \frac{1}{|E|}\sum_{e \in E} d^e \\ \text{s.t.} \quad & (\text{c1})\text{-}(\text{c3}), (\text{c4})\text{-}(\text{c5}), (\text{c9}). \end{aligned} \tag{4.30}$$

The following lemma and proof was taken from [107].

Lemma 4.2. For the case of static constraints, due to (c9), ensuring feasibility from the system's perspective is guaranteed by checking $F > 0$ on G. That is, $F > 0$ on G is equivalent to checking (c6)-(c8).

Proof. Under (c9), the edge groupings $\texttt{Gr}(q)$ become the same for all $q \in \mathcal{B}_\pi.Q$. Thus, the constraints (c6)-(c8) can be reduced onto a single flow network $\mathcal{G}_{\text{sys}} = (V_{\text{sys}}, E_{\text{sys}}, (\texttt{S}_{\text{sys}}, \texttt{T}_{\text{sys}}))$, where $\texttt{S}_{\text{sys}} := G_{\text{sys}}.I$. Equation (c8) being satisfied on $\mathcal{G}_{\text{sys}}$ implies that there is a path on G from $\texttt{S}$ to $\texttt{T}$ via Lemma 4.1. Additionally, if there is a path on G from $\texttt{S}$ to $\texttt{T}$ with the static constraints (c9), then it must be that there exists a path from $\texttt{S}_{\text{sys}}$ to $\texttt{T}_{\text{sys}}$ on G_{sys}. □

Remark 4.5. The reactive feasibility check involved checking for feasible paths on copies of system product graphs. Alternatively, this check can also be carried out using copies of the network $\mathcal{G}$, as we will see later in the section on computational complexity (Section 3.8). For implementation purposes, we choose the feasibility formulation presented in this section since it results in fewer variables and constraints in the optimization.

Mixed Constraints. In test environments with a mix of static obstacles and reactive obstacles and/or dynamic test agents, we require the static area $T_{\text{sys}}.E_{\text{static}} \subseteq T_{\text{sys}}.E$ to be given. Transitions in $T_{\text{sys}}.E_{\text{static}}$ can be restricted using static obstacles. In such mixed settings, the feasibility constraints (c6)–(c8) can be applied as normal, and the static constraints given in (c9) can be applied on edges $(u, v) \in E$ whose mapping onto the system transitions is in the static area, i.e., $(u.s, v.s) \in T_{\text{sys}}.E_{\text{static}}$.

MILP-MIXED:

$$\max_{\substack{\mathbf{f},\mathbf{d},\boldsymbol{\mu},\\ \mathbf{f}_{\text{sys}}^{(q,\mathtt{s})}\ \forall q\in\mathcal{B}_\pi.Q\ \forall \mathtt{s}\in\mathtt{S}_{\mathcal{G}_{\text{sys}}}(q)}} \quad F - \frac{1}{|E|}\sum_{e\in E} d^e \tag{4.31}$$

$$\text{s.t.} \quad \text{(c1)-(c3), (c4)-(c5), (c6)-(c8), (c9).}$$

Auxiliary Constraints. Auxiliary constraints are any additional affine constraints that are not required but can be added to the optimization to accommodate the test environment. For example, in some instances such as placing static obstacles like doors or fences, restricting a directed edge would also require the transition in the reverse direction to be blocked. This specific affine constraint can be written as

$$d^{(u,v)} = d^{(u',v')},\ \forall (u,v),\ (u',v') \in E, \text{if } u.s = v'.s \text{ and } v.s = u'.s. \qquad \text{(c10)}$$

Algorithm 5: Finding the test strategy π_{test}

```
1:  procedure FindTestStrategy($T_{\text{sys}}, H, \varphi_{\text{sys}}, \varphi_{\text{test}}$)
      Input: transition system $T_{\text{sys}}$, test harness $H$, system objective $\varphi_{\text{sys}}$, test objective $\varphi_{\text{test}}$
      Output: test strategy $\pi_{\text{test}}$
2:    $\mathcal{B}_{\text{sys}} \leftarrow \text{BA}(\varphi_{\text{sys}})$                          ▷ System Büchi automaton
3:    $\mathcal{B}_{\text{test}} \leftarrow \text{BA}(\varphi_{\text{test}})$                        ▷ Tester Büchi automaton
4:    $\mathcal{B}_{\pi} \leftarrow \mathcal{B}_{\text{sys}} \otimes \mathcal{B}_{\text{test}}$                        ▷ Specification product
5:    $G_{\text{sys}} \leftarrow T_{\text{sys}} \otimes \mathcal{B}_{\text{sys}}$                          ▷ System product
6:    $G \leftarrow T_{\text{sys}} \otimes \mathcal{B}_{\pi}$                              ▷ Virtual Product Graph
7:    $\mathtt{S}, \mathtt{I}, \mathtt{T} \leftarrow \text{IdentifyNodes}(G, \mathcal{B}_{\text{sys}}, \mathcal{B}_{\text{test}})$
8:    $\mathcal{G} \leftarrow \text{DefineNetwork}\ (G, \mathtt{S}, \mathtt{T})$
9:    $\mathfrak{G}_{\text{sys}} \leftarrow \mathtt{set()}$                               ▷ System Perspective Graphs
10:   for $q \in \mathcal{B}_{\pi}.Q$ do
11:       for $\mathtt{s} \in \mathtt{S}_{G_{\text{sys}}}(q)$ do
12:           $\mathcal{G}_{\text{sys}}^{(\mathtt{s},q)} \leftarrow \text{DefineNetwork}(G_{\text{sys}}, \mathtt{s}, \mathtt{T}_{\text{sys}})$
13:           $\mathfrak{G}_{\text{sys}} \leftarrow \mathfrak{G}_{\text{sys}} \cup \mathcal{G}_{\text{sys}}^{(\mathtt{s},q)}$
14:   $\mathbf{d}^* \leftarrow \text{MILP}(\mathcal{G}, T, \mathfrak{G}_{\text{sys}}, \mathtt{I}, H)$              ▷ Reactive, static, or mixed.
15:   $C \leftarrow \{(u,v) \in G.E \,|\, \mathbf{d}^{*(u,v)} = 1\}$                  ▷ Cuts on $G$
16:   $\pi_{\text{test}} \leftarrow$ Define test strategy according to equation (4.33)
17:   return $\pi_{\text{test}}$
```

Characterizing Optimization Results

The flow value (3.4) of the network is always integer-valued since the edge cuts are binary, and therefore, any strictly positive flow value corresponds to at least one valid test execution. In the following cases, the problem data are *inconsistent* and a flow value ≥ 1 cannot be found.

Case 1: There is no path from $\mathtt{S}$ to $\mathtt{T}$ on G (and equivalently, no path from $\mathtt{S}_{\text{sys}}$ to $\mathtt{T}_{\text{sys}}$ on G_{sys}). In this case, the optimization will not have to place any cuts because the only possible maximum flow value is 0.

Case 2: There is a path from $\mathtt{S}$ to $\mathtt{T}$ on G, but there is no path $\mathtt{S}$ to $\mathtt{T}$ in G visiting an intermediate node in $\mathtt{I}$. In this case, the partition constraints will cut all paths from $\mathtt{S}$ to $\mathtt{T}$, while by Lemma 4.1 the feasibility constraints require a path to exist from $\mathtt{S}$ to $\mathtt{T}$—a contradiction. The optimization is infeasible in this instance.

For each MILP, the set of edges that are cut are found from the optimal $\mathbf{d}^*$ as follows, $C := \{(u,v) \in E \setminus E(\mathtt{I}) \,|\, d^{*(u,v)} = 1\}$, resulting in the cut network $\mathcal{G}_{\text{cut}} = (V, E \setminus C, (\mathtt{S}, \mathtt{T}))$. The bypass flow value is computed on the network $\mathcal{G}_{\text{byp}} := (V_{\text{byp}}, E_{\text{byp}}, (\mathtt{S}, \mathtt{T}))$, where $V_{\text{byp}} := V \setminus \mathtt{I}$, and $E_{\text{byp}} := E \setminus (E(\mathtt{I}) \cup C)$. A strictly positive bypass flow value indicates the existence of a $Path(\mathtt{S}, \mathtt{T})$ on $\mathcal{G}_{\text{cut}}$ that does not visit an intermediate node in $\mathtt{I}$.

Theorem 4.1. The optimal or feasible cuts C returned by each MILP result in a bypass flow value of 0.

Proof. The partition constraints (c4) and (c5) partition the set of vertices $V \setminus \mathtt{I}$ into two groups: nodes with potential $\mu = 0$ (e.g., $\mathtt{T}$) and nodes with potential $\mu = 1$ (e.g., $\mathtt{S}$). On any path $v_0 \dots v_k$ on $\mathcal{G}_{\text{byp}}$, where $v_0 = \mathtt{S}$ and $v_k = \mathtt{T}$, the difference in potential values can be expressed as a telescoping sum: $\sum_{i=0}^{k-1} (\mu^i - \mu^{i+1}) = \mu^{\mathtt{S}} - \mu^{\mathtt{T}}$. Then, by partition constraints (c4) and (c5),

$$\sum_{i=0}^{k-1} d^{(v_i, v_{i+1})} \geq \sum_{i=0}^{k-1} (\mu^i - \mu^{i+1}) = \mu^{\mathtt{S}} - \mu^{\mathtt{T}} \geq 1.$$

Therefore, for at least one edge (v_i, v_{i+1}) on the path, where $0 \leq i \leq k-1$, the corresponding cut value is $d^{(v_i, v_{i+1})} = 1$. These edges belong to the set of cut edges C. Thus, the flow value on $\mathcal{G}_{\text{byp}}$ is zero. □

Theorem 4.2. For each MILP, the returned cuts C are such that there always exists a path to the goal from the system's perspective.

Proof. First, consider the MILP in the reactive setting. The optimal cuts C satisfy the feasibility constraints (c6), (c7), and (c8). These constraints ensure that for each history variable $q \in \mathcal{B}_\pi.Q$, there exists a path for the system from each state $\mathtt{s} \in \mathtt{S}_{G_{\text{sys}}}(q)$ to $\mathtt{T}_{\text{sys}}$ on G_{sys}. The edge cuts C are grouped by their history variable (see equation (4.28)) and mapped to the corresponding $\mathcal{G}_{\text{sys}}^{(q,\mathtt{s})}$ (see equation (c7)). Then, each copy $\mathcal{G}_{\text{sys}}^{(q,\mathtt{s})}$ represents all the cuts that can be simultaneously applied when the state of the test execution is at history variable q. Thus, all restrictions on system actions at history q are captured by the cuts on $\mathcal{G}_{\text{sys}}^{(q,\mathtt{s})}$. Since this is true for every q and every source state $\mathtt{s}$ at which the test execution enters into q, there always exists a path to the goal by equation (c8). The proof for the static and mixed settings follows similarly. □

Remark 4.6. Note that Theorems 4.1 and 4.2 are not limited to optimal solutions of the MILP, but apply to feasible solutions as well. That is, any time termination of the MILP provided that a feasible solution has been found is sufficient to find a test strategy with guarantees that the system assumptions are satisfied, and that there are no bypass paths. However, only an optimal solution can return a test strategy that is not overly-restrictive. However, the following lemma only applies to optimal solutions.

Lemma 4.3. For each MILP, the optimal cuts C correspond to maximizing the cardinality of Θ_u.

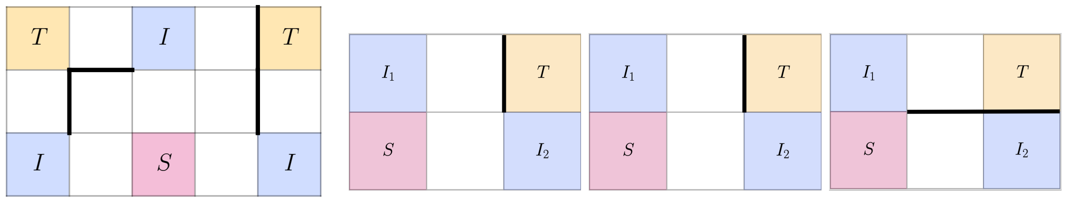
Proof. By construction, a realization of the flow $\mathbf{f}$ on $\mathcal{G}$ corresponds to a set of unique state-history traces Θ_u. The MILP objective maximizes the flow, and therefore the cardinality of Θ_u is maximized.

Additionally, the feasibility constraints do not induce any conservativeness in terms of finding a test strategy that is not overly-restrictive. Let $Path(\mathtt{S}, \mathtt{s})$ be a path from the source of the product graph G to node $\mathtt{s}$, where $\mathtt{s} \in \mathtt{S}_{G_{\text{sys}}}(q)$ for $q \in mathttB_{\pi}.Q$ is some source at which the execution updates to history variable q. Since the number of edge-cuts are minimized in the optimization objective, no $Path(\mathtt{S}, \mathtt{s})$ will be restricted unless if necessary to cut a bypass path. Even in this instance, checking that there exists a $Path(\mathtt{s}, \mathtt{T})$ in the feasibility constraints will not be an issue. If all $Path(\mathtt{s}, \mathtt{T})$ are bypass paths, then the optimization will choose to cut all $Path(\mathtt{S}, \mathtt{s})$. Thus, despite the feasibility constraints, the optimal solution of the MILP still corresponds to a not overly-restrictive strategy. □

Remark 4.7. The definition of a not overly-restrictive test strategy, both in this and the previous chapter, did not account for the number of restrictions placed. In this chapter, the routing optimization, in addition to providing optimal edge-cuts corresponding to not overly-restrictive test strategies, also returns the minimum the number of such restrictions required to realize the strategy. Overly restricting the system, especially when not necessary, could potentially increase testing effort.

4.8 Test Strategy Synthesis

This section outlines how edge-cuts found solving the optimizations can help construct a test strategy. This section is split into two parts: i) construction of a test strategy involving static and reactive obstacles matched to the optimization solution, and ii) synthesis of a test agent strategy for a given dynamic agent such that



(a) Static Obstacles in black. (b) q_0 (c) q_6 (d) q_7

Figure 4.7: Static and reactive obstacle placement for running examples. Figure 4.7a shows static obstacles synthesized for Example 4.1. Figures 4.7b, 4.9c, and 4.7d show a test environment implementation of a reactive test strategy for Example 4.2.

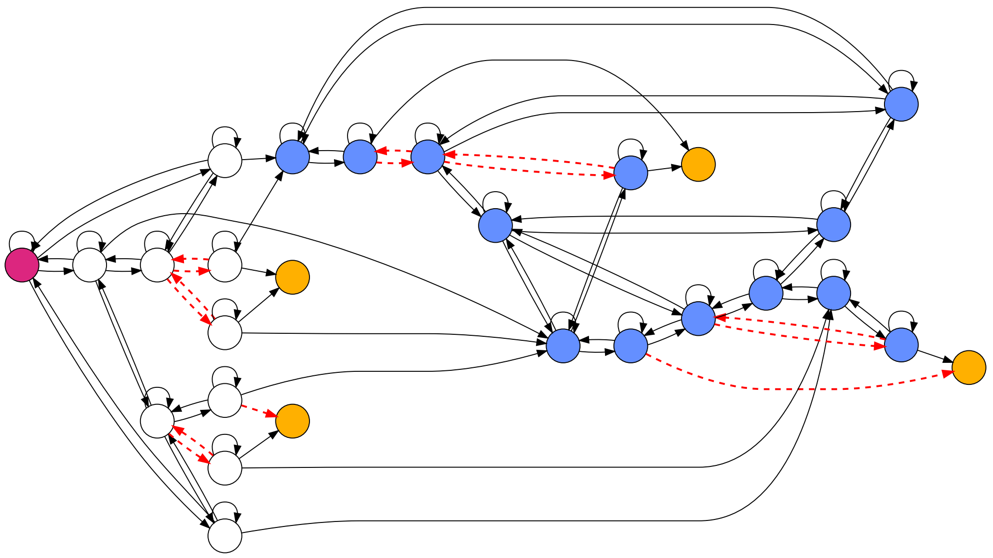
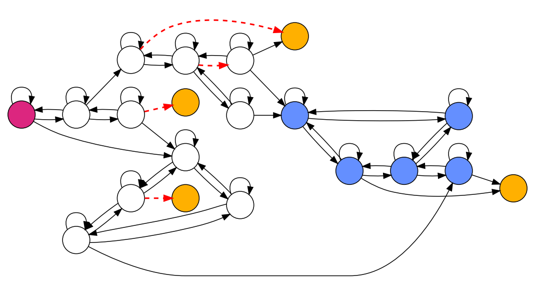


Figure 4.8: Virtual product graph with static cuts in dashed red for the medium example 4.1. Static obstacles in Fig. 4.7a corresponding to edge cuts found on this product graph for Example 4.1. States marked S, I, and T illustrated in Fig. 4.7a correspond to states `S` (magenta ●), `I` (blue ●), and `T` (yellow ●) on $\mathcal{G}$ as shown here. There are three edge-disjoint paths on this graph from the source to the target nodes.


(a) Virtual product graph G.

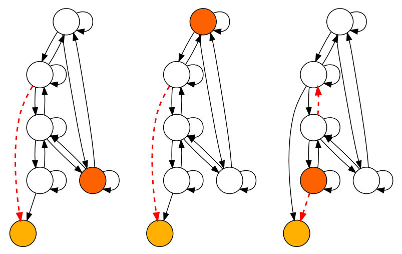

(b) $G_{\text{sys}}^{(q_0,s_3)}$ (c) $G_{\text{sys}}^{(q_6,s_1)}$ (d) $G_{\text{sys}}^{(q_7,s_{11})}$

Figure 4.9: Virtual product graph and system product graphs for Example 4.2. Fig. 4.9a shows the virtual product graph G, with the source S (magenta ●), the intermediate nodes I (blue ●), and the target nodes (yellow ●). Edge cut values for each edge in G are grouped by their history variable q and projected to the corresponding copy of G_{sys}. Figs. 4.9b—4.9d show the copies of G_{sys} with their source (orange ●) and target node (yellow ●). The graphs in Figs. 4.9b—4.9d correspond to the history variables q_0, q_6, and q_7 from $\mathcal{B}_\pi$ shown in Fig. 4.2c. The constraints (c6)—(c8) ensure that the edge cuts are such that a path from each source to the target node exists for each history variable q.

the synthesized strategy matches the restrictions on system actions from the optimization solution.

