Chapter 3

AUTOMATED TEST SYNTHESIS VIA NETWORK FLOWS: AN INTRODUCTION

3.1 Introduction

This chapter explores reactive test synthesis for discrete decision-making components in autonomous systems. This chapter originated from thinking about how to find a small set of difficult test cases for discrete decision-making behaviors. For safety as well as satisfying system requirements, a full-stack autonomous system must reason over its *own state* as well as about how the environment might *react* to its actions. Oftentimes, this involves reasoning over inputs and states that are both discrete and continuous valued, and implementations of autonomous systems accomplish this at various levels of abstraction. In this chapter, we formulate the test synthesis problem, and introduce the concept of a test objective.

This chapter is adapted from:

A. Badithela, R. M. Murray. (2020). "Synthesis of Static Test Environments for Observing Sequence-like Behaviors in Autonomous Systems." arXiv preprint: https://arxiv.org/pdf/2108.05911.

3.2 Related Work

Due to robustness metrics from their quantitative semantics, signal temporal logic (STL) and metric temporal logic (MTL), are natural paradigms for reasoning over trajectories of low-level continuous dynamics [37, 70]. In many instances, the term testing is used inter-changeably with falsification [39]. Falsification is the problem of finding initial conditions and input signals that lead to violation of a temporal logic formula with the goal of finding such failures quickly and for black-box models [36, 41, 71, 72]. Furthermore, the black-box approaches in the related topics of falsification of hybrid systems [36], and simulation-based test generation [38, 42], rely on stochastic optimization algorithms to minimize the robustness of temporal logic satisfaction. Since dense-time temporal logics better encapsulate the range of system behaviors at the with continuous dynamics, these techniques are successful at falsification at the low-level. However, some of the complexity can be attributed to the coupling between continuous dynamics with high-level discrete decision-making behaviors, a hierarchical approach to test-case generation could be effective.

Typically, high-level choices of autonomous robotic systems exhibit discrete decision-making [73, 74], and LTL specifications are often used to capture mission objectives at higher levels of abstraction. Covering arrays have been used to initialize discrete parameters of the test configuration at the start of the falsification procedure in [36, 42, 75], but are not reactive. Furthermore, linear temporal logic (LTL) model checkers for testing has been explored in [71, 76–78], in which counterexamples found via model-checking are used to exactly construct test cases. However, these are usually applied to deterministic systems, thus relying on the knowledge of the system controller, and become inconclusive if the system behavior deviates from the expected model. In this chapter and next, we focus on a framework for testing of high-level specifications in linear temporal logic (LTL) without assuming knowledge of the system controller.

3.3 Motivation

Here we adopt a different notion of testing – one that is focused on observing the autonomous agent undertake a certain behavior in its mission. The DARPA Urban Challenge test courses, that mainly comprised of static obstacles and (dynamic) human-driven cars, were carefully designed to observe the agent undertaking certain behaviors [10]. For example, a part of the test course was designed for assessing parking behavior. The static obstacles – barriers blocking the region in front of the parking lot and other parked cars – were placed such that the agent had to repeatedly reverse/pull-in to incrementally adjust its heading angle before successfully parking in the designated spot. The clever placement of static obstacles in this scenario made it a challenging test for the agent, as opposed to an environment in which the agent pulls-in straight into the parking spot. Similarly, carefully designed scenarios with human-driven cars sought to observe other behaviors of the agent. In many, but not necessarily all, of these scenarios, the high-level behavior of the agent can be described as a sequence of waypoints. In the parking lot example, the sequence of waypoints can be characterized as a sequence of agent states, which can be characterized as a product of position and heading angle in the high-level abstraction. As a step towards automatically synthesizing these test scenarios, this chapter asks the following question.

Problem (Informal): Given a valid, user-defined sequence of waypoints, a reacha-

bility objective for the mission specification, find a set of possible initial conditions for the agent (if not specified by user) and determine a set of static constraints, characterized by transitions that are blocked/restricted, such that:

i. *the agent must visit the sequence of waypoints in order before its goal, and*

ii. *the test environment is minimally restricted.*

3.4 Preliminaries

Automata Theory and Temporal Logic

Definition 3.1 (Finite Transition System)**.** A *finite transition system* (FTS) is the tuple

$$TS := (S, A, \delta, S_0, AP, L),$$

where S denotes a finite set of states, A is a finite set of actions, $\delta : S \times A \to S$ the transition relation, S_0 the set of initial states, AP the set of atomic propositions, and $L : S \to 2^{AP}$ denotes the labeling function. We denote the transitions in TS as $TS.E := \{(s, s') \in S \times S \mid \text{ if } \exists a \in A \text{ s.t. } \delta(s, a) = s'\}$. We refer to the states of TS as $TS.S$, and similarly denote the other elements of the tuple. An execution σ is an infinite sequence $\sigma = s_0 s_1 \ldots$, where $s_0 \in S_0$ and $s_k \in S$ is the state at time k. We denote the finite prefix of the trace σ up to the current time k as σ_k. A strategy π is a function $\pi : (TS.S)^* TS.S \to TS.A$.

Definition 3.2 (System)**.** The *system under test* is modeled as a finite transition system T_{sys} with a singleton initial set, $|T_{\text{sys}}.S_0| = 1$.

A directed graph $G = (V, E)$ can be induced from T_{sys} in which the vertices represent states $T_{\text{sys}}.S$ and the edges represent the transitions $T_{\text{sys}}.E$, and the labeling function assigns propositions that are true at each vertex. For a proposition $p \in AP$, and vertex $v \in V$, $v \vdash p$ means that p evaluates to $True$ at v. A *run* $\sigma = s_0 s_1 \ldots$ on the graph is an infinite sequence of its nodes where $s_i \in T_{\text{sys}}.S$ represents the system state at time step i.

We introduce the notion of a test harness to specify how the test environment can interact with the system. A test harness is used to constrain a state-action (s, a) pair of the system in the sense that the system is prevented from taking action a from state $s \in T_{\text{sys}}.S$. Let the actions $A_H \subseteq T_{\text{sys}}.A$ denote the subset of system actions that can be restricted by the test harness. The test harness $H : T_{\text{sys}}.S \to 2^{A_H}$ maps states of the transition system to actions that can be restricted from that state.

In the examples considered in this thesis, every state of the system has a self-loop transition corresponding to stay-in-place action, though the framework does not require this. Note that in our examples, A_H does not contain self-loop actions.

In this work, we synthesize tests for high-level decision-making components of the system under test and therefore model it as a discrete-state system. Linear temporal logic (LTL) has been effective in formally specifying safety and liveness requirements for discrete-decision making [14, 15, 18]. For our problem, we use LTL to capture the system and test objectives. The reach-avoid fragment of LTL is restricted to the use of logical operators and the *next*, *always*, and *eventually* temporal operators, and can capture a rich set of behaviors such as safety and coverage properties. Every LTL formula can be transformed into an equivalent non-deterministic Büchi automaton, which can then be converted to a deterministic Büchi automaton [60].

Flow Networks

We will leverage network flows to model the test synthesis problem. Flow networks are used in computer science to model several problems on graphs [79]. One of the main contributions of this thesis is in using flow networks for test synthesis.

