Chapter 5

ASSUME-GUARANTEE CONTRACTS FOR COMPOSITIONAL TESTING

5.1 Introduction

The previous chapters discuss the synthesis of test strategies from system and test objectives. In this chapter, we will motivate the idea of compositional test plans. It might be desireable to construct a more complex test objective from simpler unit tests, or to break-down a complex test into simpler tests, either by testing on smaller sub-systems or conducting simpler tests. This chapter is a step in the direction towards composable test plans. First, we will introduce a mathematical and algorithmic framework in which simpler test objectives can be merged to form a more complex test objective. Then, we will introduce mathematical frameworks based in assume-guarantee contract operations to formally describe test campaigns, and how tests can be merged or decomposed. For simplicity, we assume that the test environment has already equipped to handle the test objectives.

This work is adapted from:

J.B Graebener*, A. Badithela*, R. M. Murray. (2022). "Towards Better Test Coverage: Merging Unit Tests for Autonomous Systems." In: *2022 NASA Formal Methods (NFM)*, pp. 133–155. DOI: 10.1007/978-3-031-06773-0_7.

A. Badithela*, J.B Graebener*, I. Incer* , R. M. Murray. (2023). "Reasoning over Test Specifications Using Assume-Guarantee Contracts." In: *2023 NASA Formal Methods (NFM)*, pp. 278–294. DOI: 10.1007/978-3-031-33170-1_17.

The contract-based-design framework was first introduced as a design methodology for modular software systems [115–117] and later extended to complex cyber-physical systems [118–120]. Following the definition of assume-guarantee contracts earlier in the chapter, we will now cover background on other contract operators [121]. For ease of reading, we will repeat the definition of assume-guarantee contracts below, and introduce other operators.

Definition 5.1 (Assume-Guarantee Contract)**.** Let $\mathcal{B}$ be a universe of behaviors, then a *component* M is a set of behaviors $M \subseteq \mathcal{B}$. A *contract* is the pair $\mathcal{C} = (A, G)$, where A are the assumptions and G are the guarantees. A component E is an *environment* of the contract $\mathcal{C}$ if $E \models A$. A component M is an *implementation* of the contract, $M \models \mathcal{C}$ if $M \subseteq G \cup \neg A$, meaning the component provides the specified guarantees if it operates in an environment that satisfies its assumptions. There exists a partial order of contracts, we say $\mathcal{C}_1$ is a refinement of $\mathcal{C}_2$, denoted $\mathcal{C}_1 \leq \mathcal{C}_2$, if $(A_2 \leq A_1)$ and $(G_1 \cup \neg A_1 \leq G_2 \cup \neg A_2)$. We say a contract $\mathcal{C} = (A, G)$ is in canonical, or saturated, form if $\neg A \subseteq G$.

$$\begin{tikzcd}
\mathcal{C} & & \mathcal{C}/\mathcal{C}' \\
\mathcal{C}' \parallel \mathcal{C}_1 \arrow[u] & \text{iff} & \mathcal{C}_1 \arrow[u]
\end{tikzcd}$$

(a) Composition and quotient.

$$\begin{tikzcd}
 & \mathcal{C}_1 \bullet \mathcal{C}_2 & \\
\mathcal{C}_1 & \mathcal{C}_1 \parallel \mathcal{C}_2 \arrow[u] & \mathcal{C}_2 \\
 & \mathcal{C}_1 \wedge \mathcal{C}_2 \arrow[u] \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

(b) Order of operations.

Figure 5.1: Contract operators and the partial order, defined in relation to the refinement operator, of their resulting objects.

Assume the following contracts are in canonical form. The meet or conjunction of two contracts exists [118] and is given by $\mathcal{C}_1 \wedge \mathcal{C}_2 = (A_1 \cup A_2, G_1 \cap G_2)$. Composition [122] yields the specification of a system given the specifications of the components: $\mathcal{C}_1 \parallel \mathcal{C}_2 = ((A_1 \cap A_2) \cup \neg(G_1 \cap G_2), G_1 \cap G_2)$. Given specifications $\mathcal{C}$ and $\mathcal{C}_1$, the quotient is the largest specification $\mathcal{C}_2$ such that $\mathcal{C}_1 \parallel \mathcal{C}_2 \leq \mathcal{C}$ [123]: $\mathcal{C}/\mathcal{C}_1 = (A \cup G_1, (G \cap A_1) \cup \neg(A \cup G_1))$. Strong merger [124] yields a specification that is satisfied by a system that also satisfies the two given specifications $\mathcal{C}_1$ and $\mathcal{C}_2$: $\mathcal{C}_1 \bullet \mathcal{C}_2 = (A_1 \cap A_2, (G_1 \cap G_2) \cup \neg(A_1 \cap A_2))$. The reciprocal (or mirror) [124, 125] is a unary operation which inverts assumptions and guarantees: $\mathcal{C}^{-1} = (G, A)$.