Test Environments with Static and/or Reactive Obstacles

In this section, we will detail the construction of a test strategy from a solution of the MILP solved in the static, reactive, or mixed settings. First, consider the more general reactive setting. A solution (not necessarily optimal) of the MILP in each setting returns a set of edge-cuts C that can be parsed into a reactive map $\mathcal{C} : \mathcal{B}_\pi.Q \rightarrow T_{\text{sys}}.E$ of system restrictions:

$$\mathcal{C}(q) := \{(s, s') \in T_{\text{sys}}.E \mid ((s, q), (s', q')) \in C\}. \tag{4.32}$$

The argument of the reactive map is the state history variable q, and intuitively, $\mathcal{C}(q)$ represents the set of all system restrictions that can be active when the test execution is at the state history variable q. Formally, when the test execution ϑ arrives at a state (s, q) at some time $k \geq 0$ (and correspondingly the system trace σ is at s at $k \geq 0$), and the system restriction $(s, s') \in \mathcal{C}(q)$, the test environment must restrict the system action corresponding to (s, s') at this event. Therefore, the test strategy is constructed as

$$\begin{aligned}\pi_{\text{test}}(\sigma_{0:k}) \coloneqq \{a \in T_{\text{sys}}.A \mid s' \in T_{\text{sys}}.\delta(s,a),\ q = \textsc{Hist}(\sigma_{0:k}),\\ (s,s') \in \mathcal{C}(q)\}.\end{aligned} \tag{4.33}$$

Practically, π_{test} can be realized by the test environment by placing obstacles during the test execution in reaction to system behavior (given by the trace σ). The *set of active obstacles* at time step k denoted by $\texttt{Obs}(\sigma_{0:k})$ is the set of state-action restrictions that are placed at time k. Note that the set of active obstacles can contain more restrictions than the test strategy. For example, an action corresponding to transition (s', s'') of the system can be restricted even though the system is not at s' at time k. Intuitively, this might correspond to a static obstacle that is far away from the system and not blocking it immediately. The set of active obstacles represent different implementations of the same reactive test strategy. A few implementations of the reactive test strategy as a set of active obstacles are:

1. **Exact Reactive Placement**: In this setting, the set of active obstacles correspond exactly the set of actions restricted by the test strategy: $\texttt{Obs}(\sigma_{0:k}) \coloneqq \{(s,a) | s = \sigma_k,\ a \in \pi_{\text{test}}(\sigma_{0:k})\}$. The obstacle is only active when the system is in a state from which an action is restricted.

2. **Instantaneous Placement**: In this setting, the test environment instantaneously places all obstacles or the restrictions in $\mathcal{C}(q)$ are realized "at once" when the test execution enters a state-history trace with history variable q. Concretely, let (s_k, q_k) be the state-history of the test execution at some time $k \geq 0$, then the set of active obstacles are

$$\texttt{Obs}(\sigma_{0:k}) \coloneqq \{(s,a) \mid (s,s') \in \mathcal{C}(q_k) \text{ and } s' \in T_{\text{sys}}.\delta(s,a)\}.$$

3. **Accumulative Placement**: In this setting, active obstacles are accumulated as the system trace evolves as long as the history variable does not change. For some $k > 0$, let (s_{k-1}, q_{k-1}) and (s_k, q_k) be the state-history pairs at time steps $k-1$ and k, respectively. If $q_{k-1} \neq q_k$, then the set of active obstacles becomes $\texttt{Obs}(\sigma_{0:k}) \coloneqq \{(s_k, a) \mid a \in \pi_{\text{test}}(\sigma_{0:k})\}$. As the system trace evolves to state-history pairs (s_l, q_l), where $l > k$ and $q_l = q_k$, the set of active obstacles are accumulated: $\texttt{Obs}(\sigma_{0:l}) = \bigcup_{j=k}^{l} \texttt{Obs}(\sigma_{0:j})$. When the history variable advances, i.e., $q_l \neq q_k$, then the set of active obstacles are *reset*: $\texttt{Obs}(\sigma_{0:l}) \coloneqq \{(s_l, a) \mid a \in \pi_{\text{test}}(\sigma_{0:l})\}$.

All three methods of determining the set of active obstacles will *simulate* the reactive test strategy; they are varied implementations of the test environment. The placement of obstacles need not coincide with when the system observes these obstacles, which depends on the system implementation. However, we assume that the system can observe all restrictions placed by the test environment on its current state before it commits to an action.

Remark 4.8 (On Relaxing the Assumption 4.1)**.** Roughly speaking, the feasibility constraints (c8) ensure that placing obstacles does not block the system from its goal. This condition is checked by ensuring that there exists feasible path for the system from every possible source $\mathtt{s} \in \mathtt{S}_{G_{\text{sys}}}(q)$ for every history variable q when all restrictions in $C(q)$ have been placed. With Assumption 4.1, the above feasibility constraint is a sufficient check since the system can backtrack to the source and find an alternative path if it encounters a restriction placed by the test environment. However, this assumption is not necessary and can be relaxed in one of two ways. First, if a restriction were to cause a livelock (i.e., system has no choice but to remain in the same state or be stuck in a cycle), then the restriction must be revealed to the system before the livelock becomes inevitable. Second, for every cut $((s,q),(s',q'))$ in the set of edge-cuts C, we can check that there exists a $Path((s,q),\mathtt{T})$ on $\mathcal{G}$ after edges C have been removed. If this is not the case, then the solution corresponding to C can be added as a counterexample constraint to the MILP, which will then be resolved. This process is repeated until the set of cuts C are accepted. Implementation of a counterexample constraint is detailed in section 4.8.

Proposition 4.2. In both the instantaneous and accumulative settings, as long as no new restrictions that are not in $\mathcal{C}(q)$ are introduced, the flow value F remains the same.

Example 4.2 (Small Reactive (continued))**.** Fig. 4.7 illustrates a reactive example on gridworld introduced previously. The reactive test strategy is constructed from the optimal solution of **MILP-REACTIVE**. The optimization returns cuts on $\mathcal{G}$, which is realized as follows: when the system is at the initial state and the test execution history variable is at q0, the test environment places a restriction as shown in Fig. 4.7b. If the system chooses to visit $\mathtt{I}_1$ first, the restriction does not change even as the test execution history variable updates to q6 (see Fig. 4.7c). Alternatively, if the system visits $\mathtt{I}_2$ first, the test execution history variable updates to q7, and to prevent direct access to goal cell T, the test environment places the restrictions shown in Fig. 4.7d. These restrictions can be implemented either in the instantaneous or the accumulative setting.

Static and Mixed Test Environments: In the special case of test environments consisting of only static obstacles, the solution of **MILP-STATIC** returns a set of edge-cuts which result in a reactive map $\mathcal{C}$ in which restrictions do not change based on the history variable: $\mathcal{C}(q) = \mathcal{C}(q'), \forall q, q' \in \mathcal{B}_\pi.Q$. All system transitions constitute the static area: $T_{\text{sys}}.E = T_{\text{sys}}.E_{\text{static}}$, and the test environment instantaneously places all static obstacles at the start of the test execution:

$$\texttt{Obs}_{\text{static}} \coloneqq \{(u.s, v.s) \in T_{\text{sys}}.E_{\text{static}} \mid (u, v) \in C\}, \forall k \geq 0. \tag{4.34}$$

In the mixed setting, the test strategy is constructed according to Eq. (4.33), and the set of active obstacles are constructed similar to the reactive setting. Restrictions that are in the static area $T_{\text{sys}}.E_{\text{static}}$ can be implemented by placing static obstacles.

Example 4.1 (continued)**.** For the medium-sized grid world example illustrated in Fig. 4.3a, the static test environment is illustrated in Fig. 4.7a. Figure 4.8 illustrates edge-cuts that correspond to static obstacles. There are 14 edge-cuts on $\mathcal{G}$ that correspond to 4 static obstacles on T_{sys}. On $\mathcal{G}$, observe that there is no bypass flow, and the maximum flow after the cuts is $F^* = 3$, corresponding to the three different ways in which the system can be routed through the intermediates.

Algorithm 5 summarizes the following aspects of the framework discussed so far: i) graph construction, ii) routing optimization using flow networks, and iii) construction of a reactive test strategy from the optimization solution. Finally, the following theorem (taken from [107]) shows that the reactive test strategy is feasible and not overly-restrictive when constructed from the optimal solution of the MILP.

Theorem 4.3. If the problem data are not inconsistent (see Section 4.7), the reactive test strategy π_{test} found by Algorithm 5 solves Problem 4.1.

Proof. The test environment informs the choice of the MILP (static, reactive, or mixed). Therefore, the resulting π_{test} will be realizable by the test environment. By construction of G_{sys}, any correct system strategy corresponds to a $\texttt{Path}(\texttt{S}_{\text{sys}}, \texttt{T}_{\text{sys}})$. By Theorem 4.2, at any point during the test execution, if the system has not violated its guarantees, there exists a path on G_{sys} to $\mathtt{T}_{\text{sys}}$. Therefore, there exists a correct system strategy π_{sys}, and resulting trace $\sigma(\pi_{\text{sys}} \times \pi_{\text{test}})$, which corresponds to the path $\vartheta_{\text{sys},n} = (s,q)_0 \dots (s,q)_n$ on G_{sys}, where $(s,q)_0 \in \mathtt{S}_{\text{sys}}$ to $(s,q)_n \in \mathtt{T}_{\text{sys}}$. By Lemma 4.1 any $\mathtt{Path}(\mathtt{S}_{\text{sys}}, \mathtt{T}_{\text{sys}})$ on G_{sys} has a corresponding $\mathtt{Path}(\mathtt{S}, \mathtt{T})$ on G and by Theorem 4.1, the cuts ensure that all such paths on G are routed through the intermediate $\mathtt{I}$. Therefore, for a correct system strategy π_{sys}, the trace $\sigma(\pi_{\text{sys}} \times \pi_{\text{test}}) \models \varphi_{\text{sys}} \wedge \varphi_{\text{test}}$. Thus, π_{test} is feasible and by Proposition 4.2 and Lemma 4.3, π_{test} is not overly-restrictive. Thus, Problem 4.1 is solved. □

The resulting test strategy ensures that as long as the system does not take an incorrect action, there will always exist a path to its goal. However, the system is not aided in reaching the goal either — the test strategy will not block actions that lead to unsafe states. Therefore, a correctly implemented system should be able pass the test, and if the test fails, then it is the fault of the system design.

Algorithm 6: Reactive Test Synthesis

```
1:  procedure TEST SYNTHESIS($T_{\text{sys}}, T_{\text{TA}}, H, \varphi_{\text{sys}}, \varphi_{\text{test}}$)
      Input: system $T_{\text{sys}}$, test agent $T_{\text{TA}}$, test harness $H$, system objective $\varphi_{\text{sys}}$, test
    objective $\varphi_{\text{test}}$
      Output: test agent strategy $\pi_{\text{TA}}$
2:      $T_{\text{sys}}.E_{\text{static}} \leftarrow$ Define from $T_{\text{sys}}$, $T_{\text{TA}}$          ▷ Static area (Eq. (4.35)
3:      $\mathcal{G}, \mathfrak{G}_{\text{sys}}, \mathtt{I}, G \leftarrow$ Setup arguments          ▷ Lines 2-13 in Alg. 5
4:      $\mathtt{C}_{\text{ex}} \leftarrow \emptyset$          ▷ Initialize empty set of excluded solutions
5:      $F_{\max} \leftarrow$ MILP-AGENT($\mathcal{G}, \mathfrak{G}, \mathtt{I}, T_{\textbf{sys}}, H, \mathtt{C}_{\textbf{ex}} = \{\}$)
6:      while True do
7:          $F^*, \mathbf{d}^* \leftarrow$ MILP-AGENT($\mathcal{G}, \mathfrak{G}, \mathtt{I}, T_{\text{sys}}, H, \mathtt{C}_{\text{ex}}$)
8:          if STATUS(MILP) $=$ infeasible then
9:              return infeasible
10:         $C \leftarrow \{(u,v) \in G.E \mid \mathbf{d}^{*(u,v)} = 1\}$          ▷ Cuts on $G$
11:         Obs $\leftarrow$ Define from $C$          ▷ Static Obstacles (Eq. (4.34))
12:         $\mathcal{R} \leftarrow$Define from $C$          ▷ Reactive map (Eq. (4.36))
13:         $\mathbf{A} \leftarrow$ Assumptions (a1)–(a5) from $T_{\text{sys}}$, $T_{\text{TA}}$, $G$, $\varphi_{\text{sys}}$
14:         $\mathbf{G} \leftarrow$ Guarantees (g1)–(g7) from $T_{\text{sys}}$, $T_{\text{TA}}$, $\mathcal{R}$
15:         $\varphi \leftarrow (\mathbf{A} \rightarrow \mathbf{G})$          ▷ Construct GR(1) formula
16:         if REALIZABLE($\varphi$) then
17:             $\pi_{\text{TA}} \leftarrow$ GR1Solve($\varphi$)
18:             return $\pi_{\text{TA}}$, Obs
19:         $\mathtt{C}_{\text{ex}} \leftarrow \mathtt{C}_{\text{ex}} \cup C$
```

Strategy Synthesis for a Dynamic Test Agent

In some test scenarios, it might be beneficial to make use of an available dynamic test agent. Thus, the challenge is to find a tester strategy that corresponds to $\mathcal{C}$ while ensuring that the system's operational environment assumptions are satisfied. To accomplish this, we adapt the **MILP-MIXED** using information about the dynamic test agent. Then, we find the test agent strategy using reactive synthesis and counter-example guided search. From the optimal cuts of **MILP-MIXED** and the resulting reactive map $\mathcal{C}$, we can find states that the test agent must occupy in reaction to the system state. Then, we synthesize a strategy for the dynamic test agent using the Temporal Logic and Planning Toolbox (TuLiP) [108, 109]. If we cannot synthesize a strategy, we use a counterexample-guided approach to exclude the current solution and resolve the MILP to return a different set of optimal cuts until a strategy can be synthesized. Suppose we are given a test agent whose dynamics are given by the transition system T_{TA}, where $T_{\text{TA}}.S$ contains at least one state that is not in $T.S$, denoted as `park`. During the test execution, the test agent can navigate to these `park` states, if necessary. These states are required to synthesize a test agent strategy. From the test agent's transition system T_{TA}, we determine which states in T the test agent can occupy. Static obstacles are used to restrict transitions to states that cannot be occupied by the test agent, and thus the static area is defined as

$$T_{\text{sys}}.E_{\text{static}} \coloneqq \{(u, v) \in T_{\text{sys}}.E \mid v \notin T_{\text{TA}}.S\}. \tag{4.35}$$

Assumption 4.2. In the mixed setting with static obstacles, and a reactive dynamic agent, static obstacles can only restrict transitions in $T_{\text{sys}}.E_{\text{static}}$ as defined in Eq. (4.35).

Adapting the MILP for test agent: Since an agent can only occupy a single state at a time, solutions in which multiple edge cuts can be realized by occupying the same state are incentivized. For this, we introduce the variable $\mathbf{d}_{\text{state}} \in \mathbb{R}_+^{|V|}$, which represents whether an incoming edge into a state is cut. This is captured by the constraint

$$\forall (u, v) \in E, \quad d^{(u,v)} \leq d^v_{\text{state}}, \tag{c11}$$

where $d^v_{\text{state}} \geq 1$ corresponds to at least one incoming edge being cut. The adapted objective is then defined as

$$F - k \sum_{e \in E} d^e - m \sum_{v \in V} d^v_{\text{state}},$$

where $k \leq \frac{1}{1+|E|}$ and $m \leq \frac{1}{|V|(1+|E|)}$. The objective is chosen such that the total number of edge cuts, and the number of nodes that are blocked are minimized. The regularizers are chosen to reflect this order of priority: once the number of edges are minimized, the number of nodes that are cut are minimized. The optimal cuts from the resulting MILP are used to synthesize a reactive test agent strategy as follows. From the optimal cuts C^*, we find the set of static obstacles $\texttt{Obs} \subseteq T.E_{\text{static}}$ according to Eq. (4.34) and the reactive map $\mathcal{R} : \mathcal{B}_\pi.Q \rightarrow T.E$ as follows:

$$\mathcal{R}(q) := \{(s, s') \in T.E \mid (s, s') \notin T.E_{\text{static}} \text{ and } ((s, q), (s', q')) \in C^*\}. \tag{4.36}$$

The reactive map $\mathcal{R}$ is used to synthesize a strategy for the test agent. If no strategy can be found, a counter-example guided approach is used to resolve the MILP.

Reactive Synthesis: From the solution of the MILP, we now construct the specification to synthesize the test agent strategy using TuLiP. In particular, we construct a GR(1) formula with assumptions being our model of the system and the guarantees capturing requirements on the test agent. Note that we are synthesizing a strategy for the test agent, where the environment is the system under test. The variables needed to define the GR(1) formula consist of variables capturing the system's state $\mathtt{x}_{\text{sys}} \in T.S$ and $\mathsf{q}_{\text{hist}} \in \mathcal{B}_\pi.Q$, which track how system transitions affect the history variable q. The test agent state is represented in the variable $\mathtt{x}_{\text{TA}} \in T_{\text{TA}}.S$.

First, we set up the subformulae constituting the assumptions on the system model. The initial conditions of the system are defined as

$$(\mathtt{x}_{\text{sys}} = s_0 \wedge \mathsf{q}_{\text{hist}} = q_0), \tag{a1}$$

where $s_0 \in T.S_0$ and $\mathcal{B}_\pi.Q_0$. We define the dynamics of the system and the history variable for each state $(s, q) \in G.S$ as follows:

$$\square\Big((\mathtt{x}_{\text{sys}} = s \wedge \mathsf{q}_{\text{hist}} = q) \rightarrow \bigvee_{\substack{(s',q')\in \\ \texttt{succ}(s,q)}} \bigcirc(\mathtt{x}_{\text{sys}} = s' \wedge \mathsf{q}_{\text{hist}} = q')\Big), \tag{a2}$$

where $\texttt{succ}(s, q)$ denotes the successors of state $(s, q) \in G.S$. For simplicity, we choose a turn-based setting, in which each player will only take their action if it is their turn. To track this, we introduce the variable $\texttt{turn} \in \mathbb{B}$ as a test agent variable. For the system, this is encoded as remaining in place when $\texttt{turn} = 1$:

$$\bigwedge_{s\in T.S} \square\Big((\mathtt{x}_{\text{sys}} = s \wedge \texttt{turn} = 1) \rightarrow \bigcirc(\mathtt{x}_{\text{sys}} = s)\Big). \tag{a3}$$

If a turn-based setup is not used, we need to synthesize a Moore strategy for the test agent since it should account for all possible system actions. The system objective φ_{sys} can be encoded as the formula

$$\square\diamondsuit(\mathtt{x}_{\text{sys}} = x_{\text{goal}}) \wedge \varphi_{\text{aux}}, \tag{a4}$$

where x_{goal} is the terminal state of the system and a reachability objective specified in φ_{sys}. The other objectives specified in φ_{sys} are transformed to their respective GR(1) forms in φ_{aux}. This transformation of LTL formulas into GR(1) form is detailed in [110]. In addition, the system is expected to safely operate in the test agent's presence. The set of states where collision is possible is denoted by $S_{\cap} := T.S \cap T_{\text{TA}}.S$. Thus, the safety formula encoding that the system will not collide into the tester is given as:

$$\bigwedge_{s \in S_{\cap}} \square\Big(\mathtt{x}_{\text{TA}} = s \rightarrow \bigcirc\neg(\mathtt{x}_{\text{sys}} = s)\Big). \tag{a5}$$

Equations (a1)– (a5) represent the test agent's assumptions on the system model. Next, we describe the subformulas for the guarantees of the GR(1) specification. The initial conditions for the test agent are

$$\bigvee_{s \in T_{\text{TA}}.S_0} \mathtt{x}_{\text{TA}} = s. \tag{g1}$$

The test agent dynamics are represented by

$$\square\Big((\mathtt{x}_{\text{TA}} = s) \rightarrow \bigvee_{(s,s') \in T_{\text{TA}}.E} \bigcirc(\mathtt{x}_{\text{TA}} = s')\Big). \tag{g2}$$

The test agent can also move only in its turn and will remain stationary when $\mathtt{turn} = 0$:

$$\bigwedge_{s \in T_{\text{TA}}.S} \square\Big((\mathtt{x}_{\text{TA}} = s \wedge \mathtt{turn} = 0) \rightarrow \bigcirc(\mathtt{x}_{\text{TA}} = s)\Big). \tag{g3}$$

The $\mathtt{turn}$ variable alternates at each step:

$$(\mathtt{turn} = 1) \rightarrow \bigcirc(\mathtt{turn} = 0)\ \wedge (\mathtt{turn} = 0) \rightarrow \bigcirc(\mathtt{turn} = 1). \tag{g4}$$

To satisfy the system assumptions (Def. 4.6), the test agent should not adversarially collide into the system. This is captured via the following safety formula,

$$\bigwedge_{s \in S_{\cap}} \square\Big(\mathtt{x}_{\text{sys}} = s \rightarrow \bigcirc\neg(\mathtt{x}_{\text{TA}} = s)\Big). \tag{g5}$$

Now, we enforce the optimal cuts found from the MILP. To enforce cuts reactively during the test execution, the states occupied by the system are defined as follows,

$$\bigwedge_{q \in \mathcal{B}_\pi.Q} \bigwedge_{(s,s') \in \mathcal{R}(q)} \square\Big((\mathtt{x}_{\text{sys}} = s \wedge \mathtt{q}_{\text{hist}} = q \wedge \mathtt{turn} = 0) \rightarrow (\mathtt{x}_{\text{TA}} = s')\Big). \tag{g6}$$

Essentially, for some history variable q, if $(s, s') \in \mathcal{R}(q)$ is an edge cut, then the test agent must occupy the state s' when the system is in the state s when the test execution is at history variable q. However, the test agent should not introduce any additional restrictions on the system, which is formulated as

$$\bigwedge_{q \in \mathcal{B}_\pi.Q} \bigwedge_{\substack{(s,s') \in T.E \\ (s,s') \notin \mathcal{R}(q)}} \square\Big((\mathtt{x}_{\text{sys}} = s \wedge \mathtt{q}_{\text{hist}} = q \wedge \mathtt{turn} = 0) \rightarrow \neg(\mathtt{x}_{\text{TA}} = s')\Big). \tag{g7}$$

Intuitively, this corresponds to the requirement that the tester agent shall not restrict system transitions that are not part of the reactive map $\mathcal{R}$. A test agent strategy that satisfies the above specifications is guaranteed to not restrict any system action unnecessarily. However, the test agent can occupy a state that is not adjacent to the system and block all paths to the goal from the system's perspective. This could lead the system to not making any progress towards the goal at all, resulting in a livelock. To avoid this, we characterize the livelock condition as a safety constraint that the test agent must satisfy (e.g., if it occupies a livelock state, it must not occupy it in the next step). The specific safety formula that captures the livelock depends on the example. We find the states where the tester would block the system from reaching its goal $T.S_{\text{block}} \subseteq T_{\text{TA}}.S$. The following condition ensures that it will only transiently occupy blocking states:

$$\bigwedge_{s \in T.S_{\text{block}}} \square\Big(\mathtt{x}_{\text{TA}} = s \rightarrow \bigcirc\neg(\mathtt{x}_{\text{TA}} = s)\Big). \tag{g8}$$

Therefore, we synthesize a test agent strategy π_{TA} for the GR(1) formula with assumptions (a1)–(a5) and guarantees (g1)–(g8).