Definition 3.3 (Flow Network [80])**.** A *flow network* is a tuple $\mathcal{N} = (V, E, c, (V_s, V_t))$, where V denotes the set of nodes, $E \subseteq V \times V$ the set of edges, $c \geq 0$ represents edge capacity, $V_s \subseteq V$ the source nodes, and $V_t \subseteq V$ the sink nodes. On the flow network $\mathcal{N}$, we can define the *flow* vector $\mathbf{f} \in \mathbb{R}_{\geq 0}^{|E|}$ to satisfy the following constraints: i) the capacity constraint

$$0 \leq f^e \leq c, \forall e \in E, \tag{3.1}$$

ii) the conservation constraint

$$\sum_{u \in V} f^{(u,v)} = \sum_{u \in V} f^{(v,u)}, \forall v \in V \setminus \{V_s, V_t\}, \text{ and} \tag{3.2}$$

iii) no flow into the source or out of the sink

$$f^{(u,v)} = 0 \text{ if } u \in V_t \text{ or } v \in V_s. \tag{3.3}$$

The flow value on the network $\mathcal{N}$ is defined as

$$F := \sum_{\substack{(u,v) \in E, \\ u \in V_s}} f^{(u,v)}. \tag{3.4}$$

In the following chapters, we will primarily be using the framework of network flows to identify restrictions on system actions, which will be analogous to cuts on a graph. An edge represents a transition the system can make. For this reason, it can suffice to define flow networks that have unit capacity $c = 1$ on all edges. Unless otherwise mentioned, all references to a flow network hereafter will assume unit edge capacities.

Definition 3.4 (Cut [80])**.** Given a graph $G = (V, E)$, a *cut* of G is the tuple $\texttt{Cut} := (S, T)$ that partitions the vertices of G into disjoint sets $S \subset V$ and $T \subset V$, that is, $S \cup T = V$ and $S \cap T = $. The *cut-set* $C \subseteq E$ of the cut $\texttt{Cut} = (S, T)$ is the set of edges that when removed from G results in the disjoint node sets S and T:

$$C \coloneqq \{(u, v) \in E \mid u \in S,\ v \in T\}. \tag{3.5}$$

The expression $G_{cut}(C) := (V, E \setminus C)$ refers to the graph resulting from remove the edges in C from G. We will use the same expression to refer to any graph from which edges C are removed, even if the set C does not correspond to a $\texttt{Cut}$ (i.e., complete partition of the graph G). Any edge that is removed from G is referred to as an *edge-cut*.

In finding maximum flow, it becomes important to identify edges on the graph through which flow can be pushed through and track edges which have already been saturated. This is the concept of a residual network which is defined below. For a more detailed exposition with illustrations, see pages 726–727 of [80].

Definition 3.5. Given a graph $G = (V, E)$ and a flow f, the set of residual edges E_f are defined as

$$E_f \coloneqq E \cup \{(u, v) \mid (v, u) \in E \text{ and } f(v, u) > 0\}. \tag{3.6}$$

The corresponding *residual network* $G_f = (V, E_f)$ is a flow network with edge capacities $c_f : E_f \to [0, 1]$ defined as follows:

$$c_f(u, v) = \begin{cases} 1 - f(u, v) & \text{if } (u, v) \in E \text{ and } f(u, v) < 1, \\ f(v, u) & \text{if } (v, u) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{3.7}$$

Standard algorithms such as Edmonds-Karp [79] use residual networks to find the maximum-flow (and equivalently, minimum-cut) of a single source-sink flow problem in a graph $G = (V, E)$ in $O(|V||E|^2)$ time. Roughly, starting with zero flow, the Edmonds-Karp algorithm iteratively updates the residual network as flow from the source to target is found. Initially, the residual network is exactly the same as the original graph G. Then, the algorithm finds paths from source to target on the residual network until no such paths remain. These paths are known as augmenting paths, and are used to construct a realization of the maximum flow.

Definition 3.6 (Augmenting Path [80])**.** Given a graph $G = (V, E)$ with flow f, an *augmenting path* from a source $s \in V$ to target $t \in V$ is a simple path $Path(s,t)$(i.e., without any cycles) from s to t on the corresponding residual network G_f.

The time complexity of the Edmonds-Karp algorithm comes from finding the shortest augmenting path on the residual network in each iteration until no paths remain. If G has a maximum possible flow $F_{\max}$, then the *set of augmenting paths* $\texttt{AP} = \{Path_1(s,t), \ldots, Path_{F_{\max}}(s,t)\}$ has cardinality $F_{\max}$ since all edges in G have unit capacities. That is, once an augmenting path is identified, all of the edges in the augmenting path are saturated on the residual network. As a result, any two paths $Path_i(s,t), Path_j(s,t) \in \texttt{AP}$ are always *edge-disjoint* in the sense that there does not exist any edge $e \in E$ that is in both $Path_i(s,t)$ and $Path_j(s,t)$. The set of augmenting paths found by the Edmonds-Karp algorithm is denoted as the *set of shortest augment paths* $\texttt{SAP}$. It can be proven that finding the shortest augmenting paths and updating the residual network accordingly results in finding a realization of the maximum flow [80].

Proposition 3.1. *A set of shortest augmenting paths* $\texttt{AP}$ *comprises of edge-disjoint paths.*

3.5 Test Objective

We begin by considering reachability specifications as mission objectives for the agent under test. For the test itself, we wish to observe a sequence-like behavior of the agent in its attempt to satisfy its mission objectives. Formally, this test behavior can be described by the temporal logic formula given below.

Definition 3.7 (Test Objective (strict sequence))**.** The *test objective* for strict sequenced visit is given by the LTL formula:

$$\varphi_{\text{test}} := \Diamond(p_1 \wedge \Diamond(p_2 \wedge \Diamond(\cdots \wedge \Diamond p_n))) \bigwedge_{i=1}^{n-1} (\neg p_{i+1} \mathsf{U}\, p_i)\,, \tag{3.8}$$

where $p_1, \dots, p_n \in AP$ are propositional formulas. This is a sequence-like formula since the agent has to eventually visit every v_i, but it cannot visit v_{i+1} before visiting v_i, where $v_i \vdash p_i$, for all $i = 1, \dots, n$.

The system under test does not have access to the test objective φ_{test}. Since LTL formulae cannot be evaluated on finite test runs, the length of the test run depends on the time the agent takes to satisfy its mission objective.

Example 3.1. Consider the gridworld in Figure 3.1 on which the agent can transition between states (up, down, left, right) with the mission specification of reaching some goal state (formalized as $\varphi_g = \lozenge g$). Of the many possible paths the agent can take to meet its objective, we are interested in observing it navigate to the goal while restricted to a class of paths described by the test specification $\varphi_{test} = \lozenge(p_1 \wedge \lozenge p_2)$. How would we constrain actions of the agent in certain states, such that it navigates through the sequence of waypoints before reaching the goal? Furthermore, is it possible to synthesize these constraints such that the sequence flow value from p_1 to g is maximized?

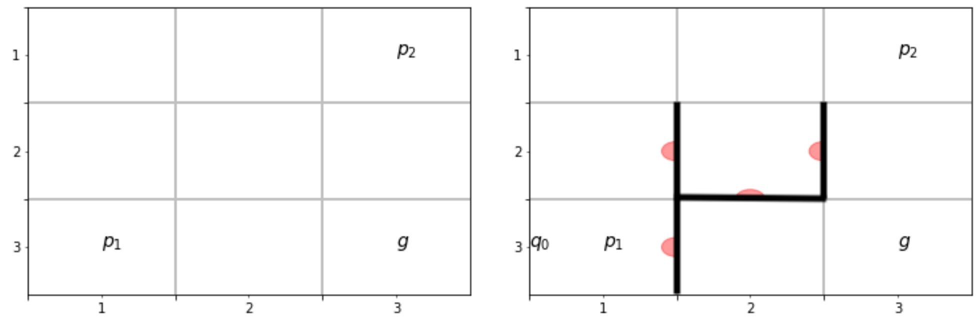


Figure 3.1: Left: Unrestricted gridworld labeled by propositional formulas. Right: A test environment synthesized by our algorithm where the transitions $(2, 1) \to (2, 2)$, $(3, 1) \to (3, 2)$, $(2, 2) \to (3, 2)$, and $(2, 2) \to (2, 3)$ blocked. Red semi-circle patches illustrate one-way constraints, that is, transition from state u to state v is restricted, but v to u is allowed, if arch of the semi-circle is in the grid corresponding to u along the transition from u to v.