Remark 5.1. The statement contract $\mathcal{C}_1$ is more refined than contract $\mathcal{C}_2$ should be interpreted as follows. A system implementation for $\mathcal{C}_1$ has fewer assumptions on its environment and must provide more guarantees. In looking at sets of behaviors: the system must handle a larger set of environment behaviors while providing stricter guarantees. Since the guarantees of $\mathcal{C}_1$ are stricter than $\mathcal{C}_2$, the set of system behaviors satisfying guarantees of C_1 is smaller than that of C_2.

5.2 Preliminary Work on Merging Unit Tests

Recall the definition of a discrete-state system introduced in Chapter 4. In the previous chapter, the system specification could not be described as a GR(1) specification since the property that there will always exist a path to satisfying the system goal could not be characterized as a GR(1) formula. This is because the precise environment agents for a scenario were not given to the system; the system is only informed that actions in the test harness can be restricted, but these restrictions will always be such that the system has a feasible path. In this chapter, we assume that such a test environment has been constructed according to the test synthesis framework established in the previous chapter, and it is such that there always exists a path for the system goal. For simplicity, we consider a class of system objectives which can be written in the assume-guarantee form. The assumptions specify the dynamics of the test environment agents that will be used in the test scenario, and tests considered are such that even the worst-case dynamics of the test agents will not prevent the system from satisfying its requirements. Since this system objectives is a subset of the system objective considered in the previous chapter, we will used the term system specification instead. Let T_{sys} be the system transition system.

Definition 5.2 (System Specification)**.** A *system specification* φ_{sys} is the $GR(1)$ formula,

$$\varphi_{\text{sys}} = (\varphi_{\text{test}}^{\text{init}} \wedge \square\varphi_{\text{test}}^{s} \wedge \square\diamondsuit\, \varphi_{\text{test}}^{f}) \rightarrow (\varphi_{\text{sys}}^{\text{init}} \wedge \square\varphi_{\text{sys}}^{s} \wedge \square\diamondsuit\, \varphi_{\text{sys}}^{f})\,, \tag{5.1}$$

where $\varphi_{\text{sys}}^{\text{init}}$ is the initial condition that the system needs to satisfy, φ_{sys}^{s} encode system dynamics and safety requirements on the system, and φ_{sys}^{f} specifies recurrence goals for the system which is defined to be on a sink state of the system. Likewise, $\varphi_{\text{test}}^{\text{init}}$, $\varphi_{\text{test}}^{s}$, and $\varphi_{\text{test}}^{f}$ represent assumptions the system has on the test environment.

Once again, the objective is to synthesize a test strategy for the test environment given the test specification. Unlike the system specification, the infinitely often sub-task specification need not be restricted to be satisfied in a terminal state.

Definition 5.3 (Test Specification)**.** A *test specification* φ_{test} is the $GR(1)$ formula,

$$\varphi_{\text{test}} := (\varphi_{\text{sys}}^{\text{init}} \wedge \square\varphi_{\text{sys}}^{s} \wedge \square\diamondsuit\, \varphi_{\text{sys}}^{f}) \rightarrow (\varphi_{\text{test}}^{\text{init}} \wedge \square\varphi_{\text{test}}^{s} \wedge \square\diamondsuit\, \varphi_{\text{test}}^{f} \wedge \square\psi_{\text{test}}^{s} \wedge \square\diamondsuit\, \psi_{\text{test}}^{f})\,, \tag{5.2}$$

where $\varphi_{\text{sys}}^{\text{init}}$, φ_{sys}^{s} and φ_{sys}^{f}, $\varphi_{\text{test}}^{\text{init}}$, $\varphi_{\text{test}}^{s}$ and $\varphi_{\text{test}}^{f}$ are propositional formulas from equation (5.1). Additionally, $\square\psi_{\text{test}}^{s}$ and $\square\diamondsuit\, \psi_{\text{test}}^{f}$ describe the safety and recurrence formulas for the test environment in addition to the dynamics of the test environment known to the system. Note that the system is unaware of these additional sub-task specifications (similarly to the previous chapters), and the test environment is such that the system is allowed to satisfy its requirements. Defining the test specification in this manner allows for i) synthesizing a test in which the system, if properly designed, can meet φ_{sys}, and ii) specifying additional requirements on the test environment, unknown to the system at design time. We assume that test specifications are defined *a priori*; we leave automatically finding relevant test specifications to future work.

Having defined the system and test specifications, we define a product transition system that represents the turn-based dynamics of the two players: Let T_{prod} be a turn-based product transition system constructed from T_{sys} and T_{test}, where $T_{\text{prod}}.S := T_{\text{sys}}.S \times T_{\text{test}}.S$, $T_{\text{prod}}.A := T_{\text{sys}}.A \times T_{\text{test}}.A$, and $T_{\text{prod}}.\delta \subseteq T_{\text{prod}}.S \times T_{\text{prod}}.A \times T_{\text{prod}}.S$ denotes the turn-based transition function. In particular, for every transition $(s, a_s, s') \in T_{\text{sys}}.\delta$, we have $((s,t),(a_s,a_t),(s',t)) \in T_{\text{prod}}.\delta$ where $t \in T_{\text{test}}.S$ and $a_t \in T_{\text{test}}.A$. The transitions originating due to test agent actions are constructed similarly. From the product transition system, we can construct a game graph that maintains two copies of each state — one from which the system player acts and the other from which the test environment acts.