Counterexample-guided Search: The MILP can have multiple optimal solutions, some of which may not be realizable for the test agent. If the GR(1) formula is unrealizable, we exclude the solution and re-solve the MILP until we find a realizable GR(1) formula. In particular, every new set of optimal cuts C that is unrealizable is added to the set $\mathtt{C}_{\text{ex}}$. Then, the MILP is resolved with an additional set of affine constraints as follows,

$$\sum_{e \in E} d^e - \sum_{e \in C} d^e \geq 1, \; \forall C \in \mathtt{C}_{\text{ex}}. \tag{c12}$$

This corresponds to removing the solution C from the constraint set. The adapted MILP is then defined as follows:

MILP-AGENT:

$$\max_{\substack{\mathbf{f},\mathbf{d},\mathbf{d}_{\text{state}},\boldsymbol{\mu},\\ \mathbf{f}_{\text{sys}}^{(q,\mathbf{s})}\ \forall q\in\mathcal{B}_\pi.Q,\\ \forall \mathbf{s}\in \mathbf{S}_{\mathcal{G}_{\text{sys}}}(q).}} \quad F - \frac{1}{1+|E|}\sum_{e\in E} d^e - \frac{1}{(1+|E|)|V|}\sum_{v\in V} d^v_{\text{state}} \tag{4.37}$$
$$\text{s.t.} \quad \text{(c1)-(c9), (c11), (c12).}$$

This process is repeated until a strategy is synthesized or the **MILP-AGENT** becomes infeasible. Algorithm 6 summarizes the approach for synthesizing the test agent strategy. The terms $F_{\max}$ and F^* (lines 5 and 7 in Algorithm 6) denote the maximum possible flow before and after accounting for counterexamples, respectively. The following lemma and theorem are adapted from [107].

Lemma 4.4. Let π_{TA} be the test agent strategy and let $\texttt{Obs}$ satisfying Assumption 4.2 be the set of static obstacles synthesized from the optimal solution C of **MILP-AGENT** according to the GR(1) formula with assumptions (a1)–(a5) and guarantees (g1)–(g8). Let π_{test} be the reactive test strategy corresponding to the optimal cuts C^*. Then π_{TA} and $\texttt{Obs}$ realize π_{test}.

Proof. By construction in Eqs. (4.32), (4.34), (4.36), we have that $\mathcal{C}(q) = \mathcal{R}(q) \cup \texttt{Obs}$ for all history variables $q \in \mathcal{B}_\pi.Q$. Due to guarantee (g6), the synthesized test agent strategy restricts the transitions in $\mathcal{R}(q)$. The test agent is also prohibited from restricting any other transitions by the guarantee (g7). Therefore, at each step of the test execution, the system actions restricted as a result of π_{TA} and static obstacles $\texttt{Obs}$ directly correspond to the system actions restricted by the test strategy π_{test}. □

Theorem 4.4. Algorithm 6 is sound with respect to Problem 4.2.

Proof. The test agent strategy is synthesized to satisfy guarantees (g1)-(g8). The guarantees (g1)-(g4) specify the dynamics of the test agent, which satisfies **A1**. The safety guarantee (g5) satisfies **A2**. Guarantees (g6) and (g7) realize the optimal cuts from **MILP-AGENT**. Due to constraint (c8) the optimal cuts ensure that there always exists a path on G_{sys}. Together with guarantee (g8), this results in π_{TA} satisfying assumptions **A3** and **A4**. By Lemma 4.4, π_{TA} is a realization of a feasible π_{test} that is not overly-restrictive.

In Algorithm 6, each iteration of **MILP-AGENT** is solved to optimality while excluding the counterexamples. If **MILP-AGENT** returns with $F^* = F_{\max}$, then π_{TA} corresponds to a π_{test} that is not overly-restrictive. By iteratively removing counterexamples, the agent strategy is synthesized for a reactive test strategy with the highest possible $F^* \leq F_{\max}$. This is valid under Assumption 4.2, which allows static obstacles only on transitions that cannot be restricted by the test agent. In **MILP-AGENT**, this condition is enforced by applying constraint (c9) on the static area $T_{\text{sys}}.E_{\text{static}}$. □

If a matching test agent strategy is found for the maximum possible F, the test agent strategy and obstacles, π_{TA} and $\texttt{Obs}$, correspond to a not overly-restrictive reactive test strategy π_{test} possible for that test environment. In future work, we will exploring relaxing Assumption 4.2.

4.9 Complexity Analysis

This framework comprises of three parts: automata-theoretic graph construction, flow-based MILP to solve the routing optimization, and finally reactive synthesis to match the solution of the optimization to a test agent strategy. The automata-theoretic framework includes construction of Büchi automata from specifications, which can be doubly exponential in the length of the formula in the worst-case [60]. Then, construction of the product graphs relies on building the Cartesian product of the transition system and the automaton. The Cartesian product implementation in this work has a worst-case time complexity of $O(|T.S|^2|\mathcal{B}_\pi.Q|^2)$. In this section, I will discuss the computational complexity of the routing optimization, and prove that the routing optimization is an NP-hard problem. Finally, the solution of the routing optimization is mapped to a strategy of the test agent via GR(1) synthesis, which has time complexity $O(|N|^3)$, where N is the number of states required to define the GR(1) formula.

To establish the computational complexity of the routing optimization, we will first look at the special case of static obstacles, and then extend the proof to the setting with reactive obstacles. Consider the problem data of the routing optimization once again: a graph $G = (V, E)$ with specially denoted nodes $\texttt{S}$, $\texttt{I}$, and $sink$, and the corresponding flow network $\mathcal{G}$. A bypass path on G is some $Path(\texttt{S}, \texttt{T})$ which does not contain an intermediate node $v \in \texttt{I}$. For all edges $e \in E \setminus E(\texttt{I})$, the Boolean variable d^e carries information on whether the edge e is cut (i.e., $d^e = 1$), and the set $C \subset E$ represents the set of all edges that are cut. The flow F on $\mathcal{G}$ is the maximum flow value from source $\mathtt{S}$ to $\mathtt{T}$, computed after accounting for the edge cuts.

The static and reactive obstacle settings are based on the grouping of edges on G, which become important for checking system feasibility. Static obstacles are grouped by the corresponding transition in the system transition T since they are present for the entire test duration. In particular, the static grouping $\mathtt{Gr}_{\text{static}} : T.E \to G.E$ groups all edges in G that correspond to the same system transition in T:

$$\mathtt{Gr}_{\text{static}}((s, s')) := \{(u, v) \in G.E \mid u.s = s, v.s = s'\}. \tag{4.38}$$

For some edge $(s, s') \in T.E$, all the corresponding edges in G, that is, all edges $e \in \mathtt{Gr}_{\text{static}}((s, s'))$ must have the same d^e value. Similarly, in the reactive setting, edges are grouped by the history variable q, as given in Eq. (4.28). System feasibility can then be checked by applying these groupings onto copies of G or G_{sys}, as detailed in Remark 4.5.

For purposes of clearly stating the static the optimization and decision versions of the routing problem, we introduce the label of a valid set of edge cuts. In the static setting, a *valid* set of edge cuts C when applied to G is such that: i) there are no bypass paths, ii) there exists at least one path from $\mathtt{S}$ to $\mathtt{T}$, and iii) edges of G respect the static grouping $\mathtt{Gr}_{\text{static}}$. Note that there can exist graphs G for which there does not exist a valid set of edge cuts in the static setting. These are graphs for which we cannot synthesize a test comprising only static obstacles to realize the test objective. One such example is Beaver Rescue. Now, the optimization version of the routing problem for the special case of static obstacles is stated as follows,

Problem 4.3 (Routing Problem, Static Setting (Optimization))**.** Given a graph G, find a valid set of edge cuts C in the static setting such that the resulting maximum flow F is maximized over all possible sets of edge cuts, and such that $|C|$ is minimized for the flow F.

In other words, the optimization follows a two-step procedure: first, identify a valid set of edge cuts C to maximizes the flow F, and second, tie-break between the optimal candidates C to choose one with the smallest cardinality $|C|$. The decision version of Problem 4.3 can be stated as follows.

Problem 4.4 (Routing Problem, Static Setting (Decision))**.** Given a graph G and an integer $M \geq 0$, does there exist a valid set of edge cuts C in the static setting such that $|C| \leq M$?

Lemma 4.5. A solution to Problem 4.3 can be used to construct a solution for Problem 4.4 in polynomial time.

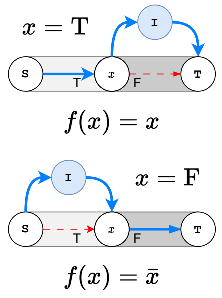
Proof. The solution of Problem 4.3 returns a set of valid edge cuts C. Thus for any given integer $M \geq 0$, we can check in polynomial time if $|C| \leq M$. □

Basics of Complexity Theory: Finding the complexity class of Problem 4.4 will help in determining the complexity of Problem 4.3 because by Lemma 4.5, we can infer that Problem 4.3 is at least as hard as Problem 4.4. The class of NP problems consists of those that are verifiable in a time polynomial to the size of the input to the problem [80]. A problem is said to be in the class of NP-complete problems if: i) it is in the class NP, and ii) it is as hard as any problem in NP. Polynomial-time algorithms for solving NP-complete problems would exist only if P=NP. The class of NP-hard problems are those that are as hard as a problem in NP. In this section, I will show that Problem 4.4 is NP-complete, and by extension that Problem 4.6 is an NP-hard problem in the size of the input: product graph G. This would also support the choice of a mixed-integer linear programming framework to solve the routing optimization, since MILPs belong to the class of NP-hard problems as well.

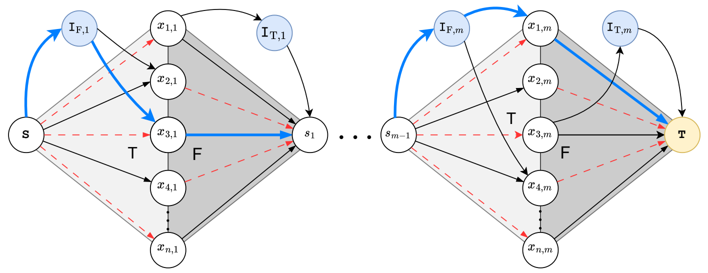
To show that Problem 4.4 is NP-complete, we have to establish its membership in NP, and then give a polynomial-time reduction of a problem in NP to Problem 4.4. We will choose the 3-SAT problem and give a polynomial-time reduction algorithm that maps any instance of the 3-SAT problem to an instance of Problem 4.4. This reduction algorithm is such that a solution to the constructed instance of Problem 4.4 can be transformed in polynomial-time to a solution of the 3-SAT instance.

Lemma 4.6. Problem 4.4 is in the class NP.

Proof. Given a solution C, we need to show that verifying that it is a valid set of edge-cuts for the static setting can be done in polynomial-time. In reference to the definition of a valid set of edge-cuts, it can be checked in polynomial-time that there are no bypass paths when the edges in C are cut from G. This would involve a simple check (e.g., via any max-flow algorithm) to verify zero maximum flow from $\mathtt{S}$ to $\mathtt{T}$ on the bypass network $\mathcal{G}_{byp} = (V \setminus \mathtt{I}, E \setminus E(\mathtt{I}), \mathtt{S}, \mathtt{T})$. Similarly, condition (ii) can also be checked in polynomial-time by running a max-flow algorithm on $\mathcal{G}$ and verifying that the max-flow is at least 1. Finally, condition (iii) can be checked in polynomial-time by iterating over all edges $e \in T.E$, and checking that exactly one of the following in true: a) $\mathtt{Gr}_{\text{static}}(e) \subseteq C$ or b) $\mathtt{Gr}_{\text{static}}(e) \cap C = \emptyset$. □



(a) Graphs matching formulas with a single variable x.



(b) Graph resulting from a reduction of the 3SAT formula $f(x_1, \ldots, x_5)$, where the resulting edge cuts correspond to the truth assignment of the variables $x_1, \ldots, x_5$.

Figure 4.10: Graphs constructed from a 3SAT formula, where a truth assignment for the variables can be found using the network flow approach for static obstacles.

Now, we introduce the 3-SAT problem .

Definition 4.17 (3-SAT [111]). Let $x_1, \ldots, x_n$ be propositions that can evaluate to true or false. A literal is a proposition x_i or its negation. The propositional logic formula $f(x_1, \ldots, x_n) := \bigwedge_{j=1}^{m} c_j$ is a conjunction of clauses $c_1, \ldots, c_m$, where each clause is a disjunction of three Boolean literals. A solution to the 3-SAT problem is an algorithm that returns *True* if there exists a satisfying assignment to $f(x_1 \ldots, x_n)$ and *False*.

Outline of the Reduction Algorithm: Given any 3-SAT formula, we will construct a product graph, an instance of 4.4 in polynomial-time. The product graph is constructed modularly — each clause in the 3-SAT formula corresponds to a sub-graph in the larger product graph (Construction 1). Then, using Construction **??**, the sub-graphs are connected to form the product graph instance to Problem 4.4. Finally, we will prove that any algorithm used to solve Problem 4.4 can be used to solve the 3-SAT problem, thus showing that Problem 4.4 is at least as hard as a problem in NP. Consequently, Problem 4.4 can be solved in polynomial-time in the size of the product graph only if there exists a polynomial-time algorithm for 3-SAT which is only possible if P=NP.

Construction 1 (Clause to Sub-graph)**.** For each clause c_j in the given 3-SAT formula, construct the following sub-graph. First, introduce nodes $x_{1,j}, \ldots, x_{n,j}$ for each of the Boolean propositions $x_1, \ldots, x_n$ that constitute the 3-SAT formula. If $j = 0$, introduce the nodes s_0 and s_1, otherwise introduce the node s_j. For all $j \in \{1, \ldots, m\}$, the nodes s_{j-1} and s_j represent the beginning and end of each sub-graph. Additionally, introduce two more nodes: $\texttt{I}_{\text{T},j}$ and $\texttt{I}_{\text{F},j}$. These nodes will serve as intermediate nodes in the constructed graph.

Second, the edges of the sub-graph are added as follows. The intermediate nodes are connected to the start and end nodes of the sub-graph via the directed edges: $(s_{j-1}, \texttt{I}_{\text{F},j})$ and $(\texttt{I}_{\text{T},j}, s_j)$. Next, for each $x_{i,j}$, add the directed edges: $(s_{j-1}, x_{i,j})$ and $(x_{i,j}, s_j)$. If neither x_i nor its negation $\bar{x}_i$ appear in the clause c_j, then these are the only directed edges connected to the node $x_{i,j}$ in the sub-graph. If the literal x_i appears, then we add the directed edge $(x_{i,j}, \texttt{I}_{\text{T},j})$, and if a negated literal $\bar{x}_i$ appears, we add the directed edge $(\texttt{I}_{\text{F},j}, x_{i,j})$.

From Construction 1, edge-cuts on the sub-graph are related to the Boolean valuations of the propositions as follows. Either the incoming or outgoing edge to each node $x_{i,j}$ must be cut. As illustrated in Fig. 4.10a, if the edge $(s_{j-1}, x_{i,j})$ remains in the sub-graph (and $(x_{i,j}, s_j)$ is cut), this implies that the proposition x_i is assigned the value $True$. Similarly, if the edge $(x_{i,j}, s_j)$ remains, then the proposition x_i is assigned the value $False$. Construction 1 ensures that a satisfying assignment to the clause c_j implies that there exists a $Path(s_{j-1}, s_j)$ and all such paths are routed through the intermediates $(s_{j-1}, \texttt{I}_{\text{F},j})$ and $(\texttt{I}_{\text{T},j}, s_j)$ (see Fig. 4.10b). An assignment that evaluates clause c_j to $False$ would only be possible if there was no $Path(s_{j-1}, s_j)$. The full graph can be constructed by stitching together the individual sub-graphs built using Construction 1.

Construction 2 (Reduction)**.** Given a 3-SAT problem with n Boolean propositions and m clauses, construct the sub-graph for each clause according to Construction 1. Denote the node s_0 as the source $\texttt{S}$ and s_m as the sink $\texttt{T}$. The node s_{j-1} is common between the sub-graphs of clauses c_{j-1} and c_j. Let the integer variable M be set to $m \times n$. Note that the constructed graph has $O(mn)$ edges and is constructed in polynomial-time in the number of propositions and clauses of the given 3-SAT formula.

In addition to constructing the graph, two groups of edges for each Boolean proposition x_i are tracked: i) an incoming group of edges $\{(s_{j-1}, x_{i,j}) \mid 1 \leq j \leq m\}$, and ii) an outgoing group of edges $\{(s_{j-1}, x_{i,j}) \mid 1 \leq j \leq m\}$. All edges in a group must share the same edge-cut value, corresponding to the static grouping map $\texttt{Gr}_{\text{static}}$. By imposing this constraint, the truth assignment to Boolean propositions across literals can be guaranteed to be the same.