Problem Statement

Now, we formalize the test environment synthesis problem. We limit our focus to *static* test environments, by which we mean that the test environment does not react to the actions of the agent during the test, leaving the reactive test synthesis problem for later in this chapter and the next chapter.

Definition 3.8 (Test Graph)**.** Given a labeled directed graph $G = (V, E)$, a mission/agent specification $\varphi_{sys} = \Diamond p_{n+1}$, a test specification φ_{test} (equation (3.8)), a *test graph* $G_{cut}(C) = (V, E \backslash C)$ is the directed graph obtained by removing set of edges C from the original graph G. On $G_{cut}(C)$, a run σ starting from state v_1 will satisfy the specification,

$$ \tag{3.9}$$

Definition 3.9 (Minimally Restricted Test Graph)**.** A test graph $G_{cut}(C)$ is *minimally restricted* if the total sequence flow value from $v_1 \models p_1$ to $v_{n+1} \models p_{n+1}$ on $G_{cut}(C)$ are maximized.

Remark 3.1. In this chapter, the definition of a *minimally restricted* graph does not relate to the actual number of cuts in the cut-set C, but only whether the flow on $G_{cut}(C)$ is maximized.

Problem 3.1. Given a system specification $\psi_{\text{sys}} = \Diamond v_{n+1}$, a labeled directed graph $G = (V, E)$ induced by the non-deterministic transition model $\mathcal{T}$ of the agent, a test specification $\varphi_{test} = \Diamond(p_1 \wedge \Diamond(p_2 \wedge \Diamond(\cdots \wedge \Diamond p_n)))$, static constraints $C \subseteq E$ such that on the resulting test graph $G_{cut}(C) = (V, E \backslash C)$ is a minimally restricted test graph.

Here we aim to find a cut that maximizes the flow from a waypoint p_i to its consecutive waypoint p_{i+1}, while eliminating any flow to waypoints p_j (for $j > i + 1$) for all $i = 1, \cdots, n$. In other words, some flows need to be cut while other flows should be maximized. The problem of constructing a minimally restricted test graph for observing a sequence-like specification can be cast as the following optimization,

$$\begin{aligned}
\max_{C \subset E} \quad & f_{G_{cut}(C)} \\
\text{s.t.} \quad & f_{G_{cut}(C)} \leq f_{G_{cut}(C)}(v_i, v_{i+1}) \qquad \forall i = 1, \cdots, n-1, \\
& f_{G_{cut}(C)}(v_i, v_j) = 0 \quad \forall i = 1, \cdots, j-2, \forall j = 3, \cdots, n,
\end{aligned} \tag{3.10}$$

where the variables $C \subset E$ are the set of edges to be restricted resulting in the unit-capacity graph $G_{cut}(C) = (V, E \backslash C)$, the scalar $f_{G_{cut}(C)}$ represents the total flow on $G_{cut}(C)$ from v_1 to v_n, the scalars $f_{G_{cut}(C)}(v_i, v_j)$ represent the total flow from source v_i to sink v_j. The problem data is the original graph $G = (V, E)$ and the sequence nodes $v_1, \ldots, v_n$. Solving this optimization directly will require constructing an integer linear program (ILP), for which constructing the constraint set is not straightforward. Furthermore, it would require solving the integer program with $|E|$ number of integer variables.

3.6 Algorithm for Synthesizing Static Test Environments

Let $G = (V, E)$ be a directed graph, with unit capacity on every edge, induced by the transition system T_{sys} of the system under test. Assuming that the test environment has complete freedom to "block" any transition in the graph G allowed by the test harness H, Algorithm 4 returns a set of edges, $C \subset E$, of the graph G that must be removed before the test run. First, we make following assumptions on G. Let $d_G(v_1, v_2)$ denote the length of the shortest path from vertex v_1 to vertex v_2 on graph G.

Assumption 3.1. For each $i \in \{1, \dots, n+1\}$, let v_i denote the vertex $v \in V$ s.t $v \vdash p_i$. Assume $|v_i| = 1$, for all $i = \{1, \dots, n+1\}$.

Informally, Assumption 3.1 states that every propositional formula, $p_1, \dots, p_{n+1}$, has a single vertex in G associated with it.

Assumption 3.2. There exists a set of edges $C \subseteq E$ such that the modified graph obtained by removing these edges, $G_{cut}(C) = (V, E \backslash C)$, is such that

$$d_{G_{cut}(C)}(v_1, v_{n+1}) > \cdots > d_{G_{cut}(C)}(v_n, v_{n+1}) > d_{G_{cut}(C)}(v_{n+1}, v_{n+1}) = 0. \quad (3.11)$$

Assumption 3.2 is equivalent to the statement that by removing some edges (or restricting certain transitions) from the original graph G, there exists some set of initial conditions Q_0 for which the only path(s) to the goal g is through the behavior φ_{test}. This assumption is imperative since there might be instances for which it is impossible to construct a test graph. For example, in the following simple labeled graph (Figure 3.2), it is impossible to construct a test graph for the test specification $\varphi_{\text{test}} = \Diamond(p_1 \wedge \Diamond p_2)$. Once the system is in state v_1, it can directly proceed to the goal state v_g without visiting v_2. For instances such as this one, a reactive test environment is necessary.

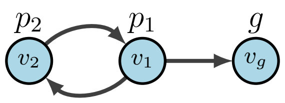


Figure 3.2: An invalid configuration of propositional formulas for test specification $\varphi_{\text{test}} = \Diamond(p_1 \wedge \Diamond p_2)$

Overview of the Approach: At a high-level, we identify all edge disjoint path combinations through the sequence specification to find edge restrictions. This can be seen as a brute-force approach to solving the problem. As we will discuss in the following chapter, this problem is NP-hard, and therefore, it is not possible to find a polynomial-time algorithm to solve this problem if we assume that P is not NP. Later in this chapter and next, we will cover a more efficient optimization formulations that can capture a wide-range of specification types with faster runtimes.

Definition 3.10 (Sequence Path)**.** Given a graph G and atomic propositions characterizing the sequence specification, $p_1, \ldots, p_n$, where $Path_G(v_i, v_{i+1})$ represent a simple path from v_i to v_{i+1}, for all $i = 1, \ldots, n$. The *sequence path* from v_1 to v_{n+1} can be constructed from the individual path segments as $Path_G(v_1, v_{n+1}) = Path_G(v_1, v_2), \ldots, Path_G(v_n, v_{n+1})$. The sequence path $Path_G(v_1, v_{n+1})$ is *valid* if it does not have a cycle involving two or more path segments. That is, if there are no edges $(u, w), (w, v) \in E$ such that edge $(u, w) \in Path_G(v_i, v_{i+1})$ and $(w, v) \in Path_G(v_j, v_{j+1})$ for some $i + 1 \leq j \leq n + 1$, except for the case in which both $w = v_{i+1}$ and $j = i + 1$. In other words, except for the sequence nodes $v_1 \ldots, v_n$ that link individual segments, there are no common nodes linking an earlier path segment to a later segment. Observe that existence of a valid sequence path implies that Assumption 3.2 is true.