Definition 5.4 (Game Graph)**.** Let V_{sys} and V_{test} be copies of the states $T_{\text{prod}}.S$. Let E_{sys} and E_{test} correspond to the transitions in the game graph:

$$\begin{aligned} E_{\text{sys}} =& \{((s,t),(s',t)) \,|\, \exists a_s \in T_{\text{sys}}.A,\ \forall a_t \in T_{\text{test}}.A,\ ((s,t),(a_s,a_t),(s',t)) \in T_{\text{prod}}.\delta\}, \\ E_{\text{test}} =& \{((s,t),(s,t')) \,|\, \exists a_t \in T_{\text{test}}.A,\ \forall a_s \in T_{\text{sys}}.A,\ ((s,t),(a_s,a_t),(s,t')) \in T_{\text{prod}}.\delta\}. \end{aligned} \tag{5.3}$$

Then, the *game graph* is a directed graph $G = (V, E)$ is a directed graph with vertices $V := V_{\text{sys}} \cup V_{\text{test}}$ and edges $E := E_{\text{sys}} \cup E_{\text{test}}$.

On the game graph, a player strategy, and the test execution resulting from it are given below.

Definition 5.5 (Strategy)**.** On the game graph G, a policy for the system is a function $\pi_{\text{sys}} : V^* V_{\text{sys}} \to V_{\text{test}}$ such that $(s, \pi_{\text{sys}}(w.s)) \in E_{\text{sys}}$, where $s \in V_{\text{sys}}$ and $w \in V^*$. Similarly defined, π_{test} denotes the test environment policy, where * is the Kleene star operator.

Definition 5.6 (Test Execution)**.** A *test execution* $\sigma = v_0 v_1 v_2 \ldots$ starting from vertex $v_0 \in V$ is an infinite sequence of states on the game graph G. Since G is a turn-based game graph, the states in the test execution alternate between V_{sys} and

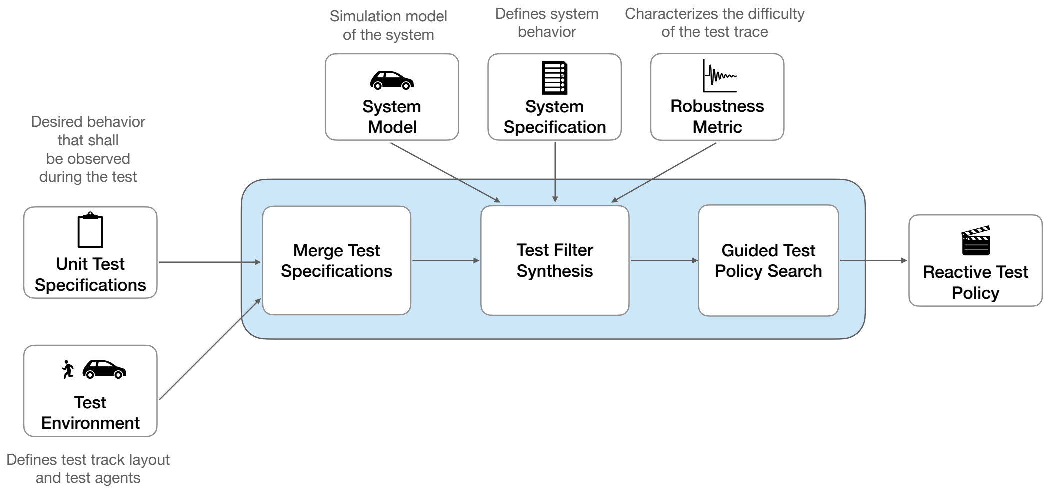

Simulation model of the system
System Model
Defines system behavior
System Specification
Characterizes the difficulty of the test trace
Robustness Metric
Desired behavior that shall be observed during the test
Unit Test Specifications
Merge Test Specifications
Test Filter Synthesis
Guided Test Policy Search
Reactive Test Policy
Test Environment
Defines test track layout and test agents

Figure 5.2: Overview of the merging unit tests.

V_{test}, so if $V_1 \in V_{\text{sys}}$, then $v_{i+1} = \pi_{\text{sys}}(v_0 \dots V_1)$. Let $\sigma_{s_0}(\pi_{\text{sys}} \times \pi_{\text{test}})$ be the test execution starting from state $s_0 \in V_{\text{sys}}$ for policies π_{sys} and π_{test}. Let Σ denote the set of all possible test executions on G. A robustness metric $\rho : \Sigma \to \mathbb{R}$ is a function evaluated assigning a scalar value to a test execution.