The reduction algorithm takes as input a 3-SAT formula, and applying Constructions 1 and 2, returns a graph with source $\mathtt{S} = s_0$, $\mathtt{I} = \bigcup_{j=1}^{m} \{\mathtt{I}_{\mathrm{T},j}, \mathtt{I}_{\mathrm{F},j}\}$, and edges are grouped according to Construction 2. Now, we will show that for the static setting, the routing problem is NP-hard. The following proof of Theorem 4.5 is taken from [107].

Theorem 4.5. Problem 4.4 is NP-complete.

Proof. We will show that Problem 4.4 is NP-hard by showing that Construction 2 is a correct polynomial-time reduction of the 3-SAT problem to Problem 4.4 i.e., any polynomial-time algorithm to solve Problem 4.4 can be used to solve 3-SAT in polynomial-time. Consider the graph constructed by Construction 2 for any propositional logic formula. The valid set of edge cuts C on this graph with cardinality $|C| \leq M$ is a witness for Problem 4.4. A witness for the 3-SAT formula is an assignment of the variables $x_1, \ldots, x_n$. A witness to a problem is *satisfying* if the problem evaluates to *True* under that witness. Next, we show that a valid set of edge cuts C is a satisfying witness for Problem 4.4 iff the corresponding assignment to variables $x_1, \ldots, x_n$ is a satisfying witness for the 3-SAT formula.

First, consider a satisfying witness for Problem 4.4. By Construction 2, the cardinality of the witness, $|C| = m \times n$ will be exactly M, which is the minimum number of edge cuts required to ensure no bypass paths on the constructed graph. This implies that each variable x_i has a Boolean assignment. By Construction 1, a strictly positive flow on the sub-graph of clause c_j implies that c_j is satisfied. By Construction 2, a strictly positive flow through the entire graph implies that all clauses in the 3-SAT formula are satisfied. Therefore, a satisfying witness to the 3-SAT formula can be constructed in polynomial-time from a satisfying witness for an instance of Problem 4.4.

Next, we consider a satisfying witness for the 3-SAT formula. The Boolean assignment for each variable x_i corresponds to edge cuts on the graph (see Fig. 4.10b). Any Boolean assignment ensures that there is no bypass path on the graph since either all incoming edges or all outgoing edges for each variable x_i are cut. This also corresponds to the minimum number of edge cuts required to cut all bypass paths, corresponding to $|C| = m \times n$. By Construction 1, a satisfying witness corresponds to a Path(s_{j-1}, s_j) on the sub-graph for each clause c_j. By Construction 2, observe that there exists a strictly positive flow on the graph. Thus, we can construct a satisfying witness to an instance of Problem 4.4 in polynomial time from a satisfying witness to the 3-SAT formula. Therefore, any 3-SAT problem reduces to an instance of Problem 4.4, and thus, Problem 4.4 is NP-hard. Additionally, Problem 4.4 is NP-complete since we can check the cardinality of C, and whether C is a valid set of edge cuts in polynomial time. □

Corollary 4.1. Problem 4.3 is NP-hard [112].

Proof. By Theorem 4.5, Problem 4.4 is NP-complete, and therefore by Lemma 4.5, Problem 4.3 is NP-hard. □

These insights can be extended to the complexity analysis for the reactive setting. In the reactive setting, a valid set of edge cuts is defined similar to the static setting, except in the grouping constraint that the edges must respect, as detailed in Remark 4.5, and restated below for clarity.

Recall that $\texttt{Gr}(q)$ is the set of possible system transitions in the history variable q. All restrictions during history variable q are a subset of $\texttt{Gr}(q)$, and in the accumulative placement of reactive constraints, they are all realized in the worst-case. Therefore, the reactive feasibility constraints (c8) checks if there is a $Path(\texttt{s}_{\text{sys}}, T_{\text{sys}})$ on $\mathcal{G}_{\text{sys}}$ (i.e., from the system perspective, are we ensuring that there is a path to the goal) when all reactive constraints are accumulated and projected onto $\mathcal{G}_{\text{sys}}$. Instead of checking on $\mathcal{G}_{\text{sys}}$, we can verify the same condition by checking on $\mathcal{G}$ by statically mapping the edges $\texttt{Gr}(q)$.

Definition 4.18 (Static Mapping)**.** For a network $\mathcal{G}$, let $E' \subseteq \mathcal{G}.E$ be a set of edges in which each edge has an associated edge-cut value d^e. The network $\mathcal{G}$ is *statically mapped* with respect to E' if for every edge $(u, v) \in E'$, the following is true:

$$d^{(u',v')} = d^{(u,v)}, \ \forall (u', v') \in \texttt{Gr}_{\text{static}}((u.s, v.s)). \tag{4.39}$$

The static mapping connects restrictions on the same system action across history variables. The feasibility networks are necessary to ensure that the system restrictions do not block the system from its goal.

Definition 4.19 (Reactive Feasibility Networks)**.** For each $q \in \mathcal{B}_\pi.Q$ and for every possible system source $\texttt{s} \in \texttt{S}_{G_{\text{sys}}}(q)$, introduce a copy of $\mathcal{G}_{\text{sys}}$ denoted $\mathcal{G}_{\text{sys}}^{(q,\texttt{s})} = (V_{\text{sys}}, E_{\text{sys}}, \texttt{s}, \texttt{T}_{\text{sys}})$. The set of edges in $\mathcal{G}$ that map to some edge $(u_{\text{sys}}, v_{\text{sys}}) \in E_{\text{sys}}$ is

$$\mathcal{P}_{E_{\text{sys}} \to E}((u_{\text{sys}}, v_{\text{sys}})) = \{(u, v) \in E | \mathcal{P}_{G \to G_{\text{sys}}}(u) = u_{\text{sys}} \text{ and } \mathcal{P}_{G \to G_{\text{sys}}}(v) = v_{\text{sys}}\}.$$

Note that multiple edges on $\mathcal{G}$ can map to the same edge on $\mathcal{G}_{\text{sys}}$. Furthermore, reactive restrictions at history variable q are all contained in $\texttt{Gr}(q)$. Therefore, if one of the edges in $\mathcal{P}_{E_{\text{sys}}\to E}((u_{\text{sys}}, v_{\text{sys}})) \cap \texttt{Gr}(q)$ is restricted, then the edge $(u_{\text{sys}}, v_{\text{sys}})$ is restricted on the copy $\mathcal{G}_{\text{sys}}^{(q,\mathtt{s})}$. Let $\mathbf{d}_{\text{sys}} \in \mathbb{B}^{|E_{\text{sys}}|}$ denote the cut values of edges on $\mathcal{G}_{\text{sys}}^{(q,\mathtt{s})}$. The system restrictions on $\mathcal{G}$ are mapped to edge-cuts on $\mathcal{G}_{\text{sys}}^{(q,\mathtt{s})}$ only for the history variable q:

$$\begin{aligned} d_{\text{sys}}^{(u_{\text{sys}},v_{\text{sys}})} = \max \quad & d^{(u,v)} \\ \text{s.t.} \quad & (u,v) \in \mathcal{P}_{E_{\text{sys}}\to E}((u_{\text{sys}}, v_{\text{sys}})) \cap \texttt{Gr}(q). \end{aligned} \tag{4.40}$$

The *reactive feasibility networks* $\mathfrak{G}_{\text{sys}}$ is the set of graphs $\mathcal{G}_{\text{sys}}^{(q,\mathtt{s})}$ whose edge cut values are mapped according to Eq. (4.40):

$$\begin{aligned} \mathfrak{G}_{\text{sys}} := \{\mathcal{G}_{\text{sys}}^{(q,\mathtt{s})} = & (V_{\text{sys}}, E_{\text{sys}}, \mathtt{s}, \mathtt{T}_{\text{sys}}) | q \in \mathcal{B}_\pi.Q,\ \mathtt{s} \in \mathtt{S}_{G_{\text{sys}}}(q) \text{ and} \\ & \mathcal{G}_{\text{sys}}^{(q,\mathtt{s})} \text{ is mapped according to Eq. (4.41)}\}. \end{aligned} \tag{4.41}$$

In Alg. 5, lines 9–13 correspond to the construction of $\mathfrak{G}_{\text{sys}}$. In the implementation of the optimizations, edge cut variables $\mathbf{d}_{\text{sys}}$ for system feasibility networks are not defined since this would dramatically increase the number of integer variables. Instead, edge cuts on $\mathcal{G}$ are directly used to cut the flow on the feasibility networks (see Eq. (c8)).

When the cut-set C is found for $\mathcal{G}$, the *reactive feasibility condition* requires that for every reactive feasibility network $\mathcal{G}_{\text{sys}}^{(q,\mathtt{s})} \in \mathfrak{G}_{\text{sys}}$, there exists a path from source $\mathtt{s}$ to target T_{sys} after the cuts C are applied to $\mathcal{G}^{(q,\mathtt{s})}$ via the mapping in Eq. (4.40). For the purpose of proving computational complexity, it is easier to reduce from 3-SAT if reasoning over graphs with similar structures. Thus, we consider the following check for reactive feasibility which reasons over copies of $\mathcal{G}$ instead of $\mathcal{G}_{\text{sys}}$.

Definition 4.20 (Statically mapped Reactive Feasibility Networks)**.** For each $q \in \mathcal{B}_\pi.Q$ and for every possible source $\mathtt{s} \in \mathtt{S}_G(q)$, introduce a copy of $\mathcal{G}$ denoted $\mathcal{G}^{(q,\mathtt{s})} := (V, E, \mathtt{s}, \mathtt{T})$. Each network $\mathcal{G}^{(q,\mathtt{s})}$ is statically mapped with respect to the edges $\texttt{Gr}(q)$. The *statically mapped reactive feasibility networks* $\mathfrak{G}$ is the set of all $\mathcal{G}^{(q,\mathtt{s})}$:

$$\begin{aligned} \mathfrak{G} := \{\mathcal{G}^{(q,\mathtt{s})} = & (\mathcal{G}.V, \mathcal{G}.E, \mathtt{s}, \mathtt{T}) | q \in \mathcal{B}_\pi.Q,\ \mathtt{s} \in \mathtt{S}_G(q) \text{ and} \\ & \mathcal{G}^{(q,\mathtt{s})} \text{ is statically mapped with respect to } \texttt{Gr}(q)\}. \end{aligned} \tag{4.42}$$

In other words, all edges restricted (i.e., $d^e = 1$) at history variable q are statically mapped on $\mathcal{G}^{(q,\mathtt{s})}$. When the cut-set C is found for $\mathcal{G}$, the *reactive feasibility* *condition (via static mapping)* requires that for every statically mapped reactive feasibility network $\mathcal{G}^{(q,\mathtt{s})} \in \mathfrak{G}$, there exists a path from source $\mathtt{s}$ to target T after the cuts C are applied to $\mathcal{G}^{(q,\mathtt{s})}$ via the static mapping. Checking for the reactive feasibility condition (via static mapping) on $\mathfrak{G}$ is equivalent to checking the reactive feasibility condition on $\mathfrak{G}_{\text{sys}}$.

Theorem 4.6. The reactive feasibility condition (via static mapping) on $\mathfrak{G}$ is true iff the reactive feasibility condition on $\mathfrak{G}_{\text{sys}}$ is true.

Proof. Suppose the reactive feasibility condition (via static mapping) is true. Then, the corresponding graphs $\mathcal{G}_{\text{sys}}$ will also have a path by theorem relating to why static checks are enough. If the reactive feasibility condition on $\mathfrak{G}_{\text{sys}}$ is true, then there exists a corresponding path on $\mathcal{G}$ as well due to lemma 4.1. $\square$

For each history variable $q \in \mathcal{B}_\pi.Q$ and for every possible system source $\mathtt{s} \in \mathtt{S}_G(q)$, introduce a copy of $\mathcal{G}$ denoted $\mathcal{G}^{(q,\mathtt{s})} = (V, E, \mathtt{s}, \mathtt{T})$. On this copy $\mathcal{G}^{(q,\mathtt{s})}$, we statically map the edges $\mathtt{Gr}(q)$, and check if the flow from $\mathtt{s}$ to $\mathtt{T}$ is at least 1. In the reactive setting, a *valid* set of edge cuts C when applied to G is such that: i) there are no bypass paths, ii) there exists at least one path from $\mathtt{S}$ to $\mathtt{T}$, and iii) edges of G respect the reactive grouping condition (via static mapping). Finally, the optimization and decision problems in the reactive setting can be stated as follows.

Problem 4.5 (Routing Problem, Reactive Setting (Optimization))**.** Given a graph G, find a valid set of edge cuts C in the reactive setting such that the resulting maximum flow F is maximized over all possible sets of edge cuts, and such that $|C|$ is minimized for the flow F.

Problem 4.6 (Routing Problem, Reactive Setting (Decision))**.** Given a graph G and an integer $M \geq 0$, does there exist a valid set of edge cuts C in the reactive setting such that $|C| \leq M$?

Once again, consider the 3-SAT reduction from the static setting. This reduction will be adapted to construct a polynomial-time reduction of 3-SAT to an instance of Problem 4.6 with a single history variable q. Similar to the static setting, I will prove that Problem 4.6 is NP-complete in the size of the product graph G. To do this, we will first establish that Problem 4.6 is in the class of NP problems.

Lemma 4.7. Problem 4.6 is in the class of NP problems.

Proof. This proof follows similarly to the proof of Lemma 4.6. Given a solution C, we need to show that verifying C to be a valid set of edge cuts for the reactive setting can be carried out in polynomial-time. Conditions (i) and (ii) can be checked similarly as in, and condition (iii) can also be checked in polynomial-time. □

Similar to the static setting, 3-SAT can be reduced to an instance of Problem 4.6. In this chapter, I will construct the reduction to an instance of Problem 4.6 with a single history variable.

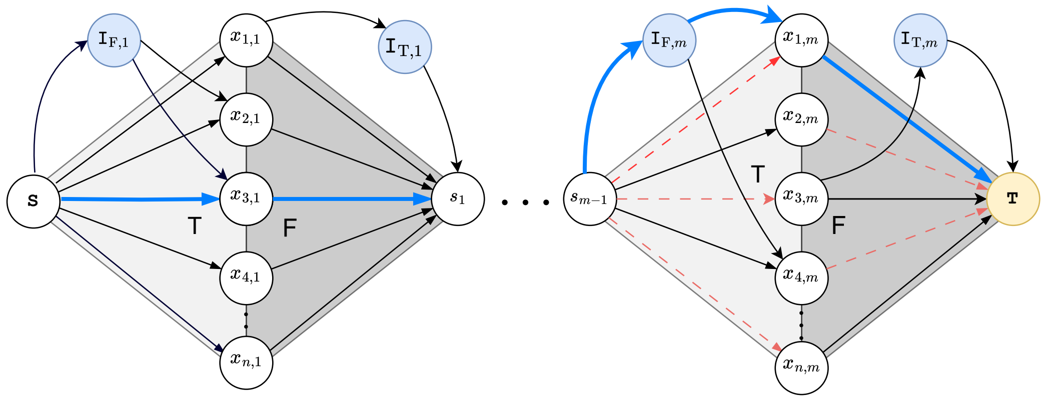
Construction 3 (Reduction from 3-SAT to Problem 4.6 with single history variable q)**.** Given a 3-SAT formula with n propositions and m clauses. Construct a graph, denoted $G^{(q,\texttt{S})}$, according to Construction 2. In addition, construct a graph G with nodes and edges according to Construction 2 without applying the constraint that all edges in a group must have the same edge-cut value. The graph $G^{(q,\texttt{S})}$ serves a reactive feasibility network (via static mapping) where S is the source at history variable q. On the other hand, edges on G are not grouped together. In addition to graphs, the edge cuts on G and $G^{(q,\texttt{S})}$ are mapped as follows: the edge-cut value of a group in $G^{(q,\texttt{S})}$ is set to the maximum edge-cut value in the equivalent group in G.

Figures 4.11a and 4.11b are constructed from a 3SAT formula for the reactive optimization problem, where a truth assignment for the variables can be found by solving **MILP-REACTIVE**.

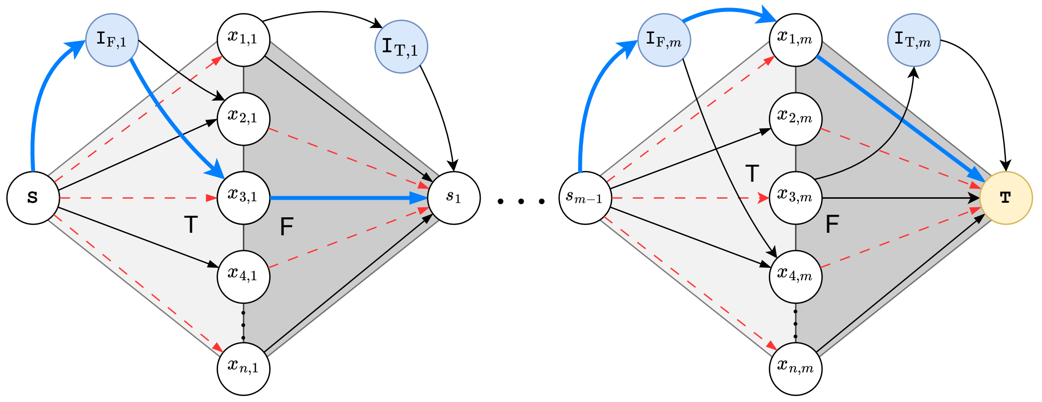
The following theorem and proof is taken from [107].

Theorem 4.7. Problem 4.6 is NP-complete and Problem 4.5 is NP-hard.

Proof. The proof follows similarly from Theorem 4.5. In this setting, a witness for Problem 4.6 comprises the maximum edge cut value of each group in G. Construction 3 relates edge cuts on G and $G^{(q,\texttt{S})}$. This implies that edge cuts on G are found under the condition that there is a strictly positive flow on $G^{(q,\texttt{S})}$ under a static mapping of edges. The minimum set of edge cuts which ensures no bypass paths on G has cardinality n, corresponding to only one of the sub-graphs having edge cuts. Furthermore, for each x_i, there will be one edge-cut in one of the two groups (incoming or outgoing edges). Therefore, for each x_i, only the incoming or the outgoing edge group will have a maximum edge cut value of 1, corresponding to the Boolean assignment for x_i. A minimum cut on G found under the conditions of no bypass paths on G and a positive flow on $G^{(q,\texttt{S})}$ results in a Boolean assignment



(a) Constructed graph G for an arbitrary 3SAT formula $f(x)$.



(b) Statically mapped cuts on G for every subgraph.

Figure 4.11: (a) Graph G and (b) graph G_{sys} constructed according to Construction 3.

that is a satisfying witness to the 3-SAT formula. Thus, we have polynomial-time construction of a satisfying witness to the 3-SAT formula from a satisfying witness to Problem 4.6. This follows similarly to Theorem 4.5.