Finding Combinations of Augmenting Paths

Maximum flow realizations on a graph need not be unique; there can exist more than one set of augmenting paths to capture maximum flow between a source and target. For some graph G, we will denote the maximum flow from source s to target t as $F_{max_G}(s, t)$ and a set of augmenting paths by $\texttt{AP}_G(s, t)$ or $\texttt{SAP}_G(s, t)$ for the set of shortest augmenting paths. Let $\mathcal{AP}_G(s, t)$ correspondingly denote the *set of sets of augmenting paths*, and let $\mathcal{SAP}_G(s, t)$ correspondingly denote the *set of sets of shortest augmenting paths*. Note that $\mathcal{AP}_G(s, t)$ captures all possible realizations of maximum flow from s to t on G, and $\mathcal{SAP}_G(s, t) \subseteq \mathcal{AP}_G(s, t)$. Intuitively, since the shortest path need not be unique, the set $\mathcal{SAP}_G(s, t)$ could have multiple elements.

By definition, on a test graph G_{cut}, the maximum sequence flow value will be bounded as follows:

$$f_{G_{cut}}(v_1, v_{n+1}) \leq \min_{i=1,\ldots,n} f_{G_{cut}}(v_i, v_{i+1}). \tag{3.12}$$

On a minimally restricted test graph, the total sequence flow $f_{G_{cut}}(v_1, v_{n+1})$ is maximized. For each $1 \leq i \leq n$, note that $\mathcal{SAP}_G(v_i, v_{i+1})$ is finite since the number of edges in G are finite, but is combinatorial in the number of edges since it requires enumerating simple paths from v_i to v_{i+1}. The total number of augmenting path combinations from v_1 to v_{n+1} that can realize the maximum flow $f_{G_{cut}}(v_1, v_{n+1})$ will be at most $\Pi_{i=1}^{n}|\mathcal{AP}_G(v_i, v_{i+1})|$. Not every augmenting path combination might lead to a valid test graph since there could exist a combination of augmenting paths that violates equation (5.2) by resulting in an invalid sequence path $Path(v_1, v_n)$. Consider the simple example of the 3×3 grid in Figure 3.4. The combination of sequence flows $(\mathtt{AP}_G(v_1, v_2), \mathtt{AP}_G^2(v_2, v_3), \mathtt{AP}_G(v_3, v_4))$ will give us $f_{G_{cut}}(v_1, v_{n+1}) = 1$, but the combination of $(\mathtt{AP}_G(v_1, v_2), \mathtt{AP}_G^1(v_2, v_3), \mathtt{AP}_G(v_3, v_4))$ does not have any valid sequence paths.

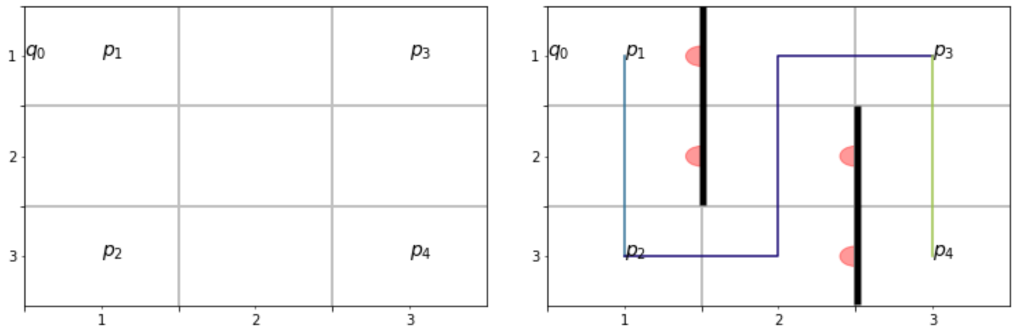


Figure 3.3: Left: 3×3 grid for the sequence specification with atomic proposition p_1, p_2, and p_3. Right: Illustrated with cuts that route the flow from p_1 to p_4.

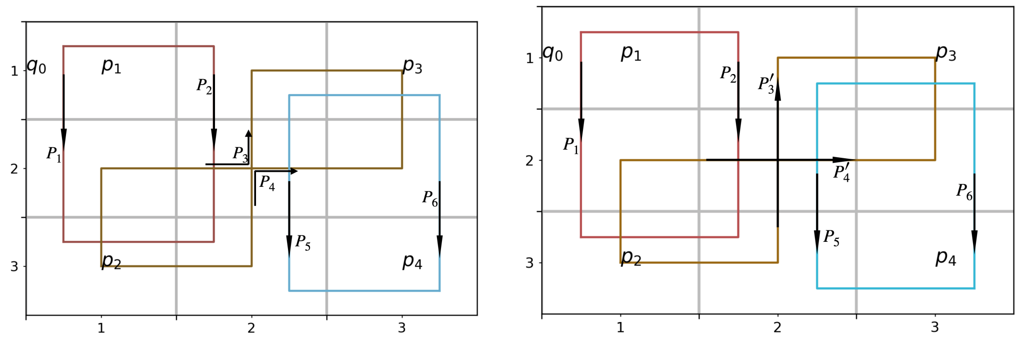


Figure 3.4: In this 3×3 grid, the left and right figures illustrate two different augmenting path combinations. Each grid shows a realization of the flow for each pair of nodes: red: (p_1, p_2), gold: (p_2, p_3), and blue: (p_3, p_4). The main difference between the two figures is in the flow from p_2 to p_3. In both figures, the augmenting paths characterizing the flow from p_1 to p_2 and p_3 to p_4 are the same: $\mathtt{AP}_G(v_1, v_2) = \{P_1, P_2\}$ characterizes the maximum flow from p_1 to p_2, and $\mathtt{AP}_G(v_3, v_4) = \{P_5, P_6\}$ characterizes the maximum flow from p_3 to p_4. On the left, $\mathtt{AP}_G^1(v_2, v_3) = \{P_3, P_4\}$, and on the right, $\mathtt{AP}_G^2(v_2, v_3) = \{P_3', P_4'\}$. It is possible to form a sequence path on the right with $Path_G(v_1, v_4) = P_1, P_3', P_6$, but not on the left. This sequence path is exactly illustrated in Fig. 3.3 (right).

Algorithm 4: Restrict Transitions

```
Input: $\varphi_{\text{test}}, \varphi_a, G = (V, E, L)$.
Output: $C \subseteq E$.
  $p \leftarrow \{p_1, \ldots, p_n, p_{n+1}\}, V_p \leftarrow \{v_1, \ldots, v_n, v_{n+1}\}$    ▷ $v_i \vdash p_i$
  $P_{cut} \leftarrow$Find-Cut-Paths($G, p$)
  $C = \{\}$
  if Assumption 3.3 then
      $flg \leftarrow 1$
  while $P_{cut} \neq \emptyset$ do    ▷ Repeat until all cuts are found
      $E \leftarrow$ Edges in $P_{cut}$
      $\mathcal{A}, \mathcal{P}_{keep}, |\mathcal{A}|, F_{max} \leftarrow$ Sequence-Flows$(G, p, flg = 0)$    ▷ Combinations of sequence flows
      for all $j = 0, \ldots, |\mathcal{F}|$ do
          $\mathcal{A}_f \leftarrow \mathcal{A}(j)$    ▷ Selecting a combination $(S_1, \ldots, S_n)$
          $P_{keep} \leftarrow \mathcal{P}_{keep}(j)$    ▷ Augmenting paths for each $v_i$ to $v_{i+1}$
          $MC_{keep} \leftarrow$ Min-Cut-Edges$(G, p, P_{keep})$
          $D_{keep} \leftarrow diag(A_{keep}\mathbb{1})$
          for all $A_f \in \mathcal{A}_f$ do
              $D_f \leftarrow diag(A_f\mathbb{1})$
              $A_{cut}, A_{keep}, D_{keep} \leftarrow$ ILP-params$(P_{cut}, P_{keep}, MC_{keep})$
              $x^*, f^*, b^* \leftarrow$ ILP$(A_{cut}, A_{keep}, D_{keep}, A_f, D_f)$    ▷ Call to ILP (3.10)
              if $\mathbb{1}^T f^* = F_{max}$ then
                  $C_{new} \leftarrow \{e_i | x_i^* = 1\}$
                  $C \leftarrow C \cup C_{new}$
                  break    ▷ Breaking out of both for loops
      $G \leftarrow G \backslash C_{new}$
      $P_{cut} \leftarrow$ Find-Cut-Paths($G, p$)
```

To avoid this issue, the algorithm searches through all combinations of sequence flows before constructing the input to the ILP (3.10). Since this is an expensive computation, a further assumption on the input graph and set of propositions can ease this bottleneck.