Problem 5.1. Given system and environment transition systems, $\mathcal{T}_{\text{sys}}$ and $\mathcal{T}_{\text{test}}$, two unit test objectives $\varphi_{\text{test},1}$ and $\varphi_{\text{test},2}$, and a robustness metric ρ, find a test strategy π^*_{test}, such that

$$\begin{aligned} \pi^*_{\text{test}} \quad &= \quad \underset{\pi_{\text{test}}}{\arg\max} \quad \rho(\sigma(\pi_{\text{sys}} \times \pi_{\text{test}})) \\ &\qquad\quad \text{s.t.} \quad \sigma(\pi_{\text{sys}} \times \pi_{\text{test}}) \models (\varphi_{\text{test},1} \wedge \varphi_{\text{test},2})\,, \quad \forall\, \pi_{\text{sys}} \models \varphi_{\text{sys}}. \end{aligned} \tag{5.4}$$

Example 5.1 (Running Example — Lane Change)**.** Consider the lane change scenario illustrated in Figure 5.3. The system (red car) is required to change lanes into the lower lane before the track ends without colliding with the test environment agents (blue cars). The system liveness requirement is, $\varphi^f_{\text{sys}} := (y_{\text{sys}} = 2)$, and its safety requirement of no collisions is with test agent labeled i, is: $\neg(y_{\text{sys}} = y_{\text{test},i} \wedge x_{\text{sys}} = x_{\text{test},i}) \in \varphi^s_{\text{sys}}$. In the first two panels, we observe the test agent changing lanes in front of and behind a test car, respectively. In the merged test execution of the third panel, we see the test agent change lanes exactly in between the two blue cars.

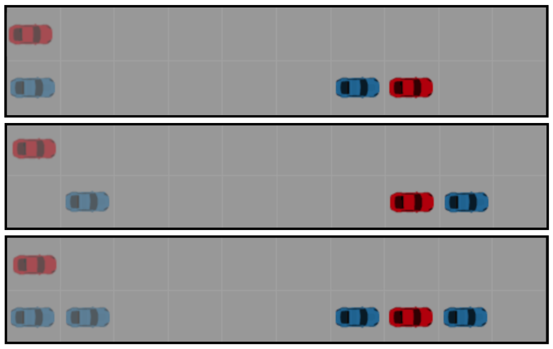


Figure 5.3: Lane change example with initial (left) and final (right) configurations. The x-coordinates are numbered from left to right, and y-coordinates are numbered top to bottom, starting from 1. The system (red) is required to merge into the lower lane without colliding. Merging in front of (top), behind (center), or in between (bottom) tester agents (blue).

5.3 Strong Merge Operator

In this section, we formalize the the construction of a single test specification from unit test specifications using the *strong merge* operator from contract theory. Additionally, we will introduce the notion of adding temporal constraints to the merged test specification to ensure that the resulting test execution reliably satisfies all the unit test specifications. Finally, for the merged test specifications, we use Monte-Carlo Tree Search to find a test strategy on the game graph such that a metric of difficulty is maximized.

The *strong merge operator* defines the merge of two contracts $\mathcal{C}_1$ and $\mathcal{C}_2$ as follows:

$$\begin{aligned} \mathcal{C}_1 \bullet \mathcal{C}_2 =&(a_1 \wedge a_2, (a_1 \wedge a_2) \rightarrow [(a_1 \rightarrow g_1) \wedge (a_2 \rightarrow g_2)]) \\ =&(a_1 \wedge a_2, \neg a_1 \vee \neg a_2 \vee (g_1 \wedge g_2)). \end{aligned} \tag{5.5}$$

Additionally, other operators from assume-guarantee contract theory such as *composition* and *conjunction* [122, 124] will be introduced later in the chapter. Among all these operators, strong merge is the only operator that conjoins assumptions of the individual contracts, and consequently, enforces all unit test specifications to hold true. Thus, we choose the strong merge operator to derive the merged test specification.

Given any two unit test specifications, $\varphi_{\text{test},1}$ and $\varphi_{\text{test},2}$, the corresponding contracts are $\mathcal{C}_1 = (a_1, a_1 \rightarrow g_1)$ and $\mathcal{C}_2 = (a_2, a_2 \rightarrow g_2)$, where $a_i = (\varphi^{\text{init}}_{\text{sys}} \wedge \square\varphi^{s}_{\text{sys}} \wedge \square\diamond\varphi^{f}_{\text{sys}})$ is the assumptions on the system (under test), and $g_i = (\varphi^{\text{init}}_{\text{test},i} \wedge \square\varphi^{s}_{\text{test},i} \wedge \square\diamond\varphi^{f}_{\text{test},i} \wedge \square\psi^{s}_{\text{test},i} \wedge \square\diamond\psi^{f}_{\text{test},i})$ is the guarantees for unit test i. We use the term $g_{t,i} := \psi^{f}_{\text{test},i})$ to refer to the liveness portion of the test objective unknown to the system under test.