Likewise, a satisfying witness to the 3-SAT formula can be mapped to edge cuts on one of the sub-graphs of G. These edge cuts will be such that there is no bypass path on G, and will be the minimum set of edge cuts to accomplish this task, corresponding to $|C| = n$. Additionally, by construction of the graphs, this will correspond to a strictly positive flow on $G^{(q,\mathtt{S})}$. Thus, we can construct a satisfying witness to Problem 4.6 in polynomial time from a satisfying witness of the 3-SAT formula. Therefore, any 3-SAT problem reduces to an instance of Problem 4.6. As a result, Problem 4.6 is NP-complete and following similarly to Corollary (4.1), Problem 4.5 is NP-hard. $\square$

4.10 Comparison to Reactive Synthesis

We presented an approach to solve Problems 4.1 and 4.2 leveraging tools from automata theory and network flow optimization. In particular, for Problem 4.2, we rely on the optimization solution to construct a GR(1) specification to reactively synthesize a test agent strategy. One indication of the optimization step being necessary is the computational complexity of the problem. If the problem data are consistent, there exists a GR(1) specification for the test agent that would solve the problem, but directly expressing this specification is impractical. Essentially, the challenge is in finding the restrictions on system actions, which are then captured in the sub-formulas of the GR(1) specification. In this section, we argue that we cannot solve Problems 4.1 and 4.2 solely via synthesis from an LTL specification.

To the authors' knowledge, directly capturing the different perspectives of the system and the tester in this neither fully adversarial nor fully cooperative setting is not possible with current state-of-the-art approaches in GR(1) synthesis. Particularly in the reactive setting, the test strategy must ensure that from the system's perspective, there always exists a path to the system goal. To capture this constraint, we reason over a second product graph that represents the system perspective. It is not obvious how this semi-cooperative setting can be directly encoded as a synthesis problem in common temporal logics.

In the static setting, the problem can be posed on a single graph. However, it is difficult to find the set of static obstacles directly from GR(1) synthesis. Every state in the winning set describes an edge-cut combination, but qualitative GR(1) synthesis cannot maximize the flow or minimize the cuts. Furthermore, the winning set can include states that vacuously satisfy the formula, i.e., not allowing the system any path to the goal. Finally, the combinatorial complexity of the problem would manifest as follows. Although the time complexity of GR(1) synthesis is $O(N^3)$ in the number of states N, we require an exponential number of states to characterize the GR(1) formula. For example, in Figure 4.12, this is illustrated for the GR(1) formula:

$$\square\varphi_{\text{sys}}^{\text{dyn}} \wedge \square\lozenge\,\mathrm{T} \rightarrow \square\varphi_{\text{test}}^{\text{dyn}} \wedge \square\varphi_{\text{test}}^{\text{aux\_dyn}} \wedge \square\lozenge\, I_{\text{aux}},$$

where $\varphi_{\text{sys}}^{\text{dyn}}$ captures the system transitions on the gridworld, $\varphi_{\text{test}}^{\text{dyn}}$ are the dynamics of the test environment, and $\varphi_{\text{test}}^{\text{aux\_dyn}}$ and I_{aux} capture the $\lozenge I$ condition in GR(1) form. In this example, each edge in the system transition system T can take 0/1 values, and once an edge is cut, it remains cut and the system cannot take a transition that corresponds to a cut edge. Due to this, the number of states N to describe

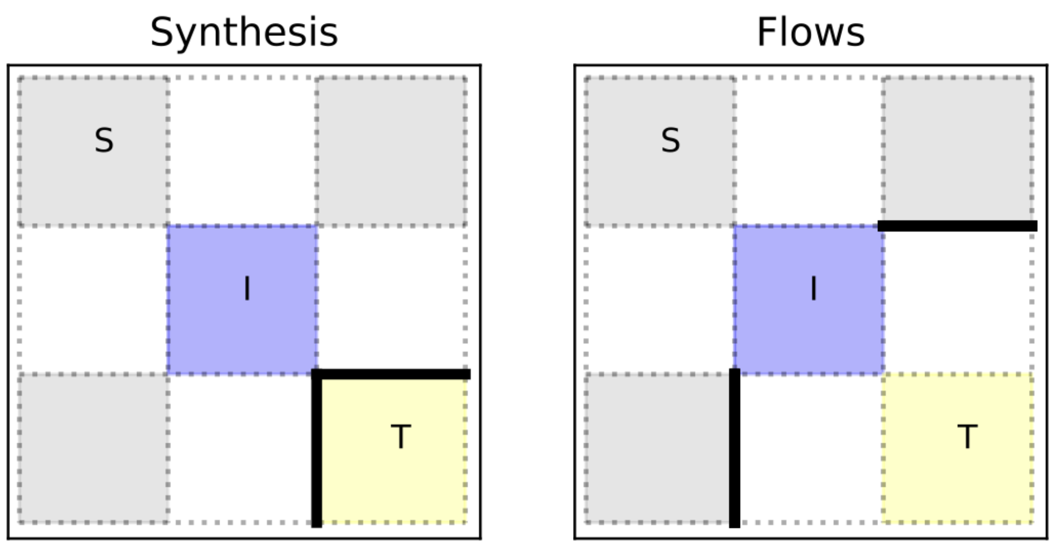


Figure 4.12: Solution returned by GR(1) synthesis and the network flow optimization in the case of static constraints

the GR(1) formula includes the $2^{|T.E|}$ states that characterize the edge cuts. As seen in Figure 4.12, the direct GR(1) synthesis approach returns a trivial solution corresponding to an impossible setting for the system. Finally, even when an acceptable solution is returned, the problem being at least NP-hard will result in the combinatorial complexity manifesting in the synthesis approach.

One key advantage of the network flow optimization is reasoning over flows as opposed to paths, which allows for tractable implementations. These insights from network flow optimization in this work can help in driving further research along these directions.

4.11 Experiments

We illustrate the synthesized test strategy in simulation and hardware with Unitree A1 quadrupeds. These experiments show that the high-level abstraction models is useful in high-level test synthesis as long as the lower levels of the system are implemented to simulate the high-level abstraction. The low-level control of the quadruped is managed at the motion primitive layer, which abstracts the underlying dynamics and facilitates transitions between primitives as described in [113]. These primitives include behaviors such as lying down, walking at various speeds, jumping, standing, and reduced-order model-based tracking of waypoints that rely on a unicycle or single integrator model. These motion primitives can directly be commanded from a high-level controller implemented by a temporal logic planning

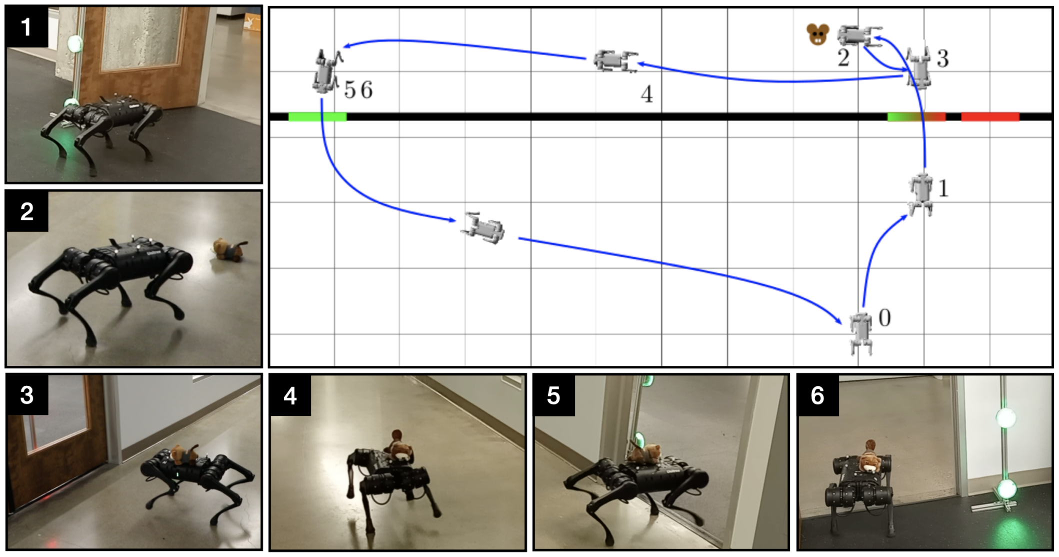


Figure 4.13: Beaver Rescue Hardware Experiment with door$_1$ on the right and door$_2$ on the left.

toolbox such as TuLiP [109]. Each motion primitive is implemented in C++, with control laws, sensing, and estimation executed at 1kHz.

For experiments demonstrating the flow-based synthesis in the MILP formulation, examples with static test environments solve the routing optimization **MILP-STATIC**, examples with reactive test environments solve **MILP-REACTIVE**, and those with reactive dynamic agents solve **MILP-AGENT**, unless otherwise stated. These optimizations are solved using Gurobipy [114]. The reactive test agent strategies are synthesized using the temporal logic planning toolbox TuLiP [109].

Hardware Experiments for Tests Synthesized from solving the Min-Max Stackelberg Game

We will see two hardware experiments — beaver rescue and motion primitive examples — for the flow-based synthesis formulated via min-max Stackelberg games. These optimizations were implemented in Pyomo [104], which interfaces to Gurobipy. These examples will be revisited in simulation and solved using the MILP framework.

Beaver Rescue Example: This example is inspired by a search and rescue mission and the hardware trace is shown in Fig. 4.13. In this example, the quadruped (system under test) is tasked with picking up a beaver (located in the corridor), and returning to lab safely: $\varphi_{\text{sys}} = \Diamond\, goal$, where $goal$ is satisfied when the beaver is brought into the lab. The lab has two doors which the quadruped can use to navigate into the corridor. In our implementation of the discretized abstraction of this experiment, the transitions of the quadruped are as follows,

$$\begin{aligned} T_{\text{sys}}.E =& \{(s_0, d_1), (s_0, d_2), (d_1, d_2), (d_2, d_1), (d_1, b), (d_2, b), (b, p_1), \\ & (b, p_2), (p_1, p_2), (p_2, p_1), (p_1, g), (p_2, g)\}, \end{aligned} \tag{4.43}$$

where i) states d_1 and d_2 are states in the lab adjacent to doors: $T_{\text{sys}}.L(d_1) = \{\text{door}_1\}$ and $T_{\text{sys}}.L(d_2) = \{\text{door}_2\}$, ii) states p_1 and p_2 are states in the *corridor* adjacent to doors: $T_{\text{sys}}.L(p_1) = \{\text{door}_1\}$ and $T_{\text{sys}}.L(p_2) = \{\text{door}_2\}$, iii) state b in the hallway is the rescue location of the beaver, and iv) states s_0 and g represent the lab. The test objective is to route the system to visit both doors: $\varphi_{\text{test}} = \Diamond\, \text{door}_1 \wedge \Diamond\, \text{door}_2$.

There are several ways in which the test environment could have routed the system. If the system visits door$_1$ from d_1 (or likewise door$_2$ from d_2), the door could then be blocked; forcing the system to re-plan to exit into the corridor through the other door. Alternatively, the system could visit door$_1$ from d_1 (or likewise door$_2$ from d_2) and exit into the corridor, and on its return with the beaver, door$_1$ from p_1 (or likewise door$_2$ from p_2) can be blocked while leaving the other door at p_2 (or likewise at p_1) open for the quadruped to re-enter the lab. Our algorithm found edge-cuts that resulted in the latter test case that allows the system to exit through the door of its choice and blocks that door on the return path. The synthesized test is reactive to the choice of system actions — depending on the door approached by the system, the synthesized constraints are placed accordingly.

Motion Primitive Example: In this example, the quadruped is can execute the following motion primitives: “jump”, “stand”, “lie”, and “walk”. Once again consider the lab-corridor setup. The quadruped’s goal is to reach the beaver in the corridor: $\varphi_{\text{sys}} = \Diamond\, \text{goal}$. The test objective is $\varphi_{\text{test}} = \Diamond\, \text{jump} \wedge \Diamond\, \text{lie} \wedge \Diamond\, \text{stand}$ in order to test the system to demonstrate the three motion primitives.

Unlike the previous example in which doors were closed to restrict the system, in this example each door has three lights located at different heights to signal motion primitives that might unlock the door. There are three such doors, and the states p_i (for $i = 1, 2, 3$) represents the state of the quadruped standing in front of door$_i$ before demonstrating a motion primitive. The state prim_i for motion primitive $\text{prim} \in \{\text{lie}, \text{stand}, \text{jump}\}$ represents the abstract state of the quadruped performing the motion primitive in front of door$_i$. After it performs a motion primitive, the quadruped state transitions to $d_{i,\text{prim}}$, from which it can proceed to the goal or be

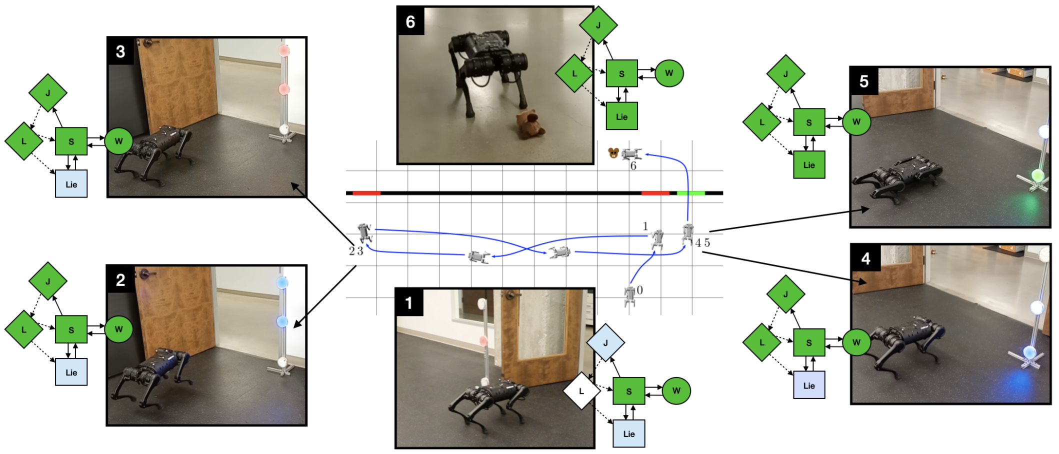


Figure 4.14: Motion Primitive example: Snapshots of the hardware test execution on the Unitree A1 quadruped.

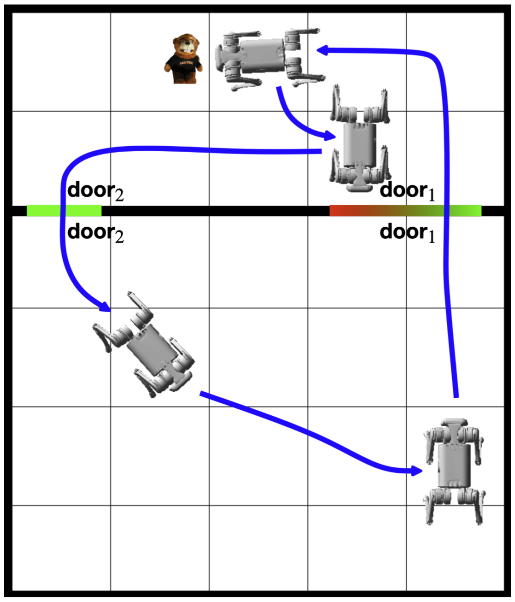
returned to the state p_i. The test harness comprises of system actions corresponding to the following transitions: $\{(d_{i,\text{prim}}, goal)\}$.

For example, if the middle light is blue, it implies that demonstrating the stand motion primitive could unlock the door (by the light turning green). The test strategy is reactive to the system; depending on the order in which the quadruped approaches the doors and demonstrates motion primitives, the lights turn red/green to restrict/permit the system to pass.

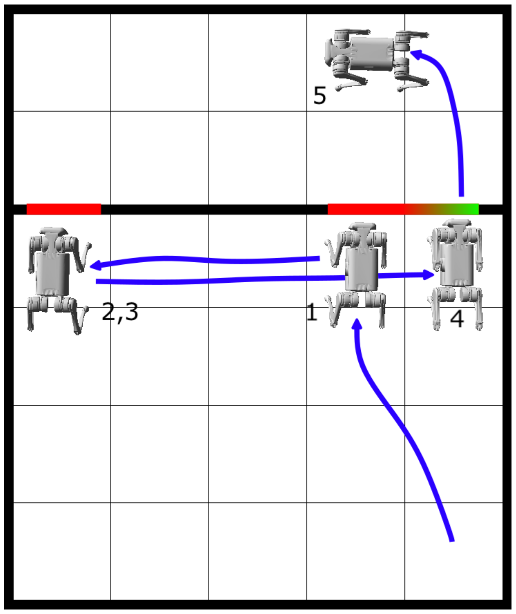
In this test execution, the system chooses to approach the middle door (door$_2$) first which can only be unlocked by the stand motion primitive. The quadruped successfully demonstrates this (panel 1 of Fig. 4.14), but the light turns red. Following this, the quadruped approaches (door$_1$) and demonstrates the jump and stand motion primitives, but is still not permitted to pass (panels 2 and 3 of Fig. 4.14). Finally, after approaching door$_3$ on the right, the system demonstrates the lie motion primitive, after which the corresponding light turns green (panels 4 and 5 of Fig. 4.14), and the quadruped finally navigates to the corridor. In this manner, the test strategy reacts to system behavior and routes the test execution to lead the system to demonstrate all three motion primitives before being allowed to pass.

Simulated Experiments

First, we will revisit the beaver rescue and motion primitive examples, in which test strategies will be implemented by the test environment using reactive obstacles For the beaver rescue example, the test harness consists of doors that connect the lab and corridor, and system transitions can be restricted by closing the doors. For the



(a) Beaver rescue.



(b) Motion primitive example.

Figure 4.15: Simulated experiment results with test strategy found by solving **MILP-REACTIVE**. In (b), system (gray) demonstrates primitives in the order: stand (1), stand (2), jump (3), and lie (4), before advancing to goal (5).

motion primitive example, the test harness consists of restricting transitions after motion primitives have been demonstrated. Figure 4.15 shows the simulated experiments for these examples where the test strategy was found by solving **MILP-REACTIVE**. The simulated test executions are qualitatively similar to the hardware demos discussed previously, even with the new MILP formulation. As shown in Table 4.4, the graph size $|G|$ for these examples is relatively small compared to other examples, with the exception of the running examples 4.2 and 4.1. Despite this, the MILP approach is faster by three orders of magnitude. Both the game formulation and the MILP formulation aim to solve the problem exactly. However, defining a single set of flows and directly solving the problem as an MILP makes the problem much more tractable. A large part of this can be attributed to Gurobi, and the algorithms for solving min-max stackelberg games with coupled constraints have not been optimized for efficiency as much in comparison to solving MILPs.

Maze 1: This example consists of a system quadruped (gray) navigating on the grid shown in Fig. 4.17a, and a dynamic test agent (yellow) that can traverse the middle column. The test agent is restricted to only walk up or stay in a cell. From the middle cell of the top row, the test agent can navigate off the grid into a parking state. The system objective is to reach the goal on the top-left corner of the grid,

$\varphi_{\text{sys}} = \Diamond\, goal$, and the test objective is to route the system through intermediate states I_1, I_2, and I_3: $\varphi_{\text{test}} = \Diamond\, I_1 \wedge \Diamond\, I_2 \wedge \Diamond\, I_3$.

Figures 4.16a– 4.16c visualize a counterexample that is not dynamically realizable by the test agent. This solution is added as a counterexample, and the MILP is resolved until a realizable solution (see Figures 4.16d– 4.16f) is found.

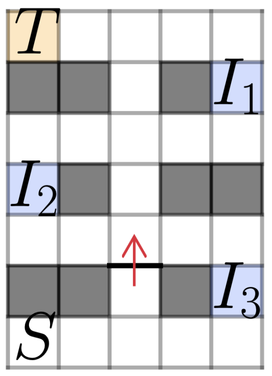
Brief explanation for counterexample: In Figure 4.16a, an agent would have to occupy cell $(4, 2)$ when the system occupies the cell $(5, 2)$. This would result in a livelock — from the system perspective, there is no incentive to back up and navigate around through $\mathtt{I}_3$ since the test agent would block all paths to the goal, and if the test agent moves out of cell $(4, 2)$, the system can navigate to the goal without being routed through $\mathtt{I}_3$.