Assumption 3.3. Let f_i be the maximum flow on G from source v_i to target v_{i+1})/ Let $\mathcal{SAP}_G(v_i, v_{i+1}) = \{\mathtt{SAP}_G^1 = \{P_1, \ldots, P_{f_i}\}\}$, represent the set of sets of shortest augmenting paths that characterizes the flow from v_i to v_{i+1} on G. Then, there exists a combination $(\mathtt{SAP}_G^1, \ldots, \mathtt{SAP}_G^n)$ on which a maximum sequence flow can be characterized.

In other words, Assumption 3.3 allows us to reason over combinations of shortest augmenting path flows, which is combinatorial in all shortest paths, instead of combinations of the set of augmenting flows, which exacerbates the combinatorial complexity by enumerating all possible paths. All shortest paths are a subset of all simple paths between two nodes.

3.7 Iterative Synthesis of Constraints

Algorithm 4 details how edge cuts are computed by iteratively solving the following integer linear program.

$$
\begin{aligned}
\max_{\substack{x\in\mathbb{B}^n,\, f\in\mathbb{B}^l\\ b\in\mathbb{B}^m}} \quad & \mathbb{1}^T f \\
\text{s.t.} \quad & \mathbb{1} \le A_{cut}x \\
& A_{keep}x \le D_{keep}b \\
& b \le A_{keep}x \\
& D_f f \le A_f(\mathbb{1} - b)\,, \\
& A_f(\mathbb{1} - b) - D_f\mathbb{1} + \mathbb{1} \le f,
\end{aligned}
\tag{3.13}
$$

where (x, b, f) are the optimization variables, and $A_{cut} \in \mathbb{B}^{k\times n}$, $A_{keep} \in \mathbb{B}^{m\times n}$, $D_{keep} \in \mathbb{B}^{m\times m}$, $D_f \in \mathbb{B}^{l\times l}$, $A_f \in \mathbb{B}^{l\times m}$ are problem data described in more detail below.

Variables: The variable $x \in \mathbb{B}^n$, where $n = |E_{cut}|$, is the Boolean vector corresponding to edges E_{cut} such that for some $k \le n$, if $x_k = 1$, then the corresponding edge is restricted, and $x_k = 0$ means that it is left in the graph for future iterations. Given $P_{keep} = (\texttt{SAP}_G^1, \ldots, \texttt{SAP}_G^n) \in \mathcal{P}_{keep}$, a combination of set of shortest augmenting paths, the variable $b \in \mathbb{B}^m$ keeps track of whether an augmenting path in some $\texttt{SAP}_G^i$ $(1 \le i \le n)$ is restricted or not.

For some $k \le m$, if $b_k = 1$, then the corresponding augmenting path in some $\texttt{SAP}_G^i$ has minimum-cut edge(s) restricted by the ILP, and $b_k = 0$ if none of the minimum-cut edges of that augmented path have been restricted. The variable $f \in \mathbb{B}^l$ is the sequence flow vector for a given sequence flow, S_f, such that $l = |S_f|$ is the number of edge-disjoint paths constituting the sequence flow.

Constraints: The first constraint of the ILP, $A_{cut}x \ge \mathbb{1}$, enforces the requirement that each path in $P \in P_{cut}$ is restricted. Each row of A_{cut} corresponds to a path $P \in P_{cut}$. The q-th row of A_{cut} is constructed as follows:

$$
(A_{cut})_{q,r} = \begin{cases} 1 & \text{if } E_{cut}(r) \in P = P_{cut}(q) \\ 0 & \text{otherwise.} \end{cases} \tag{3.14}
$$

In the second and third constraints, $A_{keep}x \leq D_{keep}b$ and $b \leq A_{keep}x$, is used to determine the variable b from the variable x. Each row of $A_{keep} \in \mathbb{B}^{m\times n}$ corresponds to some path $P \in \mathtt{SAP}^i_G$, and $D_{keep} \in \mathbb{B}^{m\times m}$ is a diagonal matrix. Suppose the q-th row of A_{keep} corresponds to a path $P \in \mathtt{SAP}^i_G$ for $P_{keep} = (\mathtt{SAP}^1_G, \ldots, \mathtt{SAP}^n_G)$, and $MC_{keep}(i)$ is the set of minimum-cut edges on some path in $\mathtt{SAP}^i_G$, then the q-th row is constructed as follows:

$$(A_{keep})_{q,r} := \begin{cases} 1, & \text{if } E_{cut}(r) \in P \cap MC_{keep}(i). \\ 0, & \text{otherwise.} \end{cases} \tag{3.15}$$

The q-th diagonal entry of D_{keep} stores the total number of minimum-cut edges in the path corresponding to the q-th row of A_{keep}:

$$D_{keep} := diag(A_{keep}\mathbb{1}) \tag{3.16}$$

These two constraints ensure that for some $q \leq n$, $b_q = 1$ iff at least one minimum-cut edge on the path corresponding to the q-th row of A_{keep} is restricted, and $b_q = 0$ iff none of the minimum-cut edges on the path corresponding to the q-th row of A_{keep} are restricted.

The fourth and fifth constraints, $D_f f \leq A_f(\mathbb{1} - b)$ and $f \geq A_f(\mathbb{1} - b) - D_f\mathbb{1} + \mathbb{1}$, determine the flow value for a given set of sequence flow paths, S_f. Suppose the q-th row of the matrix $A_f \in \mathbb{B}^{l\times m}$ corresponds to some sequence flow path $P = (P_1, \ldots, P_n) \in S_f$. Let $R = (r_1, \ldots, r_n)$ denote the indices of the paths $P_1, \ldots, P_n$ according to the ordering of the paths constituting all $\mathtt{SAP}^i_G$ that is consistent with the construction of A_{keep} and D_{keep}. Then, the q-th row of A_f is defined as follows:

$$(A_f)_{q,r} := \begin{cases} 1, & \text{if } r = r_i \text{ for some } 1 \leq i \leq n. \\ 0, & \text{otherwise.} \end{cases} \tag{3.17}$$

The q-th diagonal entry of matrix $D_f \in \mathbb{B}^{l\times l}$ stores the total number of ones in the q-th row of A_f:

$$D_f := diag(A_f\mathbb{1}). \tag{3.18}$$

The fourth constraint ensures that if any of the constituent paths, $P_1, \ldots, P_n$, in the q-th sequence flow path $P = (P_1, \ldots, P_n) \in S_f$ (for $1 \leq q \leq l$), is restricted, then the flow value, $f_q = 0$. The last constraint ensures that if none of the constituent paths, $P_1, \ldots, P_n$, in the q-th sequence flow path $P = (P_1, \ldots, P_n) \in S_f$ (for $1 \leq q \leq l$), are restricted, then the flow value, $f_q = 1$.