Remark 5.2. We make a few simplifying assumptions on the unit test guarantees g_i. First, we assume that the only recurrence requirements in the test specification is $\Box \Diamond \psi^f_{\text{test},i}$, which is not known to the system since it is not a part of the system's assumptions on the environment. Second, we assume that the merged test environment $T_{\text{test},m}$ is a simple Cartesian product of the unit test environments, $T_{\text{test},1}$ and $T_{\text{test},2}$. On the merged test environment, we take the agents from the individual tests: we translate the initial conditions of the agents in the unit tests $\varphi^{\text{init}}_{\text{test},1}$ and $\varphi^{\text{init}}_{\text{test},2}$, and test agent dynamics $\varphi^s_{\text{test},1}$ and $\varphi^s_{\text{test},2}$ are also the same.

Definition 5.7 (Merged Test Specification)**.** From the merged contract $\mathcal{C}_m := \mathcal{C}_1 \bullet \mathcal{C}_2 = (a_m, a_m \rightarrow g_m)$, the specification $\varphi_{\text{test},m} = a_m \rightarrow g_m$, where $a_m = a_1 \wedge a_2$, and $g_m = [(a_1 \rightarrow g_1) \wedge (a_2 \rightarrow g_2)]$ is the *merged test objective*. A test environment strategy $\pi_{\text{test},m}$ for merged test objective $\varphi_{\text{test},m}$ results in a test execution $\sigma \models \varphi_{\text{test},m}$.

The following result is taken from [126].

Lemma 5.1. Given unit test specifications $\varphi_{\text{test},1}$ and $\varphi_{\text{test},2}$ such that $\varphi_{\text{test},m} = a_m \rightarrow g_m$ is the corresponding merged test specification. Then, for every test execution $\sigma \models \varphi_{\text{test},m}$ such that $\sigma \models a_m$, we also have that $\sigma \models \varphi_{\text{test},1}$ and $\sigma \models \varphi_{\text{test},2}$.

Proof. Suppose C_1 and C_2 are the assume-guarantee contracts corresponding to unit test specifications $\varphi_{\text{test},1}$ and $\varphi_{\text{test},2}$. Applying strong merge operator on contracts C_1 and C_2, we get:

$$\begin{aligned} \mathcal{C}_1 \bullet \mathcal{C}_2 =& (a_1 \wedge a_2, (a_1 \wedge a_2) \rightarrow [(a_1 \rightarrow g_1) \wedge (a_2 \rightarrow g_2)]) \\ =& (a_1 \wedge a_2, \neg a_1 \vee \neg a_2 \vee (g_1 \wedge g_2)). \end{aligned} \tag{5.6}$$

Thus, the merged test specification $\varphi_{\text{test},m} = \neg a_1 \vee \neg a_2 \vee (g_1 \wedge g_2)$ requires either one of the assumptions to not be satisfied, or for both the guarantees hold. Since $\sigma \models a_m = a_1 \wedge a_2$, and $\sigma \models \varphi_{\text{test},m}$, we get that $\sigma \models \varphi_{\text{test},1}$ and $\sigma \models \varphi_{\text{test},2}$. $\Box$ $\Box$

Guarantees g_1 and g_2 are used guide the choice of a test strategy; strategies that vacuously satisfy the merged test specification by violating the assumptions are not returned. This is necessary in order to give the system an opportunity to satisfy its specification. If the assumptions on the merged test specifications are violated, it would be because of a fault in system design.

Example 5.2 (Lane Change (continued))**.** In the lane change example, the unit test specifications are changing into the lane *behind* a blue car and changing into the lane *in front* of the blue car. For each specification, the saturated assume guarantee contracts are defined as $\mathcal{C}_1 = (a_1, a_1 \rightarrow g_1)$ and $\mathcal{C}_2 = (a_2, a_2 \rightarrow g_2)$ with $a_1 = \varphi_{\text{sys}}^{\text{init}} \wedge \Box\varphi_{\text{sys}}^{s} \wedge \Box\Diamond(y = 2)$ and $g_1 = \Box\Diamond(y = y_1 = 2 \wedge x = x_1 + 1)$, and $a_2 = \varphi_{\text{sys}}^{\text{init}} \wedge \Box\varphi_{\text{sys}}^{s} \wedge \Box\Diamond(y = 2)$ and $g_2 = \Box\Diamond(y = y_2 = 2 \wedge x = x_2 - 1)$ being the assumptions and guarantees of the two individual tests. Thus, applying the strong merge operation to the unit contracts results in the guarantee,

$$g_m = \Box\Diamond(y = y_1 = 2 \wedge x = x_1 + 1) \wedge \Box\Diamond(y = y_2 = 2 \wedge x = x_2 - 1). \quad (5.7)$$

5.4 Temporal Constraints on Merging Tests

Naively merging test objectives might not always result in a merged test execution that checks the constituent unit test objectives. In the running example on lane change, lane change maneuver behind a vehicle in the other lane does not always coincide with a proper lane change in front of another vehicle. That is, there may exist many test executions of changing lanes behind a vehicle, and some of them, but not all, coincide with changing lanes in front of another vehicle. In these scenarios, the test specifications can be merged *in parallel*, without any additional temporal constraints on how agents for each test environment must operate.

However, when all executions resulting from a one of the unit test specification also satisfy the other (as we will see in the unprotected left turn example), the merged test specification alone is not sufficient. We need to add temporal constraints so that there is a time in which each test specification is checked individually.