The test quadruped first begins in the lower-most row and moves out of the way but still blocking the path through the center column so that the system is routed through I_3. Once the system visits I_3, the test agent walks up to the middle cell in the grid to block it so it is routed through I_2. Similarly, the test agent routes the system through I_1. After the system visits I_1 but before it reaches the center cell in the first row, the test agent walks off the grid, and into its parking state. This is due to the temporal logic constraint to not over-restrict the system (equation (g7)). When any cell occupied by the test agent (say v) is adjacent to the system (say occupying cell u), then the transition (u, v) is registered as a restriction on the system. To avoid over restricting the system, the test agent navigates of the grid.

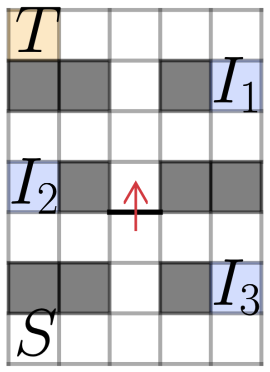
Hardware Experiments

Running Example 4.1: The experiment trace for the medium example is given in Fig. 4.17c. The corresponding solution is shown in Fig. 4.7.

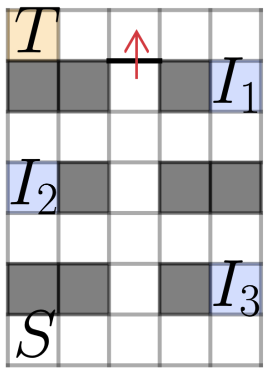
Refueling Example: In this example, the system quadruped (gray) navigates on the grid shown in Figure 4.18a. In addition to coordinates $\mathbf{x} = (x, y)$, the system state also includes a discretized fuel state $fuel$. The maximum value of $fuel$ is 10, and every cell transition on the grid decreases this value by 1. Visiting the refueling station in the bottom-right corner of the grid resets $fuel$ to its maximum value. The desired test behavior is to place the system in a state in which its fuel level is not sufficient for it to directly navigate to the goal. The system objective is given as $\varphi_{\text{sys}} = \Diamond T \wedge \Box\neg(fuel = 0)$. The test objective is set to $\varphi_{\text{test}} = \Diamond(y < 4 \wedge fuel < 2)$, which seeks to place the system in the lower 3 cells of the grid with less than two units of fuel. The sub-tasks used in describing these objectives are safety and



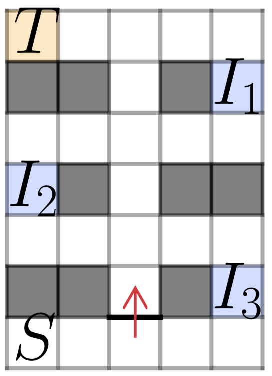
(a) q0



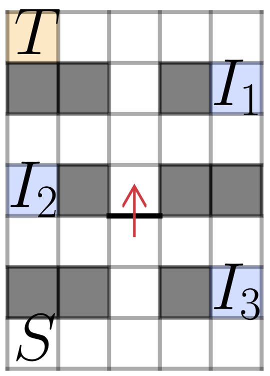
(b) q15



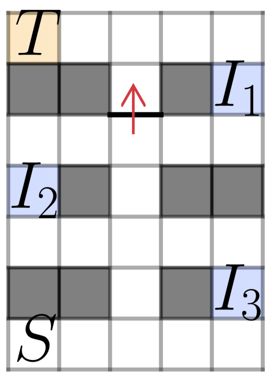
(c) q12



(d) q0

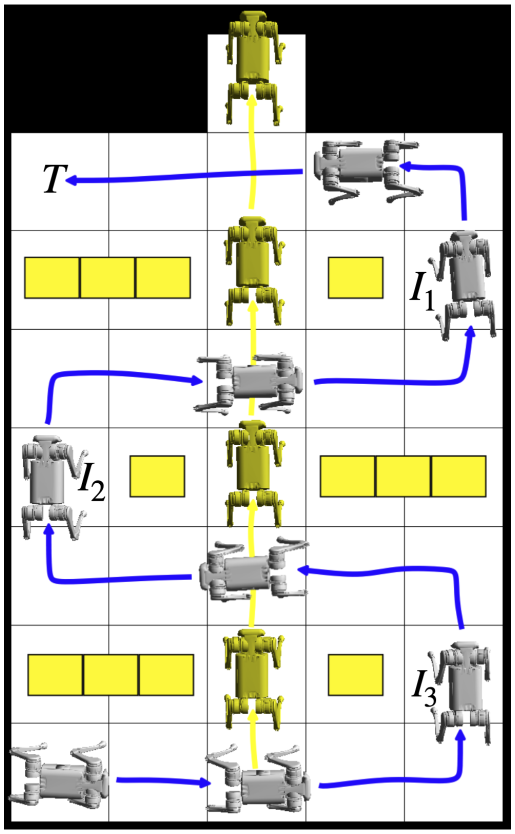


(e) q15

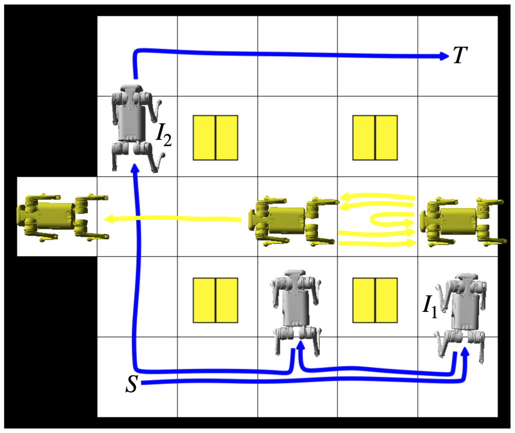


(f) q12

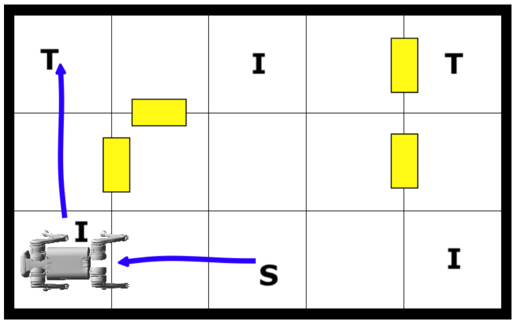
Figure 4.16: Illustration of dynamically unrealizable (top (a)–(c)) and dynamically realizable reactive obstacles (bottom (d)–(f)). In Figures 4.16a– 4.16c: Reactive obstacles returned by **MILP-REACTIVE** that cannot be realized by a dynamic test agent. In Figures 4.16d– 4.16f: Accepted solution for which a test agent strategy is synthesized. Red arrow indicates the direction of the restriction; the edge-cuts found by **MILP-REACTIVE** are not subject to the (optional) bidirectional cut constraint. History variable q0 refers to the state of the test execution before I_3 is visited by the system, q15 is the state of the test execution after only I_3 is visited, and q12 is the state of the test execution after I_3 and I_2 have been visited.



(a) Simulated experiment for Maze 1.



(b) Simulated alternative trace, Maze 2.

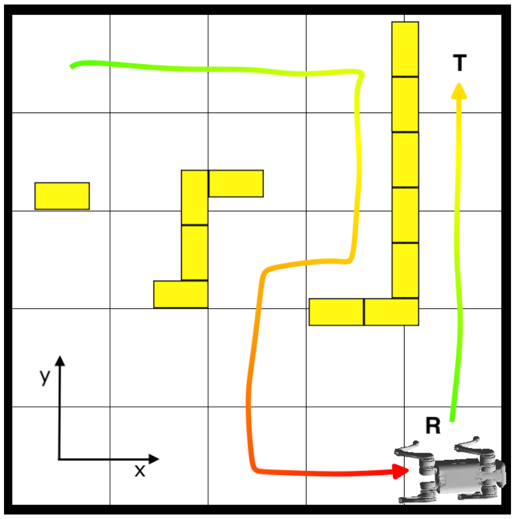


(c) Hardware trace for the medium example 4.1 with static obstacles found by test strategy.

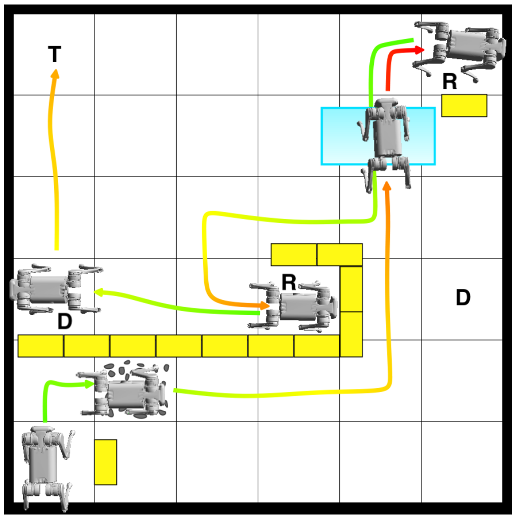
Figure 4.17: Yellow boxes in (a) and (b) are pre-defined obstacles to indicate states that are not navigable in T_{sys}. Yellow obstacles in (c) are static obstacles placed by the test environment. Gray quadruped is the system, and yellow quadruped in (a) and (b) is the test agent, which chooses to navigate off-grid after the test objective is realized.

reachability. Note that the intermediate states resulting from this test objective also include states with $fuel = 0$, but the restrictions from the MILP will not force the system into these *unsafe* states, giving the system the option to have a fuel level of 1 and refuel. This still satisfies the test objective without making it impossible for the system to satisfy the test objective.

The experiment trace of the test execution in shown in Fig. 4.18a, in which the color of the trace indicates the comparative fuel level at that state. The yellow boxes represent static obstacles placed to restrict transitions according to the solution of **MILP-STATIC**. As given in Tables 4.4 and 4.5, the product graph has over 1000 edges resulting in around 1000 binary variables for the routing optimization. The optimization is solved to optimality in 0.87 seconds, and the maximum flow is found



(a) Refueling example experiment trace.



(b) Mars exploration experiment trace.

Figure 4.18: Traces of hardware demos with test environment consisting of static obstacles.

to be 2. The 199 cuts on G correspond to the 8 transitions restricted (bidirectionally) on the transition system. This example illustrates the usefulness of our framework — test objectives are not limited to being defined over atomic propositions of the pose $\mathbf{x}$ of the system. The solution to this specific example is not one that can be easily identified like the previous examples we have discussed thus far.

Mars Exploration Example: This example is inspired by a sample collection mission on Mars. The sub-tasks reachability, avoidance, and delayed reaction are used to characterize system and test objectives. The system quadruped (gray) can traverse the grid shown in Figure 4.20a, which has states with "rock" and "ice" samples, and states designated as sample drop-off locations D, and refueling stations denoted R. The system is required to reach the goal in the top-level corner (labeled T), and must drop-off any samples collected during its navigation without running out of fuel. The system state carries a fuel level $fuel$ in addition to its pose state $\mathbf{x} = (x, y)$. Similar to the refueling example, the maximum fuel value is 10, it decreases by 1 for every transition on the grid, and it can be refueled by visiting the refueling states R.

The system objective is given by the formula:

$$\varphi_{\text{sys}} = \Diamond T \wedge \Box\neg(f = 0) \wedge \Box(\text{ice} \vee \text{rock} \rightarrow \Diamond\,\text{drop-off}).$$

The test objective consists of reachability sub-tasks that include triggers of the reaction sub-task of the system objective, and also a sub-task to place the system in a

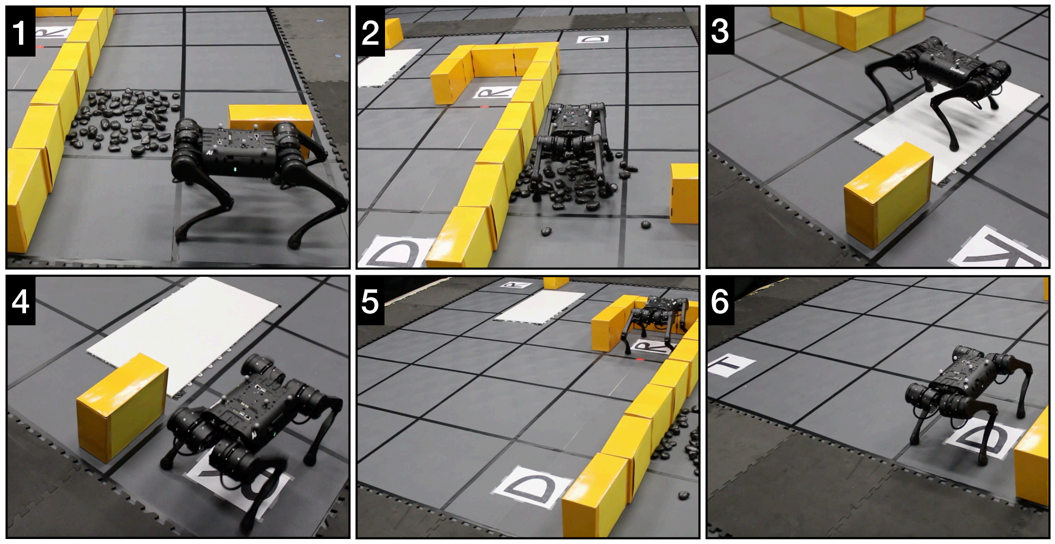


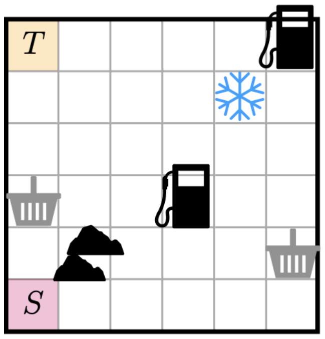
Figure 4.19: Mars exploration experiment snapshots from resulting on the Unitree A1 quadruped for static test environments. The overview is shown in Fig. 4.18b.

low-fuel state:

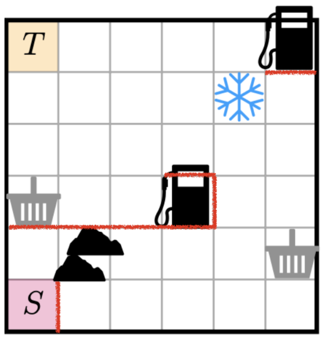
$$\varphi_{\text{test}} = \Diamond \, \text{rock} \wedge \Diamond \, \text{ice} \wedge \Diamond (d > f),$$

where $d = |\mathbf{x} - \mathbf{x}_{\text{goal}}|$ is the distance to the goal and f is the fuel level. Figure 4.20b shows a sub-optimal placement of static obstacles with maximum flow $F = 1$, and Figure 4.20c shows the optimal placement permitting a maximum flow of $F = 2$ on the product graph. The experiment trace (Figure 4.18b) and accompanying demo (Figures 4.19) are test executions in the test environment realizing the sub-optimal test strategy. The system begins in the bottom-left corner of the grid with a full fuel tank. From these figures, we can observe the quadruped being routed to pick up the rock sample close to the initial condition. Then, the placement of static obstacles in both the sub-optimal and the optimal settings is such that the system needs to visit the top-right refueling station at least once. In order to visit that refueling station without running out of fuel, the system must navigate the state with ice samples. In the test execution from the experiment, the system is routed to visit states with rock and ice samples, after which it refuels twice — first at the top-right refueling station and then at the refueling station at the center of the grid — and finally, drops off the samples before navigating to goal T. Table 4.5 lists this example as one of the largest with around $13,000$ integer variables. Despite this, the routing optimization is solved optimally in about 45 seconds.

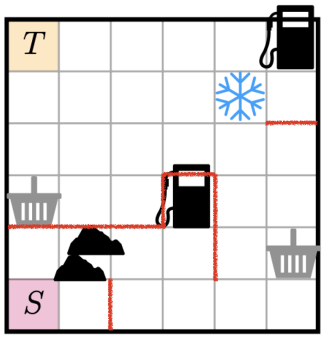
Patrolling Quadruped: This examples involves a dynamic test agent whose strategy is synthesized to be consistent with the solution of the routing optimization in



(a) Empty grid for Mars exploration example. Refueling stations, rock and ice sample locations are denoted. Drop-off locations are denoted by the basket.



(b) Static obstacles for Mars exploration example with a feasible solution of **MILP-STATIC** (max-flow F=1).

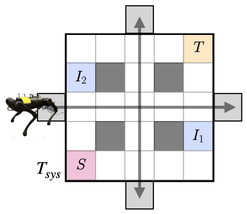


(c) Static obstacles for Mars exploration example with an optimal solution of **MILP-STATIC** (max-flow $F = 2$).

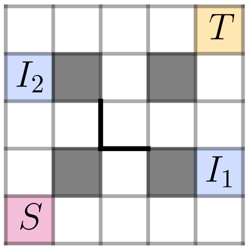
Figure 4.20: Feasible and optimal solutions for the Mars exploration example. The hardware experiment corresponded to the feasible solution.

MILP-AGENT. The context of this example is similar to the refueling example except that the test environment can now consist of static obstacles and a dynamic test agent (see Figure 4.1). The system (gray quadruped) is tasked with beginning in the lower right corner of the grid, and reaching the target cell in the lower-left corner without running out of fuel. Additionally, the system must not collide with obstacles. The test objective is to put the system in a low-fuel state similar to the Mars exploration example. The test agent dynamics allow it to traverse up and down the center column of the grid, and from the center cells of the top and bottom rows, it can choose to move off the grid into a parking state. Thus, the system and test objectives are given by the formulas: $\varphi_{\text{sys}} = \diamondsuit T \wedge \square \neg (fuel = 0)$, and the test objective is $\varphi_{\text{test}} = \diamondsuit (d > f)$, where $d = |\mathbf{x} - \mathbf{x}_{\text{goal}}|$ is the distance to the goal.

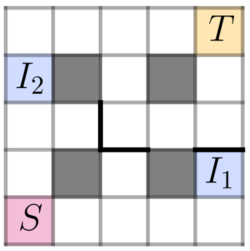
As seen in Figure 4.1, the test environment places a static obstacle near the initial state of the system (panel 1 snapshot). Then, as the system proceeds to go to the goal, the test agent blocks the quadruped from crossing the center column of the grid — in panels 1 and 2, the test agent blocks the system in the lowermost row, and when the system advances up in panels 3 and 4, the test agent continues to block the system. The test agent blocks the system until the it can no longer directly navigate to the goal, and must refuel. Thus, the system refuels and is then able to navigate to T without any further interactions with the test agent. Some implementation details are as follows. the system controller in this test execution resynthesizes its strategy each time it is restricted by the test agent. Furthermore, the optimization **MILP-**



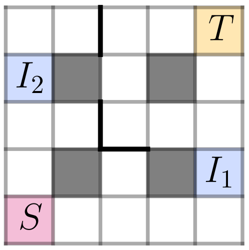
(a) Grid world layout.



(b) Reactive cuts in q_0.

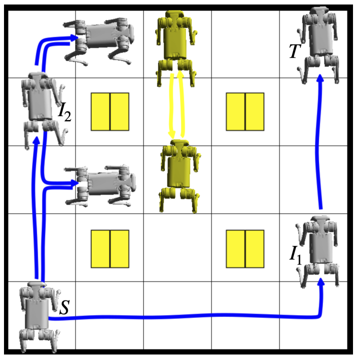


(c) Reactive cuts in q_6.