Parameters: The parameters used to construct the problem data for the ILP (3.13) are the set of paths that need to be restricted, P_{cut}, the set of paths whose combination constitutes sequence flow and should not be restricted, P_{keep}, and the set of minimum-cut edges, MC_{keep}, on the paths constituting P_{keep}. The set $\mathcal{P}_{keep} = \{(\mathtt{SAP}^1_G, \ldots, \mathtt{SAP}^n_G) \mid \mathtt{SAP}^i_G \in \mathcal{SAP}^i_G\}$ is a set of all combinations of shortest augmenting paths in the sequence. For a given combination of sets of augmenting paths, $P_{keep} = (\mathtt{SAP}^1_G, \ldots, \mathtt{SAP}^n_G)$, with the cardinality of $\mathtt{SAP}^i_G$ being denoted as follows: $k_i := |\mathtt{SAP}^i_G|$, and $m := \Sigma_{i=1}^n k_i$. In P_{keep}, suppose a combination of augmenting paths, $S_f = \{P = (P_1, \ldots, P_n) \mid P_i \in \mathtt{SAP}^i_G\}$, represents a sequence flow, then a matrix $A_f \in \mathbb{B}^{|S_f| \times m}$ can be constructed to represent the sequence flow S_f. This construction is outlined in the descriptions of Constraints of the ILP. An instance of P_{keep} can have several sequence flows, S_f, and correspondingly, several matrices, A_f, all of which are collectively denoted by $\mathcal{A}_f$. The set of all such $\mathcal{A}_f$ is denoted by $\mathcal{A}$, which has cardinality $|\mathcal{A}| = |\mathcal{P}_{keep}|$, since each $\mathcal{A}_f$ corresponds to an instance of P_{keep}. The maximum sequence flow value is given by $F_{\max}$.

Cost Function: The cost function computes the maximum sequence flow value. Algorithm 4 does not proceed to the next iteration of P_{cut} until it finds the set of static constraints that return the maximum possible sequence flow value, F_{max}. To guarantee completeness of Algorithm 4, we need to prove that the cuts synthesized in prior iterations do not preclude feasibility of further iterations with regards to assumption 3.2. See Section 3.8 for complexity of the subroutines in Algorithm 4.

Sub-routines of Algorithm 4

The MIN-CUT-EDGES sub-routine takes as input a graph G, a list of propositions $\{p_1, \ldots, p_n\}$, and for each $1 \leq i \leq n$, a non-empty set of shortest augmenting paths for the source-sink pair (v_i, v_{i+1}). This sub-routine returns as output the set of minimum cut-set on those augmenting paths, which is then used in constructing the problem data for the ILP.

The SEQUENCE-FLOWS sub-routine takes as input a graph G, a list of propositions $\{p_1, \ldots, p_n\}$, and a parameter to indicate if Assumption 3.3 holds. It then computes the combination of all augmenting flows (or all shortest augmenting flows) that can result in a non-zero sequence flow from v_1 to v_{n+1}. It returns as output the set of all sets of matrices that capture sequence-flow paths, $\mathcal{A}$, a set of $\mathcal{P}_{keep} = \{(\mathtt{SAP}^1_G, \ldots, \mathtt{SAP}^n_G) | \mathtt{SAP}^i_G \in \mathcal{SAP}^i_G\}$, the total number of combinations, $|\mathcal{A}|$, and the maximum possible sequence flow value, F_{max}, which is determined when $\mathcal{A}$ is constructed.

The sub-routine FIND-BYPASS-PATHS takes as input a graph G and list of propositions, $\{p_1, \ldots, p_n\}$, and uses the Edmonds-Karp algorithm to find bypass paths for every source-sink pair (v_i, v_j), where $i + 1 \le j \le n + 1$. Specifically, this sub-routine finds a set of shortest augmenting paths from v_i to v_j, and to ensure that they are bypass paths, the sub-routine is applied on $G_{ij} = G = (V \setminus V_k, E \setminus E(V_k))$, where:

$$V_k \coloneqq \{v_k \vDash p_k \mid 1 \le k \le n+1 \text{ and } k \ne i, k \ne j\},$$

and the edges associated with V_k are denoted by E_k:

$$E_k \coloneqq \{(u, v) \in E \mid u \in V_k \text{ or } v \in V_k\}.$$

All of these augmenting paths are collectively returned as the output P_{cut}, and the edges constituting these cuts are denoted by E_{cut}. Note that P_{cut} does not return all simple paths from v_i to $v_{j>i+1}$, but just a set of edge-disjoint paths. As a result, transitions are iteratively restricted until P_{cut} is empty.

3.8 Characteristics of the Algorithm

Lemma 3.1. In a graph $G = (V, E)$, let $\mathcal{P}$ represent a maximal set of sequence flow paths from v_1 to v_n. Let $\mathcal{P}_{cut}$ be the set of paths that need to restricted, with the edges constituting the paths in $\mathcal{P}_{cut}$ denoted by $E_{cut} \subset E$. Then, the set of constraint edges $C \subseteq E_{cut}$ can be found such that C does not constrain any path in $\mathcal{P}$.

Proof. A path $P_{cut} \in \mathcal{P}_{cut}$ can be restricted by removing at least one of its constituent edges. The number of edges of P_{cut} that are not in some path $P \in \mathcal{P}$ is non-zero, since otherwise it would imply that $P_{cut,i} \in \mathcal{P}$, and would not need to be restricted. The set C can simply be chosen by selecting one or more edges on every $P_{cut} \in \mathcal{P}_{cut}$ that are not a part of some path in $\mathcal{P}$. $\square$

Proposition 3.2. Let $G_m = (V, E_m)$ denote the graph for which the m-th iteration of the ILP (3.13) synthesizes new cuts $C_m \subset E_m$. Then, Assumption 3.2 is satisfied on $G_{m+1} = (V, E_m \backslash C_m)$.

Proof. In the first iteration, from Assumption 3.2, we know there exists at least one test graph $G' = (V, E \backslash C)$ that satisfies equation (3.11). Assume that the m-th iteration graph $G_m = (V, E_m)$ also satisfies Assumption 3.2. We will show by induction that the graph resulting from the the $(m+1)$-th iteration, $G_{m+1} = (V, E_m \backslash C_m)$, also satisfies Assumption 3.2. By construction, Algorithm 4 chooses a combination of set of shortest augmenting paths $(\mathtt{SAP}^1_G, \ldots, \mathtt{SAP}^n_G)$, such that there exists a non-empty set of sequence flow paths $\mathcal{F} = \{(P_1, \ldots, P_n) | \, P_i \in \mathtt{SAP}^i_G\}$ such that the simple path from v_1 to v_n characterized by $\Gamma = (P_1, \ldots, P_n) \in \mathcal{F}$ does not form an ij-cycle for some $i < j \leq n$. This implies that on the subgraph comprising of the edges in Γ, equation (3.11) is satisfied.

If the maximum possible sequence flow in a minimally restricted test graph is $F_{\max}$, then we can find a combination $(\mathtt{SAP}^1_G, \ldots, \mathtt{SAP}^n_G)$ such that for each $i = 1, \ldots, n$, there exists a subset of edge-disjoint paths carrying flow $F_{\max}$:

$$\mathtt{S}_i = \{P_i^1, \ldots, P_i^{F_{\max}}\} \subseteq \mathtt{SAP}^i_G,$$

from which we can construct the set of sequence flow paths:

$$\mathcal{F}' = \{(P_1^{k_1}, \ldots, P_n^{k_n}) | P_i^{k_i} \in \mathtt{S}_i, \, 0 \leq k_i \leq F_{\max}\} \subseteq \mathcal{F}.$$

By construction of the input variables to the ILP (3.13), the constraints of ILP (3.13) require that the sequence flow variable f has atleast one element that is 1. This is possible only if there exists a set of edges C_m that constrain $A_{cut,m}$ such that there exists at least one sequence path $P \in \mathcal{F}$ that does not have any of its minimum-cut edges restricted, which is true as shown in Lemma (3.1). Therefore, the new graph $G_{m+1} = (V, E_m \backslash C_m)$ satisfies Assumption (3.2). □ □

Theorem 3.1. Under Assumption (3.2), Algorithm 4 is complete and returns a test graph G' from Definition 3.8 that satisfies equation (3.11).