The following result is taken from [126].

Lemma 5.2. If for two test specifications $\varphi_{\text{test},1}$ and $\varphi_{\text{test},2}$, and the set of all test executions Σ, we have $\sigma \models \varphi_{\text{test},1} \iff \sigma \models \varphi_{\text{test},2} \forall \sigma \in \Sigma$, then these tests cannot be parallel-merged. Instead, the temporal constraint must be enforced on $g_{t,1}$ and $g_{t,2}$.

Proof. We refine the general specification in equation (5.6), which allows any temporal structure, to include the temporal constraints in the guarantees. The temporally constrained merged test specification is thus defined as $\varphi'_{\text{test},m} = a_m \rightarrow g'_m$, with

$$g'_m = \neg a_1 \vee \neg a_2 \vee (\Diamond(g_{t,1} \wedge \neg g_{t,2}) \wedge \Diamond(\neg g_{t,1} \wedge g_{t,2}) \wedge (g_1 \wedge g_2)). \quad (5.8)$$

Because any trace σ satisfying $\varphi'_{\text{test},m}$ will also satisfy $\varphi_{\text{test},m}$, $\sigma \models \varphi'_{\text{test},m} \Rightarrow \sigma \models \varphi_{\text{test},m}$. Any test trace satisfying this specification will consist of at least one occurrence of visiting a state satisfying $g_{t,1}$ and not $g_{t,2}$ and vice versa. Thus the guarantees of the specifications for each unit test, $g_{t,1}$ and $g_{t,2}$ are checked individually during the merged test which satisfies the temporal constraints. $\square$

Receding Horizon Synthesis of Test strategy Filter

Since the test specification characterizes the set of possible test executions, we need a strategy for the test environment that is consistent with the test specification. In this section, we detail the construction of an auxiliary game graph and algorithms for receding horizon synthesis of the test specification on the auxiliary game graph. This filter will then be used to find the test strategy using Monte-Carlo Tree Search.

Auxiliary Game Graph G_{aux}

Assume we are given a game graph $G = (V, E)$ constructed according to Definition 5.4, and a (merged) test specification $\varphi_{\text{test},m}$ in $GR(1)$ form as in equation (5.2). Then, for each recurrence requirement in the test specification, $\square\diamond\, \psi^f_{\text{test}}$, we can find a set of states $\mathcal{I} = \{i_1, \ldots, i_n\} \subseteq V$ that satisfy the propositional formula ψ^f_{test}. For each $i \in \mathcal{I}$, there exists a non-empty subset of vertices $V^s \subseteq V$ that can be partitioned into $\{\mathcal{V}^i_0, \ldots, \mathcal{V}^i_n\}$. We follow [18] in partitioning the states; $\mathcal{V}^i_k$ is the set of states in V that is exactly k steps away from the goal state i. From this partition of states, we can construct a partial order, $\mathcal{P}^i = (\{\mathcal{V}^i_0, \ldots, \mathcal{V}^i_n\}, \leq)$, such that $\mathcal{V}^i_l \leq \mathcal{V}^i_{l-1}$ for all $l \in \{0, \ldots, n\}$. This partial order will be useful in the receding horizon synthesis of the test strategy outlined below [18].

We construct an auxiliary game graph $G_{\text{aux}} = (V_{\text{aux}}, E_{\text{aux}})$ (illustrated in Figure 5.4) to accommodate any temporal constraints on the merged test specification before proceeding to synthesize a filter for the test strategy. Without loss of generality, we elaborate on the auxiliary graph construction in the case of one recurrence requirement in each unit specification, but this approach can be easily extended to multiple progress requirements. An illustration of the auxiliary graph is given in Figure 5.4. Let $\varphi_{\text{test},1}$ and $\varphi_{\text{test},2}$ be the two unit test specifications, with $\psi^f_{\text{test},1}$ and $\varphi^f_{\text{test},2}$, respectively. First, we make three copies of the game graph $G = (V, E)$ — $G_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}} = (V_{1\vee 2}, E_{1\vee 2})$, $G_{\varphi_{\text{test},1}} = (V_1, E_1)$, and $G_{\varphi_{\text{test},2}} = (V_2, E_2)$. Note that, $V_{1\vee 2}$, V_1 and V_2 are all copies of V, but are denoted differently for differentiating between the vertices that constitute G_{aux}, and a similar argument applies to edges of these subgraphs. Let $\mathcal{V}^i_0 = \bigcup \mathcal{V}^{i_j}_0 \subseteq V_{1\vee 2}$ be the set of states in $G_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}$ that satisfy propositional formula $\psi^f_{\text{test},1}$. Likewise, the set of states $\mathcal{V}^k_0 \subseteq V_{1\vee 2}$ satisfy