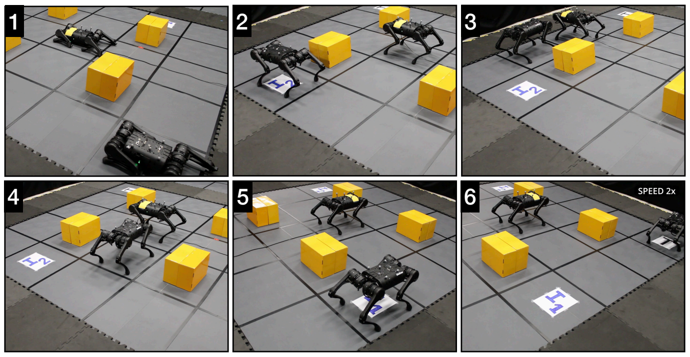


(d) Reactive cuts in q_7.

Figure 4.21: Grid world layout and reactive cuts corresponding to the history variables for the Maze 2 experiment. (a) Grid world layout with cells traversible by the test agent marked. Dark gray cells are not traversible by either agent. (b)–(d) Black edges indicate reactive cuts corresponding to the history variables for the Maze 2 experiment. Note that the cuts are not bidirectional. The history variable states q0, q6, and q7 can be inferred from $\mathcal{B}_\pi$ illustrated in Fig. 4.2c, and correspond to initial state, visiting I_1 first, and visiting I_2 first.



(a) Maze 2 trace.



(b) Maze 2 experiment snapshots.

Figure 4.22: Resulting test execution for the Maze 2 experiment with a dynamic test agent.

AGENT with the modified objective is solved to minimize the occupied states.

Maze 2: The grid world layout for this example is shown in Figure 4.21a, in which the gray boxes denote states that the system cannot navigate to. The system is tasked with navigating to goal T from state S: $\varphi_{\text{sys}} = \Diamond T$. The test objective is to route the system to visit states I_1 and I_2: $\varphi_{\text{test}} = \Diamond I_1 \wedge \Diamond I_2$. The test environment has access to a dynamic test agent that can traverse the center column and row of the grid as illustrated in yellow lines in Figure 4.21a. In addition, the test agent can walk off the grid into a parking state from the four cells at the boundaries of the grid (top and bottom cell of center column, and left-most and right-most cells of the center row). While the test environment can also place static obstacles, it chooses to restrict the system using just the test agent.

The specification product is exactly the same as the running example 4.2, and is illustrated in Figure 4.2c. Observe that to route the test execution through the test objective acceptance states, we need to find cuts for the history variables q0 (initial state), q6 (I_1 has been visited but not I_2), and q7 (I_2 has been visited but not I_1). The reactive cuts found by the flow-based synthesis procedure are shown in Figs. 4.21b-4.21d. The trace and snapshots of the resulting test execution is shown in Figs. 4.22a and 4.22b. We observe that the system quadruped decides to take the top path first, visits I_2 (see panel 2 in Fig. 4.22b), and is blocked by the test agent (see panel 3). It then decides to try navigating through the center of the grid, and is again blocked by the test agent (see panel 4). Subsequently, it decides to try the bottom path, visits I_1 (see panel 5), and successfully reaches the goal without any further test agent intervention. If the system decided to visit I_1 first, the adaptive test agent strategy would have blocked the system from reaching the goal directly from I_1 until it visits I_2. This is an example with a maximum flow of $F = 2$, corresponding to the two unique ways for the system to reach the goal. For an alternative system controller in which the system chooses to approach the goal through I_1, the simulated trace resulting from the test agent strategy is shown in Fig. 4.17b.

Runtimes

Tables 4.5 and 4.6 list the optimization size and runtimes for all the simulated and hardware experiments discussed prior. Table 4.5 corresponds to experiments that identify static and/or reactive obstacles, and Table 4.6 corresponds to experiments in which a test agent strategy is synthesized from the output of the routing optimization. The problem size (e.g., automaton size, graph size) and graph construction times for all experiments are given in Table 4.4. The sizes of automata, transition systems, and product graphs are listed by the tuple $(|V|, |E|)$.

To further study the scalability of the routing optimization, I tabulate the runtimes for randomized gridworld experiments for various specification sub-tasks in Tables 4.7 and 4.8. These computations were conducted on an Apple M2 Pro with 16 GB of RAM using Gurobipy [114]. The construction of DBAs from specifications was implemented using Spot [98]. In the randomized experiments, the Gurobi solver for the MILP has a timeout condition set at 10 minutes to find at least a feasible solution. Once the solver finds a feasible solution, it is given another minute to return a solution with the optimality guarantee. If the solver cannot guarantee opti-

Table 4.4: Automata and graph construction runtimes for simulated and hardware experiments

Experiment	$\lvert\mathcal{B}_\pi\rvert$	$\lvert T\rvert$	$\lvert G\rvert$	G[s]
Example 4.1	(4, 9)	(15, 53)	(27, 96)	0.0270
Refueling	(6, 18)	(265, 1047)	(332, 1346)	0.6655
Mars Exploration	(36, 354)	(376, 1522)	(4073, 17251)	75.8313
Example 4.2	(8, 27)	(6, 17)	(20, 56)	0.0452
Beaver Rescue	(12, 54)	(7, 19)	(15, 39)	0.0470
Motion Primitives	(16, 81)	(15, 42)	(72, 207)	0.4286
Maze 1	(16, 81)	(26, 80)	(196, 604)	1.6226
Patrolling	(6, 18)	(386, 1539)	(210, 831)	0.4573
Maze 2	(8, 27)	(21, 66)	(80, 252)	0.2195

Table 4.5: Routing optimization runtimes for simulated and hardware experiments with static and/or reactive obstacles

Experiment	|BinVars|	|ContVars|	|Constraints|	Opt[s]	Flow	|cuts|
Solving **MILP-STATIC**						
Example 4.1	73	87	540	0.0003	3.0	14
Refueling	1014	1261	19819	0.8682	2.0	199
Mars Exploration	13178	16604	1646480	46.6209	2.0	1641
Solving **MILP-REACTIVE**						
Example 4.2	25	115	409	0.0003	2.0	4
Beaver Rescue	8	154	441	0.0001	2	2
Motion Primitives	106	761	2606	0.0005	3.0	15

Table 4.6: Runtimes for simulated and hardware experiments with dynamic agents

Experiment	BinVars	Opt[s]	Controller[s]	$\lvert \mathsf{C}_{ex}\rvert$	Flow	|cuts|
Solving **MILP-AGENT**						
Maze 1	355	0.0010	100.0	4	1.0	3
Patrolling	621	6.0535	16.1191	0	1.0	13
Maze 2	176	0.0292	7.151	8	2.0	8

Table 4.7: Run times (with mean and standard deviation) for random grid world experiments solving **MILP-REACTIVE**

Experiment		5 × 5		10 × 10		15 × 15		20 × 20	
$\lvert AP\rvert$	$\lvert\mathcal{B}_\pi\rvert$	Optimization[s], Success Rate (%)							
Reachability:									
2	(4, 9)	5.63 ± 13.43	100	64.62 ± 38.75	100	67.38 ± 25.47	100	68.63 ± 31.12	100
3	(8, 27)	23.36 ± 38.15	100	61.68 ± 35.12	100	91.54 ± 31.41	100	117.82 ± 34.89	100
4	(16, 81)	22.49 ± 36.33	100	83.52 ± 29.25	100	171.49 ± 50.72	100	317.62 ± 89.08	100
Reachability & Reaction:									
3	(6, 21)	5.97 ± 13.21	100	61.06 ± 34.67	100	71.64 ± 41.03	100	85.20 ± 19.49	100
5	(20, 155)	17.19 ± 25.51	100	78.44 ± 34.71	100	159.91 ± 76.63	100	279.86 ± 148.23	90
7	(68, 1065)	52.71 ± 41.23	100	331.32 ± 187.28	90	585.21 ± 67.58	15	$\mathbf{600.00 \pm 0.00}$	**0**
Reachability & Safety:									
3	(6, 18)	0.76 ± 1.52	100	70.82 ± 89.70	100	63.68 ± 27.54	100	80.58 ± 20.79	100
4	(6, 18)	0.15 ± 0.29	100	71.47 ± 80.61	100	59.59 ± 38.92	100	76.02 ± 27.11	100
5	(6, 18)	0.12 ± 0.18	100	94.68 ± 88.04	100	71.34 ± 30.89	100	82.54 ± 22.69	100

Table 4.8: Run times (with mean and standard deviation) for random grid world experiments solving **MILP-STATIC**.

Experiment		5 × 5		10 × 10		15 × 15		20 × 20	
$\lvert AP\rvert$	$\lvert\mathcal{B}_\pi\rvert$	Optimization [s], Success Rate (%)							
Reachability:									
2	(4, 9)	8.17 ± 13.14	100	54.07 ± 17.98	100	60.17 ± 0.12	100	60.17 ± 0.10	100
3	(8, 27)	27.78 ± 21.71	100	60.17 ± 0.10	100	60.48 ± 0.86	100	74.02 ± 38.70	100
4	(16, 81)	52.60 ± 14.05	100	60.42 ± 0.34	100	82.02 ± 41.26	100	265.41 ± 203.51	80
Reachability & Reaction:									
3	(6, 21)	10.62 ± 14.85	100	60.09 ± 0.06	100	60.23 ± 0.24	100	60.34 ± 0.46	100
5	(20, 155)	20.41 ± 19.21	100	67.77 ± 31.90	100	95.31 ± 116.65	95	268.50 ± 222.14	75
7	(68, 1065)	36.64 ± 23.34	100	110.63 ± 92.81	100	419.77 ± 214.30	55	$\mathbf{556.38 \pm 131.06}$	**10**
Reachability & Safety:									
3	(6, 18)	1.27 ± 1.47	100	60.08 ± 0.06	100	57.27 ± 12.61	100	60.32 ± 0.24	100
4	(6, 18)	0.17 ± 0.23	100	60.06 ± 0.05	100	60.14 ± 0.10	100	60.30 ± 0.19	100
5	(6, 18)	0.11 ± 0.16	100	54.15 ± 17.80	100	60.17 ± 0.09	100	60.29 ± 0.26	100

mality in that time frame, the feasible solution is returned. If the optimizer returns at least a feasible solution, the run is counted as a success. An empirical observation is that Gurobi often finds an optimal solution but takes even longer to produce an optimality guarantee. For the randomized experiments, gridworlds from size 5×5 to 20×20 are considered, and for each gridsize, problem instances are randomly generated. In the allotted time, if the optimization returns that a problem instance is infeasible, then the instance is discarded and a new one is generated in its place.

Tables 4.7 and 4.8 tabulate the optimization runtimes and success rate for solving **MILP-REACTIVE** and **MILP-STATIC**, respectively. The optimization runtime lists the mean and standard deviation for 20 instances. The success rate indicates

Table 4.9: Graph construction runtimes (with mean and standard deviation) for random grid world experiments

Experiment		5×5	10×10	15×15	20×20
$\lvert AP\rvert$	$\lvert\mathcal{B}_\pi\rvert$	Graph Construction [s]			
Reachability:					
2	(4, 9)	0.046 ± 0.001	0.224 ± 0.0056	0.554 ± 0.009	1.078 ± 0.011
3	(8, 27)	0.344 ± 0.007	1.661 ± 0.022	4.004 ± 0.048	7.376 ± 0.061
4	(16, 81)	1.997 ± 0.077	9.895 ± 0.109	23.512 ± 0.179	43.188 ± 0.454
Reachability & Reaction:					
3	(6, 21)	0.090 ± 0.001	0.424 ± 0.016	1.037 ± 0.004	2.044 ± 0.013
5	(20, 155)	1.628 ± 0.087	7.560 ± 0.023	18.019 ± 0.129	33.539 ± 0.144
7	(68, 1065)	44.809 ± 0.996	209.612 ± 1.732	488.611 ± 6.308	869.060 ± 16.870
Reachability & Safety:					
3	(6, 18)	0.102 ± 0.002	0.508 ± 0.010	1.278 ± 0.022	2.557 ± 0.023
4	(6, 18)	0.116 ± 0.002	0.590 ± 0.009	1.485 ± 0.024	2.918 ± 0.046
5	(6,18)	0.179 ± 0.027	0.960 ± 0.037	2.329 ± 0.072	4.482 ± 0.116

the percentage of instances in which at least one feasible solution was returned within the allotted time. In addition to the gridsize, the specification length was also scaled for three classes of system and test objectives: i) reachability, ii) reachability and reaction, and iii) reachability and safety. In the first case with reachability objectives, the system and test specification are $\varphi_{\text{sys}} = \Diamond p_0$ and $\varphi_{\text{test}} = \bigwedge_{i=1}^n \Diamond p_i$, and the total number of atomic propositions are $|AP| = |\{p_0, \dots, p_n\}| = n+1$, scaled upto $n = 3$ (or $|AP| = 4$). For the reachability and reaction objectives, the system objective comprises of a reachability objective and a conjunction of delayed reaction specification pattern: $\varphi_{\text{sys}} = \Diamond p_0 \wedge \bigwedge_{i=1}^n \Box(p_i \rightarrow \Diamond q_i)$. The test objective for this case is a conjunction of the triggers corresponding to the system objective: $\varphi_{\text{test}} = \bigwedge_{i=1}^n \Diamond p_i$. Therefore, the total number of atomic propositions are $|AP| = |\{p_0, \dots, p_n, q_1, \dots, q_n\}| = 2n+1$, scaled upto $n = 3$. Finally, in the reaction and safety case, the system objective consists of reachability and safety specifications: $\varphi_{\text{sys}} = \Diamond p_1 \wedge \bigwedge_{i=2}^n \Box \neg p_i$ and the test objective is a single reachability specification $\varphi_{\text{test}} = \Diamond p_0$. The total number of atomic propositions are $|AP| = |\{p_0, \dots, p_n\}| = n+1$, scaled upto $n = 3$. Note that only the length of the system objective changes as the specification size is increased.

Table 4.9 lists the sizes of the specifications as well as runtimes for graph construction (mean and standard deviation across 20 instances). The product graph construction is a basic implementation in Python, and is not optimized for speed. In future work, off-the-shelf symbolic methods can be leveraged to compute the product graphs much more quickly.

4.12 Conclusions and Future Work

This work on flow-based synthesis of test strategies can help test engineers automatically synthesize test environments (e.g., where should obstacles be placed, how should the test agent strategy be implemented) that are guaranteed to meet specified system and test objectives. This chapter simplifies the routing optimization introduced in the previous chapter, and presents MILP formulations for the different types of test environments. Furthermore, this chapter shows how a reactive test strategy can inform the choice of a test agent and also find an agent strategy that implements the reactive test strategy. This is made possible by via GR(1) synthesis and a counter-example guided approach to resolving the MILP to exclude dynamically infeasible test strategies. Another important contribution of this chapter is in establishing the computational complexity of the routing problem, which means that using an MILP for the routing problem is an appropriate choice. Despite the combinatorial nature of the problem, extensive experiments show that it can handle medium-sized problem instances (thousands of integer variables) in a reasonable time. The synthesized test strategies are reactive to system behavior, and route it through the test objective, and if the system demonstrates unsafe behavior, it is a fault in the system design. When the routing problem is solved to optimality, the resulting test strategy is not overly-restrictive, and the is realized with the fewest number of obstacles.

There are several exciting directions for future work. First, this framework can be extended to automatically select from a library of test agents to optimize for testing cost. Secondly, the use of symbolic methods in graph construction to improve the overall runtime of the framework. Thirdly, finding good convex relaxations to the MILP would result in dramatic speed-up since we would only have to solve a linear program. However, a straight-forward convex relaxation on the binary variables does not return meaningful solutions; finding an often-tight convex solution would require more careful study. Fourth, integrating the high-level test synthesis in this work with dynamics from lower levels of the control hierarchy is an important open problem. This effort would include: i) interfacing with falsification tools to automatically synthesis difficult tests, and ii) incorporate timing constraints into system and test objectives. Finally, we must relate the synthesized tests to a notion of coverage, and choose system and test objectives that that maximize the coverage metric. A more comprehensive discussion on future directions is given in Chapter 6.

Bibliography

[1] Zoox, "Putting Zoox to the test: preparing for the challenges of the road," 2021. `https://zoox.com/journal/structured-testing/`, Last accessed on 2024-04-11.

[2] Waymo, "A blueprint for av safety: Waymo's toolkit for building a credible safety case," 2020. `https://waymo.com/blog/2023/03/a-blueprint-for-av-safety-waymos/#:~:text=A%20safety%20case%20for%20fully,evidence%20to%20support%20that%20determination.`, Last accessed on 2024-05-05.

[3] F. Favarò, L. Fraade-Blanar, S. Schnelle, T. Victor, M. Peña, J. Engstrom, J. Scanlon, K. Kusano, and D. Smith, "Building a credible case for safety: Waymo's approach for the determination of absence of unreasonable risk," 2023. www.waymo.com/safety.

[4] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?," *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016.

[5] N. Webb, D. Smith, C. Ludwick, T. Victor, Q. Hommes, F. Favaro, G. Ivanov, and T. Daniel, "Waymo's safety methodologies and safety readiness determinations," 2020.

[6] I. S. Organization, "Road vehicles: Safety of the intended functionality (ISO Standard No. 21448:2022)," 2022. `https://www.iso.org/standard/77490.html`, Last accessed on 2024-04-11.

[7] L. Li, W.-L. Huang, Y. Liu, N.-N. Zheng, and F.-Y. Wang, "Intelligence testing for autonomous vehicles: A new approach," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 2, pp. 158–166, 2016.

[8] H. Winner, K. Lemmer, T. Form, and J. Mazzega, "Pegasus—first steps for the safe introduction of automated driving," in *Road Vehicle Automation 5*, pp. 185–195, Springer, 2019.

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

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

[11] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016.

[12] J. Eskenazi and W. Jarett, "Explore: See the 55 reports — so far — of robot cars interfering with SF fire dept.," 2023. https://missionlocal.org/2023/08/cruise-waymo-autonomous-vehicle-robot-taxi-driverless-car-reports-san-francisco/, Last accessed on 2024-04-11.

[13] H. Zhao, S. K. Sastry Hari, T. Tsai, M. B. Sullivan, S. W. Keckler, and J. Zhao, "Suraksha: A framework to analyze the safety implications of perception design choices in avs," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pp. 434–445, 2021.

[14] 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.

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

[16] M. Lahijanian, S. B. Andersson, and C. Belta, "A probabilistic approach for control of a stochastic system from LTL specifications," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pp. 2236–2241, IEEE, 2009.

[17] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *53rd IEEE Conference on Decision and Control*, pp. 81–87, IEEE, 2014.

[18] 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.

[19] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*, pp. 97–117, Springer, 2017.

[20] M. Fazlyab, M. Morari, and G. J. Pappas, "Probabilistic verification and reachability analysis of neural networks via semidefinite programming," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 2726–2731, IEEE, 2019.

[21] M. Fazlyab, M. Morari, and G. J. Pappas, "Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming," *IEEE Transactions on Automatic Control*, 2020.

[22] H.-D. Tran, X. Yang, D. M. Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, "NNV: The neural network verification tool for

deep neural networks and learning-enabled cyber-physical systems," in *International Conference on Computer Aided Verification*, pp. 3–17, Springer, 2020.

[23] T. Dreossi, S. Jha, and S. A. Seshia, "Semantic adversarial deep learning," in *International Conference on Computer Aided Verification*, pp. 3–26, Springer, 2018.