Proof. Consider iteration m of the outer while loop in Algorithm 4, and let the graph at the m-th iteration be $G_m = (V, E_m)$. Denote $V_p = \{v_i | v_i \vdash p_i, \forall 1 \leq i \leq n+1\}$. Let $F^{i,j}_{\max}$ denote the maximum flow value from v_i to v_j on $G_{ij} = (V \backslash (V_p \backslash \{v_i, v_j\}), E_m)$, for some i, j such that $1 \leq i < j - 1 \leq n$. That is, G_{ij} is a copy of G_m, but with nodes in V_p, except for source v_i and sink v_j, removed.

This implies that there is a set $\mathtt{SAP}^{i,j}_{G_{ij}}$ of $F^{i,j}_{\max}$ edge-disjoint paths that characterize the maximum flow from v_i to v_j on G_{ij}. Let $(\mathcal{P}_{i,j}(k))_m$ be the set of all simple paths from v_i to v_j that share an edge with the k-th path in $\mathtt{SAP}^{i,j}_{G_{ij}}$. Let $(MC_{i,j})_m$ be the set of minimum-cut edges on the paths in $\mathtt{SAP}^{i,j}_{G_{ij}}$ and let $(E_{i,j})_m \subset E_m$ be the set of all edges on some path from v_i to v_j on G_{ij}. Clearly, $(MC_{i,j})_m \subseteq (E_{i,j})_m$.

For every $m \geq 1$, we can claim that $|(E_{i,j})_{m+1}| < |(E_{i,j})_m|$ because edges are removed to constrain $\texttt{SAP}^{i,j}_{G_{ij}}$ in the m-th iteration. Let $\tilde{m}$ be the number of iterations for G_{ij} to become disjoint. In the worst-case, edges continue to be restricted until iteration $\tilde{m}$ at which $(E_{i,j})_{\tilde{m}} = (MC_{i,j})_{\tilde{m}}$, at which point constraining edges to cut $(\texttt{SAP}^{i,j}_{G_{ij}})_{\tilde{m}}$ results in a cut separating v_i and v_j. Thus, $\tilde{m}$ has to be finite for every such i, j.

At the same time, from Proposition 3.2, the synthesized cuts are such that Assumption 3.2 is maintained as an invariant. Therefore, when the last set of paths $\texttt{SAP}^{i,j}_{G_{ij}}$ are restricted, the final test graph G' is such that $d_{G'}(v_1, v_{n+1}) > \ldots > d_{G'}(v_n, v_{n+1})$. □

In addition to Assumption 3.2, if Assumption 3.3 holds, Algorithm 4 can be modified by a parameter setting. The proof of Theorem 3.1 still holds.

Lemma 3.2. On the test graph G', any test run σ starting from state v_1 will satisfy the specification ((5.2)).

Proof. From Assumption 3.1, there is only one node in G' for each proposition in characterizing the test specification ((5.2)), and node satisfying proposition p_i is labeled as v_i. For every $i \in \{1, \cdots, n\}$, v_i is the only state in test graph G' that is successor to all states v on paths $Paths(v_{j<i}, v_{n+1})$ for which $d_{G'}(v, v_{n+1}) = d_{G'}(v_i, v_{n+1}) + 1$. This is true by construction of the ILP constraints. All paths in the set $Paths(p_{j<i}, g)$ on the test graph G' must pass through v_i.

Let σ denote the test run of the agent starting at v_1. We define a metric on the test graph G': $m_t := \min_t d_{G'}(\sigma_t, v_{n+1})$ to be the closest distance to node v_{n+1} in the first t steps of the test run. Note three properties of this metric m_t: (a) $m_t \geq 0$, (b) m_t decreases: $m_{t+1} := \min\{\sigma_{t+1}, m_t\} \leq m_t$, and (c) there exists a successor q_{t+1} to $\sigma_t = q_t$ on G' such that $d_{G'}(q_{t+1}, v_{n+1}) = d_{G'}(q_t, v_{n+1}) - 1$ that decreases m_t. The metric m_t starts at $m_0 \geq d_{G'}(v_1, v_{n+1})$ and decreases to 0 at the end of the test run. Thus, we can observe that $\sigma \models \Diamond(p_1 \wedge \Diamond(p_2 \cdots \wedge \Diamond\, p_{n+1})) \wedge_{i=1}^{n} (\neg p_{i+1} \mathsf{U}\, p_i) \iff \sigma \models \Diamond\, v_{n+1}$. □ □

From Theorem 3.1 and Lemma 3.2, Algorithm 4 synthesizes a test graph G' for the test specification (5.2), solving Problem 3.1.

Lemma 3.3. Consider the test graph G' from Definition 3.8 for the test specification σ from (5.2). If Assumption 3.2 holds, Algorithm 4 returns a minimally restricted test graph.

Proof. By construction, the inputs to the ILP (3.13) are constructed based on a maximal set of sequence flow paths from v_1 to v_n. By Lemma 3.1, at each iteration of the ILP (3.13) from which constraint edges are chosen, the maximum sequence flow value does not decrease at each iteration. Since there are a finite number of edges, there are a finite number of iterations until test graph is found. Therefore, the Algorithm 4 returns a minimally restricted test graph. □ □

Complexity of Subroutines in Algorithm 4

Since Find-Cut-Paths is determining a set of augmenting paths for a single source-sink flow, it has a complexity of Edmonds-Karp algorithm, $O(|V||E|^2)$ time for graph $G = (V, E)$ [79]. The complexity of Min-cut-Edges is $O(|V||E|^3)$ time since it runs a max-flow algorithm for each edge in the worst-case. The main computational bottleneck is in the Sequence-Flows subroutine, which constructs sets of augmenting flows by computing combinations of all simple paths and all shortest paths. In the worst-case, enumerating all simple paths between two nodes is $O|V!|$, and enumerating all shortest paths is slightly better in several cases.

3.9 Examples

We illustrate the iterative synthesis of restrictions on a simple graph and a small gridworld, and then show runtimes of Algorithm 4 on random gridworld instances for both the case for which Assumption 3.2 is true, and the case for which Assumptions 3.2 and 3.3 are true.

Simple graph: Consider a simple non-deterministic Kripke structure representing an autonomous agent, shown in Figure 3.5, with propositional formulas labeled adjoining the states. The agent mission objective is to reach g while being restricted to start from state q_0. The test environment seeks to restrict transitions such that the agent is prompted to pass through waypoint w in its trajectory to g.

Inputs to Algorithm 4 include the labeled graph G induced by the Kripke structure, the agent specification $\diamondsuit p_3$, the test specification $\diamondsuit p_2$, and the initial condition constraint $\diamondsuit p_1$. Algorithm 4 constrains the edges $\{(v_2, v_4), (v_4, v_6)\}$ in the first iteration, and the edges $\{(v_2, v_5), (v_5, v_6)\}$ in the second iteration. Although in this simple example, searching the set of all augmented paths becomes searching over all paths, in larger examples discussed below, each augmented path represents a class of paths that share some edge(s) with it.