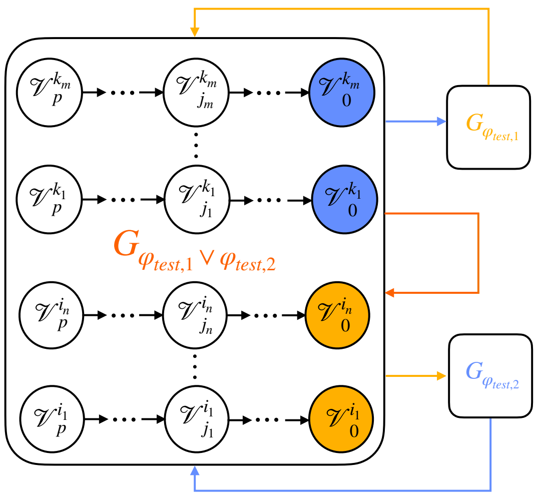


Figure 5.4: Auxiliary game graph construction for the merged test specification of unit test specifications $\varphi_{\text{test},1}$ and $\varphi_{\text{test},2}$. Subgraphs $G_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}$, $G_{\varphi_{\text{test},1}}$ and $G_{\varphi_{\text{test},2}}$ are copies of the game graph G constructed per Definition 5.4. In $G_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}$, the sets of states at which the progress propositional formulas of test specifications, $\varphi_{\text{test},1}$ and $\varphi_{\text{test},2}$, are satisfied are shaded yellow and blue, respectively.

the propositional formula $\psi^f_{\text{test},2}$.

Now, we connect the various subgraphs through the vertices in $\mathcal{V}^i_0$ and $\mathcal{V}^k_0$. Let (v^k_0, u) be an outgoing edge from a node $v^k_0 \in \mathcal{V}^k_0$, and let u_1 be the vertex in subgraph $G_{\text{test},1}$ that corresponds to vertex u in $G_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}$. Remove edge (v^k_0, u) and add the edge (v^k_0, u_1). Likewise, every outgoing edge from $\mathcal{V}^i_0 \cup \mathcal{V}^k_0$ in $G_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}$ is replaced by adding edges to $G_{\varphi_{\text{test},1}}$ and $G_{\varphi_{\text{test},2}}$. On subgraphs $G_{\varphi_{\text{test},1}}$ and $G_{\varphi_{\text{test},2}}$, vertices are partitioned and partial orders are constructed once again for $\psi^f_{\text{test},1}$ and $\psi^f_{\text{test},2}$, respectively. From $\mathcal{V}^i_0$ defined on the nodes of the graph $G_{\varphi_{\text{test},1}}$, every outgoing edge is replaced by a corresponding edge to $G_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}$. Subgraph $G_{\varphi_{\text{test},2}}$ is connected back to $G_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}$ in a similar manner. The construction of the auxiliary graph G_{aux} and partial order $\mathcal{P}^i$ is summarized in Algorithm 7. Our choice of constructing the auxiliary graph in this manner is amenable to constructing a simple partial order as outlined below.

Assumption 5.1. For unit test specifications $\varphi_{\text{test},1}$ and $\varphi_{\text{test},2}$ with recurrence specifications φ^p_1 and φ^p_2, respectively, such that $\varphi^p_1 = \square\diamond\, \psi^f_{\text{test},1}$ and $\varphi^p_2 = \square\diamond\, \psi^f_{\text{test},2}$. Suppose there exist partial orders $\mathcal{P}^i = (\{\mathcal{V}^i_n, \ldots, \mathcal{V}^i_0\}, \leq)$ and $\mathcal{P}^k = (\{\mathcal{V}^k_m, \ldots, \mathcal{V}^k_0\}, \leq)$ on G corresponding to $\psi^f_{\text{test},1}$ and $\psi^f_{\text{test},2}$, respectively. Assume that at least one of the following is true: (a) there exists an edge (u_1, v_2) where $u_1 \in \mathcal{V}^i_0$ and $v_2 \in \mathcal{V}^k_j$ for some $j \in \{1, \ldots, m\}$, (b) there exists an edge (u_2, v_1) where $u_2 \in \mathcal{V}^k_0$ and $v_1 \in \mathcal{V}^i_j$ for some $j \in \{1, \ldots, n\}$.

The following Lemma is taken from [126].

Lemma 5.3. If Assumption 5.1 holds, there exists a partial order on G_{aux} for the merged recurrence propositional formula, $\psi^f_{\text{test},m}$, where $\psi^f_{\text{test},m}$ is the propositional