[24] S. A. Seshia, A. Desai, T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, S. Shivakumar, M. Vazquez-Chanlatte, and X. Yue, "Formal specification for deep neural networks," in *International Symposium on Automated Technology for Verification and Analysis*, pp. 20–34, Springer, 2018.

[25] T. Dreossi, A. Donzé, and S. A. Seshia, "Compositional falsification of cyber-physical systems with machine learning components," *Journal of Automated Reasoning*, vol. 63, no. 4, pp. 1031–1053, 2019.

[26] S. Topan, K. Leung, Y. Chen, P. Tupekar, E. Schmerling, J. Nilsson, M. Cox, and M. Pavone, "Interaction-dynamics-aware perception zones for obstacle detection safety evaluation," in *2022 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1201–1210, IEEE, 2022.

[27] K. Chakraborty and S. Bansal, "Discovering closed-loop failures of vision-based controllers via reachability analysis," *IEEE Robotics and Automation Letters*, vol. 8, no. 5, pp. 2692–2699, 2023.

[28] A. Dokhanchi, H. B. Amor, J. V. Deshmukh, and G. Fainekos, "Evaluating perception systems for autonomous vehicles using quality temporal logic," in *International Conference on Runtime Verification*, pp. 409–416, Springer, 2018.

[29] A. Balakrishnan, A. G. Puranic, X. Qin, A. Dokhanchi, J. V. Deshmukh, H. B. Amor, and G. Fainekos, "Specifying and evaluating quality metrics for vision-based perception systems," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1433–1438, IEEE, 2019.

[30] B. Bauchwitz and M. Cummings, "Evaluating the reliability of Tesla model 3 driver assist functions," 2020.

[31] H. Kress-Gazit, D. C. Conner, H. Choset, A. A. Rizzi, and G. J. Pappas, "Courteous cars," *IEEE Robotics & Automation Magazine*, vol. 15, no. 1, pp. 30–38, 2008.

[32] H. Kress-Gazit and G. J. Pappas, "Automatically synthesizing a planning and control subsystem for the DARPA Urban Challenge," in *2008 IEEE International Conference on Automation Science and Engineering*, pp. 766–771, IEEE, 2008.

[33] T. Wongpiromsarn, S. Karaman, and E. Frazzoli, "Synthesis of provably correct controllers for autonomous vehicles in urban environments," in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pp. 1168–1173, IEEE, 2011.

[34] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Conference on Robot Learning*, pp. 1–16, PMLR, 2017.

[35] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: a language for scenario specification and scene generation," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 63–78, 2019.

[36] 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*, pp. 254–257, Springer, 2011.

[37] 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.

[38] G. E. Fainekos, S. Sankaranarayanan, K. Ueda, and H. Yazarel, "Verification of automotive control applications using s-taliro," in *2012 American Control Conference (ACC)*, pp. 3567–3572, IEEE, 2012.

[39] 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*, pp. 125–134, 2012.

[40] S. Bak, S. Bogomolov, A. Hekal, N. Kochdumper, E. Lew, A. Mata, and A. Rahmati, "Falsification using reachability of surrogate koopman models," in *Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '24, (New York, NY, USA), Association for Computing Machinery, 2024.

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

[42] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski, "Simulation-based adversarial test generation for autonomous vehicles with machine learning components," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1555–1562, IEEE, 2018.

[43] C. Menghi, P. Arcaini, W. Baptista, G. Ernst, G. Fainekos, F. Formica, S. Gon, T. Khandait, A. Kundu, G. Pedrielli, *et al.*, "Arch-comp 2023 category report: Falsification," in *10th International Workshop on Applied Verification of Continuous and Hybrid Systems. ARCH23*, vol. 96, pp. 151–169, 2023.

[44] 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*, pp. 432–442, Springer, 2019.

[45] A. Corso, P. Du, K. Driggs-Campbell, and M. J. Kochenderfer, "Adaptive stress testing with reward augmentation for autonomous vehicle validatio," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 163–168, IEEE, 2019.

[46] S. Feng, H. Sun, X. Yan, H. Zhu, Z. Zou, S. Shen, and H. X. Liu, "Dense reinforcement learning for safety validation of autonomous vehicles," *Nature*, vol. 615, no. 7953, pp. 620–627, 2023.

[47] X. Qin, N. Arechiga, J. Deshmukh, and A. Best, "Robust testing for cyber-physical systems using reinforcement learning," in *Proceedings of the 21st ACM-IEEE International Conference on Formal Methods and Models for System Design*, MEMOCODE '23, (New York, NY, USA), p. 36–46, Association for Computing Machinery, 2023.

[48] S. A. Seshia, D. Sadigh, and S. S. Sastry, "Toward verified artificial intelligence," *Commun. ACM*, vol. 65, p. 46–55, jun 2022.

[49] B. Johnson and H. Kress-Gazit, "Probabilistic analysis of correctness of high-level robot behavior with sensor error," 2011.

[50] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[51] X. Wang, R. Li, B. Yan, and O. Koyejo, "Consistent classification with generalized metrics," 2019.

[52] P. Antonante, H. Nilsen, and L. Carlone, "Monitoring of perception systems: Deterministic, probabilistic, and learning-based fault detection and identification," *arXiv preprint arXiv:2205.10906*, 2022.

[53] M. Hekmatnejad, S. Yaghoubi, A. Dokhanchi, H. B. Amor, A. Shrivastava, L. Karam, and G. Fainekos, "Encoding and monitoring responsibility sensitive safety rules for automated vehicles in signal temporal logic," in *Proceedings of the 17th ACM-IEEE International Conference on Formal Methods and Models for System Design*, pp. 1–11, 2019.

[54] T. Wongpiromsarn and E. Frazzoli, "Control of probabilistic systems under dynamic, partially known environments with temporal logic specifications," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 7644–7651, 2012.

[55] A. Badithela, T. Wongpiromsarn, and R. M. Murray, "Leveraging classification metrics for quantitative system-level analysis with temporal logic specifications," in *2021 60th IEEE Conference on Decision and Control (CDC)*, (Austin, TX, USA (virtual)), pp. 564–571, IEEE, 2021.

[56] C. S. Pasareanu, R. Mangal, D. Gopinath, S. G. Yaman, C. Imrie, R. Calinescu, and H. Yu, "Closed-loop analysis of vision-based autonomous systems: A case study," *arXiv preprint arXiv:2302.04634*, 2023.

[57] S. Beland, I. Chang, A. Chen, M. Moser, J. Paunicka, D. Stuart, J. Vian, C. Westover, and H. Yu, "Towards assurance evaluation of autonomous systems," in *Proceedings of the 39th International Conference on Computer-Aided Design*, pp. 1–6, 2020.

[58] Y. V. Pant, H. Abbas, K. Mohta, R. A. Quaye, T. X. Nghiem, J. Devietti, and R. Mangharam, "Anytime computation and control for autonomous systems," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 2, pp. 768–779, 2021.

[59] P. Karkus, B. Ivanovic, S. Mannor, and M. Pavone, "Diffstack: A differentiable and modular control stack for autonomous vehicles," in *Proceedings of The 6th Conference on Robot Learning* (K. Liu, D. Kulic, and J. Ichnowski, eds.), vol. 205 of *Proceedings of Machine Learning Research*, pp. 2170–2180, PMLR, 14–18 Dec 2023.

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

[61] O. Koyejo, N. Natarajan, P. Ravikumar, and I. S. Dhillon, "Consistent multilabel classification.," in *NeurIPS*, vol. 29, (Palais des Congrès de Montréal, Montréal CANADA), pp. 3321–3329, Advances in Neural Information Processing Systems, 2015.

[62] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in *International conference on computer aided verification*, pp. 585–591, Springer, 2011.

[63] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A Storm is coming: A modern probabilistic model checker," in *International Conference on Computer Aided Verification*, pp. 592–600, Springer, 2017.

[64] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11621–11631, 2020.

[65] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (Los Alamitos, CA, USA), pp. 12689–12697, IEEE Computer Society, jun 2019.

[66] M. Contributors, "MMDetection3D: OpenMMLab next-generation platform for general 3D object detection." `https://github.com/open-mmlab/mmdetection3d`, 2020.

[67] S. Gupta, J. Kanjani, M. Li, F. Ferroni, J. Hays, D. Ramanan, and S. Kong, "Far3det: Towards far-field 3d detection," in *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, (Los Alamitos, CA, USA), pp. 692–701, IEEE Computer Society, jan 2023.

[68] I. Incer, A. Badithela, J. Graebener, P. Mallozzi, A. Pandey, S.-J. Yu, A. Benveniste, B. Caillaud, R. M. Murray, A. Sangiovanni-Vincentelli, *et al.*, "Pacti: Scaling assume-guarantee reasoning for system analysis and design," *arXiv preprint arXiv:2303.17751*, 2023.

[69] A. Badithela, T. Wongpiromsarn, and R. M. Murray, "Evaluation metrics of object detection for quantitative system-level analysis of safety-critical autonomous systems," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Detroit, MI, USA), p. To Appear., IEEE, 2023.

[70] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *International Conference on Formal Modeling and Analysis of Timed Systems*, pp. 92–106, Springer, 2010.

[71] 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.

[72] 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.

[73] T. Wongpiromsarn, M. Ghasemi, M. Cubuktepe, G. Bakirtzis, S. Carr, M. O. Karabag, C. Neary, P. Gohari, and U. Topcu, "Formal methods for autonomous systems," *arXiv preprint arXiv:2311.01258*, 2023.

[74] G. Fainekos, H. Kress-Gazit, and G. Pappas, "Hybrid controllers for path planning: A temporal logic approach," in *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 4885–4890, 2005.

[75] R. Majumdar, A. Mathur, M. Pirron, L. Stegner, and D. Zufferey, "Paracosm: A language and tool for testing autonomous driving systems," *arXiv preprint arXiv:1902.01084*, 2019.

[76] L. Tan, O. Sokolsky, and I. Lee, "Specification-based testing with linear temporal logic," in *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, 2004. IRI 2004.*, pp. 493–498, IEEE, 2004.

[77] G. Fraser and F. Wotawa, "Using LTL rewriting to improve the performance of model-checker based test-case generation," in *Proceedings of the 3rd International Workshop on Advances in Model-Based Testing*, pp. 64–74, 2007.

[78] G. Fraser and P. Ammann, "Reachability and propagation for LTL requirements testing," in *2008 The Eighth International Conference on Quality Software*, pp. 189–198, IEEE, 2008.

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

[80] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[81] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger, "Specification patterns for robotic missions," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, pp. 2208–2224, 2019.

[82] R. Bloem, G. Fey, F. Greif, R. Könighofer, I. Pill, H. Riener, and F. Röck, "Synthesizing adaptive test strategies from temporal logic specifications," *Formal methods in system design*, vol. 55, no. 2, pp. 103–135, 2019.

[83] J. Tretmans, "Conformance testing with labelled transition systems: Implementation relations and test generation," *Computer Networks and ISDN Systems*, vol. 29, no. 1, pp. 49–79, 1996.

[84] B. K. Aichernig, H. Brandl, E. Jöbstl, W. Krenn, R. Schlick, and S. Tiran, "Killing strategies for model-based mutation testing," *Software Testing, Verification and Reliability*, vol. 25, no. 8, pp. 716–748, 2015.

[85] R. Hierons, "Applying adaptive test cases to nondeterministic implementations," *Information Processing Letters*, vol. 98, no. 2, pp. 56–60, 2006.

[86] A. Petrenko and N. Yevtushenko, "Adaptive testing of nondeterministic systems with FSM," in *2014 IEEE 15th International Symposium on High-Assurance Systems Engineering*, pp. 224–228, IEEE, 2014.

[87] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 179–190, 1989.

[88] 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.

[89] M. Yannakakis, "Testing, optimization, and games," in *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004.*, pp. 78–88, IEEE, 2004.

[90] L. Nachmanson, M. Veanes, W. Schulte, N. Tillmann, and W. Grieskamp, "Optimal strategies for testing nondeterministic systems," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 4, pp. 55–64, 2004.

[91] A. David, K. G. Larsen, S. Li, and B. Nielsen, "Cooperative testing of timed systems," *Electronic Notes in Theoretical Computer Science*, vol. 220, no. 1, pp. 79–92, 2008.

[92] E. Bartocci, R. Bloem, B. Maderbacher, N. Manjunath, and D. Ničković, "Adaptive testing for specification coverage in CPS models," *IFAC-PapersOnLine*, vol. 54, no. 5, pp. 229–234, 2021.

[93] T. Marcucci, J. Umenberger, P. Parrilo, and R. Tedrake, "Shortest paths in graphs of convex sets," *SIAM Journal on Optimization*, vol. 34, no. 1, pp. 507–532, 2024.

[94] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," *Science Robotics*, vol. 8, no. 84, p. eadf7843, 2023.

[95] H. Zhang, M. Fontaine, A. Hoover, J. Togelius, B. Dilkina, and S. Nikolaidis, "Video game level repair via mixed integer linear programming," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, pp. 151–158, 2020.

[96] M. Fontaine, Y.-C. Hsu, Y. Zhang, B. Tjanaka, and S. Nikolaidis, "On the Importance of Environments in Human-Robot Coordination," in *Proceedings of Robotics: Science and Systems*, (Virtual), July 2021.

[97] J. R. Büchi, *On a Decision Method in Restricted Second Order Arithmetic*, pp. 425–435. New York, NY: Springer New York, 1990.

[98] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, É. Renault, and L. Xu, "Spot 2.0 — a framework for ltl and omega-automata manipulation," in *Automated Technology for Verification and Analysis* (C. Artho, A. Legay, and D. Peled, eds.), (Cham), pp. 122–129, Springer International Publishing, 2016.

[99] F. Fuggitti, "Ltlf2dfa," June 2020.

[100] S. Bansal, Y. Li, L. Tabajara, and M. Vardi, "Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 9766–9774, Apr. 2020.

[101] N. Klarlund and A. Møller, *MONA Version 1.4 User Manual*. BRICS, Department of Computer Science, University of Aarhus, January 2001. Notes Series NS-01-1. Available from `http://www.brics.dk/mona/`.

[102] D. Goktas and A. Greenwald, "Convex-concave min-max Stackelberg games," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[103] 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.

[104] M. L. Bynum, G. A. Hackebeil, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Siirola, J.-P. Watson, and D. L. Woodruff, *Pyomo–optimization modeling in python*, vol. 67. Springer Science & Business Media, third ed., 2021.

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

[106] M. Fischetti and M. Monaci, "A branch-and-cut algorithm for mixed-integer bilinear programming," *European Journal of Operational Research*, vol. 282, no. 2, pp. 506–514, 2020.

[107] J. B. Graebener, A. S. Badithela, D. Goktas, W. Ubellacker, E. V. Mazumdar, A. D. Ames, and R. M. Murray, "Flow-based synthesis of reactive tests for discrete decision-making systems with temporal logic specifications," *arXiv preprint arXiv:2404.09888*, 2024.

[108] 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*, pp. 313–314, 2011.

[109] I. Filippidis, S. Dathathri, S. C. Livingston, N. Ozay, and R. M. Murray, "Control design for hybrid systems with tulip: The temporal logic planning toolbox," in *2016 IEEE Conference on Control Applications (CCA)*, pp. 1030–1041, IEEE, 2016.

[110] S. Maoz and J. O. Ringert, "Gr (1) synthesis for ltl specification patterns," in *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, pp. 96–106, 2015.

[111] S. A. Cook, "The complexity of theorem-proving procedures," in *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook*, pp. 143–152, 2023.

[112] C. H. Papadimitriou, *Computational complexity*, p. 260–265. GBR: John Wiley and Sons Ltd., 2003.

[113] W. Ubellacker and A. D. Ames, "Robust locomotion on legged robots through planning on motion primitive graphs," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 12142–12148, 2023.

[114] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023.

[115] E. W. Dijkstra, "Guarded commands, nondeterminacy and formal derivation of programs," *Communications of the ACM*, vol. 18, no. 8, pp. 453–457, 1975.

[116] L. Lamport, "win and sin: Predicate transformers for concurrency," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 12, no. 3, pp. 396–428, 1990.

[117] B. Meyer, "Applying 'design by contract'," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[118] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis, "Multiple viewpoint contract-based specification and design," in *Formal Methods for Components and Objects: 6th International Symposium, FMCO 2007, Amsterdam, The Netherlands, October 24-26, 2007, Revised Lectures* (F. S. de Boer, M. M. Bonsangue, S. Graf, and W.-P. de Roever, eds.), (Berlin, Heidelberg), pp. 200–225, Springer Berlin Heidelberg, 2008.

[119] A. L. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming Dr. Frankenstein: Contract-based design for cyber-physical systems," *Eur. J. Control*, vol. 18, no. 3, pp. 217–238, 2012.

[120] P. Nuzzo, A. L. Sangiovanni-Vincentelli, D. Bresolin, L. Geretti, and T. Villa, "A platform-based design methodology with contracts and related tools for the design of cyber-physical systems," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 2104–2132, 2015.

[121] I. Incer, *The Algebra of Contracts*. PhD thesis, EECS Department, University of California, Berkeley, May 2022.

[122] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. L. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, K. G. Larsen, *et al.*, "Contracts for system design," *Foundations and Trends in Electronic Design Automation*, vol. 12, no. 2-3, pp. 124–400, 2018.

[123] I. Incer, A. L. Sangiovanni-Vincentelli, C.-W. Lin, and E. Kang, "Quotient for assume-guarantee contracts," in *16th ACM-IEEE International Conference on Formal Methods and Models for System Design*, MEMOCODE'18, pp. 67–77, October 2018.

[124] R. Passerone, Í. Íncer Romeo, and A. L. Sangiovanni-Vincentelli, "Coherent extension, composition, and merging operators in contract models for system design," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–23, 2019.

[125] R. Negulescu, "Process spaces," in *CONCUR 2000 — Concurrency Theory* (C. Palamidessi, ed.), (Berlin, Heidelberg), pp. 199–213, Springer Berlin Heidelberg, 2000.

[126] J. B. Graebener^*, A. Badithela^*, and R. M. Murray, "Towards better test coverage: Merging unit tests for autonomous systems," in *NASA Formal Methods* (J. V. Deshmukh, K. Havelund, and I. Perez, eds.), (Cham), pp. 133–155, Springer International Publishing, 2022. A. Badithela and J.B. Graebener contributed equally to this work.

[127] R. Bloem, B. Könighofer, R. Könighofer, and C. Wang, "Shield synthesis," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 533–548, Springer, 2015.

[128] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*, pp. 282–293, Springer, 2006.

[129] I. Incer, L. Mangeruca, T. Villa, and A. Sangiovanni-Vincentelli, "The quotient in preorder theories," *arXiv:2009.10886*, 2020.

[130] O. Hussien, A. Ames, and P. Tabuada, "Abstracting partially feedback linearizable systems compositionally," *IEEE Control Systems Letters*, vol. 1, no. 2, pp. 227–232, 2017.

[131] P. Tabuada, G. J. Pappas, and P. Lima, "Composing abstractions of hybrid systems," in *International Workshop on Hybrid Systems: Computation and Control*, pp. 436–450, Springer, 2002.

[132] S. Coogan and M. Arcak, "Efficient finite abstraction of mixed monotone systems," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, HSCC '15, (New York, NY, USA), p. 58–67, Association for Computing Machinery, 2015.

[133] J. Liu and N. Ozay, "Abstraction, discretization, and robustness in temporal logic control of dynamical systems," in *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pp. 293–302, 2014.