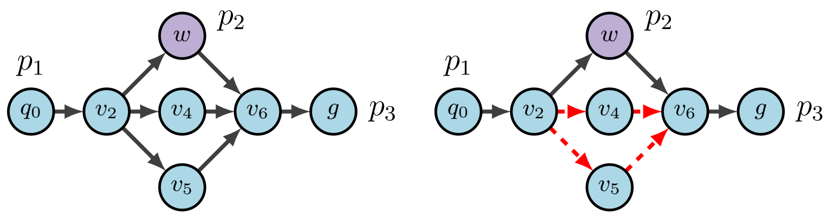


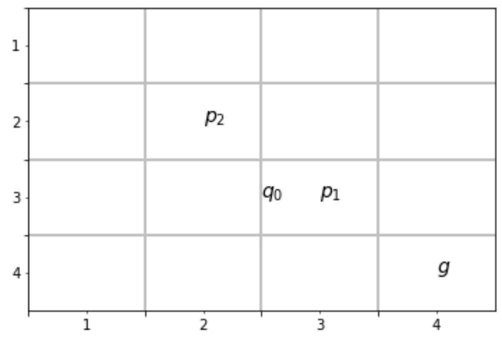
Figure 3.5: *Left:* Simple Kripke structure representing states that the agent can occupy. The waypoint, w, is highlighted in purple to indicate that transitions are restricted corresponding to propositional formula $p_2 = L(w)$. *Right:* A test graph. Dashed edges in red illustrate transitions that have been restricted/removed from the Kripke structure above.

Simple Gridworld: In Figure 3.6, we illustrate the iterative synthesis of obstacles in a gridworld instance. Note that this configuration can be synthesized only by considering all sets of augmenting paths between (p_1, p_2) and (p_2, p_3). Since there is no shortest augmenting path from p_2 to p_3 that does not form a cycle with some (in this example, there is only one) shortest augmenting path flow from p_1 to p_2, it is imperative to use all sets of augmenting paths in the Sequence-Flows subroutine.

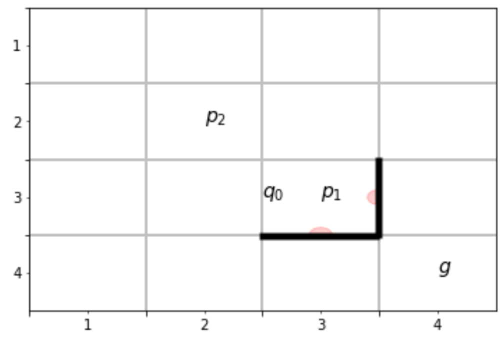
Random Gridworld Instances: For the case of setting all augmenting paths in the Sequence-Flows subroutine, we ran 50 random instances each for small gridworlds and propositions and plotted the average runtimes in Figure 3.7a. The number of propositions are limited by the size of the gridworld instances, which is restricted by the combinatorial nature of finding all sets of augmenting paths, and all combinations of sets of augmenting paths.

If we choose initial gridworld instances that satisfy Assumption 3.3, then Algorithm 4 can synthesize static constraints for slightly larger $t \times t$ grid sizes. The average runtimes for 50 random iterations for various grid sizes t is plotted in Figure 3.7b. The small increase to larger grid size is due to the Sequence-Flows subroutine reasoning over shortest augmenting paths, and not all augmenting paths.

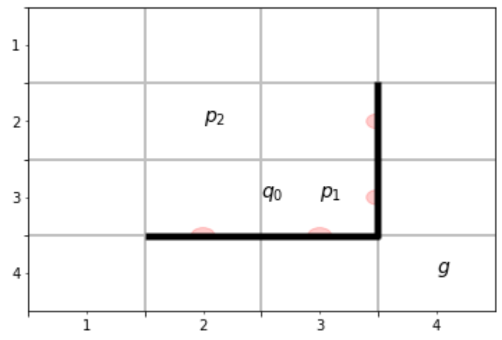
The average runtimes increase exponentially with the size of the grid. The number of propositions, denoted by $|P|$, is labeled n if the test specification φ_{test} (5.2) is comprised of propositions $(p_1, \ldots, p_n)$. In both Figures 3.7a and 3.7b, the average runtime for fewer propositions is at times higher that the average runtime for more propositions. This can be attributed to the Sequence-Flows subroutine



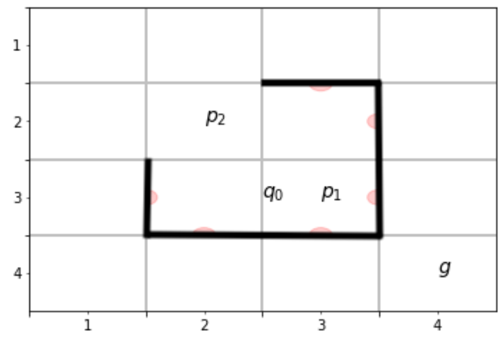
(a) Initial grid



(b) Iteration 1



(c) Iteration 2



(d) Iteration 3. The colored paths highlight a sequence flow from p_1 to g.

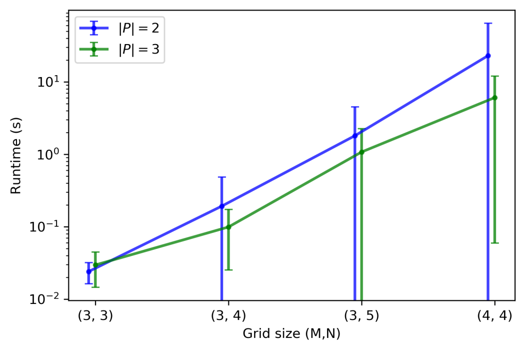
Figure 3.6: Synthesizing static test environment for $\varphi_{test} = \Diamond(p_1 \wedge \Diamond p_2) \wedge \neg p_2 \mathsf{U}\, p_1$ and $\varphi_a = \Diamond g$.

taking longer to enumerate all simple paths (or all shortest paths in case of Assumption 3.3) between two nodes, which could be greater in number due to fewer propositions constraining the graph.

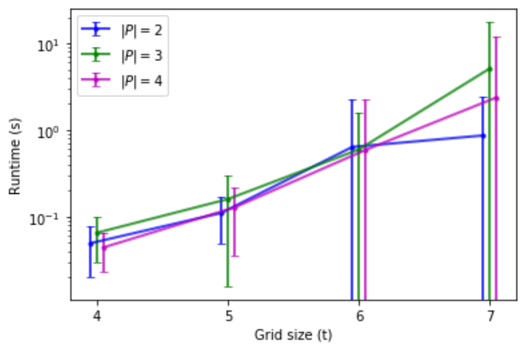
Another paradigm for the problem of synthesizing static test environments for sequence behaviors could be multi-commodity network flows, which will be explored later in this chapter. The multi-commodity flow setting typically considers multiple source-sink flows simultaneously drawing from the capacity of each edge, and here we compute separate network flows for every source-sink pair of nodes.

3.10 Conclusions

An algorithm to synthesize a static test environment to observe sequence-like behavior in a discrete-transition system was introduced. First, we formulated this test environment synthesis problem as a problem of synthesizing cuts on graphs using concepts of flow networks. Then, we proposed an algorithm which synthesized the cuts iteratively using an integer linear program. We proved that this algorithm is



(a) Small gridworld configurations using all augmenting flows.



(b) Gridworld configurations using only shortest augmenting flows.

Figure 3.7: Average runtime over 50 random instances. The number of propositions in φ_{test} is denoted by $|P|$ in the legend. Error bars represent standard deviation of runtimes.

complete and that the edges restricted by the ILP at each iteration maintain feasibility of the constraint in the next iteration. Finally, we conducted numerical experiments on random gridworld instances to assess the runtime of our algorithm. Simulation results preclude this algorithm from being tractable to larger examples.

However, the integer linear program requires reasoning over all possible paths on the transition system in order to identify the set of augmenting paths with the highest flow. This essentially becomes a brute-force approach to finding a set of edge disjoint paths from S to T that are routed through the propositions $p_1, \ldots, p_n$ in a sequence. The poor scalability is due to the exponential number of constraints (in the number of edges of T_{sys}) in the ILP formulation. To alleviate this, we will present an alternative flow-based formulation in the form of a min-max game with coupled constraints.

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.