Algorithm 7: Construction of Partial Order and Auxiliary Graph

```
1: procedure GAUX($(G, \psi^f_{\text{test},1}, \psi^f_{\text{test},2})$)
     Input: Game graph $G = (V, E)$, propositional formulas $\psi^f_{\text{test},1}$ and $\psi^f_{\text{test},2}$ constituting the progress requirements of unit test specifications
     Output: Auxiliary game graph $G_{\text{aux}}$
2:
3:     $G_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}} := (V, E) \leftarrow G$ Initialize subgraph
4:     $G_{\varphi_{\text{test},1}} := (V_1, E_1) \leftarrow G$ Initialize subgraph
5:     $G_{\varphi_{\text{test},2}} := (V_2, E_2) \leftarrow G$ Initialize subgraph
6:     $[\mathcal{P}^i_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}, \mathcal{P}^k_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}] \leftarrow$ Partial order$(G_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}, [\psi^f_{\text{test},1}, \psi^f_{\text{test},2}])$
7:     $\mathcal{P}^i_{\varphi_{\text{test},1}} \leftarrow$ Partial order$(G_{\varphi_{\text{test},1}}, \psi^f_{\text{test},1})$
8:     $\mathcal{P}^k_{\varphi_{\text{test},2}} \leftarrow$ Partial order$(G_{\varphi_{\text{test},2}}, \psi^f_{\text{test},2})$
9:     $E^r_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}} \subseteq E$ Deleting outgoing edges from $\mathcal{V}^i_0 \cup \mathcal{V}^k_0 \subseteq V$ within $G_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}$
10:    $E^a_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}$ Adding edges from $\mathcal{V}^i_0 \cup \mathcal{V}^k_0 \subseteq V$ to subgraphs $G_{\varphi_{\text{test},1}}$ and $G_{\varphi_{\text{test},2}}$
11:    $E^r_{\varphi_{\text{test},1}} \subseteq E_1$ Deleting outgoing edges from $\mathcal{V}^i_0 \subseteq V_1$ within $G_{\varphi_{\text{test},1}}$
12:    $E^a_{\varphi_{\text{test},1}}$ Adding edges from $\mathcal{V}^i_0 \subseteq V_1$ to subgraph $G_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}$
13:    $E^r_{\varphi_{\text{test},2}} \subseteq E_2$ Deleting outgoing edges from $\mathcal{V}^k_0 \subseteq V_2$ within $G_{\varphi_{\text{test},2}}$
14:    $E^a_{\varphi_{\text{test},2}}$ Adding edges from $\mathcal{V}^k_0 \subseteq V_2$ to subgraph $G_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}$
15:    $V_{\text{aux}} = V \cup V_1 \cup V_2$
16:    $E_{\text{aux}} = (E \setminus E^r_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}) \cup (E_1 \setminus E^r_{\varphi_{\text{test},2}}) \cup (E_2 \setminus E^r_{\varphi_{\text{test},2}}) \cup E^a_{\varphi_{\text{test},2}} \cup E^a_{\varphi_{\text{test},1}} \cup E^a_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}$
17:    $G_{\text{aux}} = (V_{\text{aux}}, E_{\text{aux}})$
18:    return $G_{\text{aux}}, \mathcal{P}^i_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}, \mathcal{P}^k_{\varphi_{\text{test},1} \vee \varphi_{\text{test},2}}, \mathcal{P}^i_{\varphi_{\text{test},1}}, \mathcal{P}^k_{\varphi_{\text{test},2}}$
```

formula that evaluates to true at: (i) all $v \in V_{1 \vee 2}$ such that $v \models \psi^f_{\text{test},1} \wedge \psi^f_{\text{test},2}$, (ii) all $v \in V_1$ such that $v \models \psi^f_{\text{test},1}$, and (iii) all $v \in V_2$ such that $v \models \psi^f_{\text{test},2}$.

Proof. Let $\mathcal{V}^m_0 \subseteq V_{\text{aux}}$ denote the non-empty set of states at which $\psi^f_{\text{test},m}$ evaluates to true. Then, let $\mathcal{V}^m_j \subseteq V_{\text{aux}}$ be the subset of states that is at least j steps away from a vertex in $\mathcal{V}^m_0$. Then, construct the partial order $\mathcal{P}^m = (\{\mathcal{V}^m_l, \ldots, \mathcal{V}^m_0\}, \leq)$, where l is the distance of the farthest vertex connected to $\mathcal{V}^m_0$. The subset of vertices $\bigcup_j \mathcal{V}^m_j \subseteq V_{\text{aux}}$ is non-empty because $\mathcal{V}^m_0$ is non-empty. Furthermore, from Assumption 5.1, if (a) holds, there exists a $j \in \{1, \ldots, l\}$ such that $\mathcal{V}^m_j \cap \mathcal{V}^i_0$ is non-empty. Likewise, if (b) holds, there exists a $j \in \{1, \ldots, l\}$ such that $\mathcal{V}^m_j \cap \mathcal{V}^k_0$ is non-empty. Therefore, for some $j \in \{1, \ldots, l\}$ there exists a test execution σ over the game graph G_{aux} such that $\sigma \models \Box \Diamond \psi^f_{\text{test},m}$. □

Remark 5.3. If Assumption 5.1 is not true, the unit tests corresponding to test objectives $\varphi_{\text{test},1}$ and $\varphi_{\text{test},2}$ cannot be merged.

Receding Horizon Synthesis on G_{aux}

We use receeding horizon synthesis for a more scalable construction of the winning set $\mathcal{W}^{\mathcal{I}}$ — the set of states from which the test environment can still satisfy the test objective. This winning set will then serve as a safety filter during Monte Carlo Tree Search to exclude trajectories that do not satisfy the test objectives. Further details on receeding horizon temporal logic planning can be found in [18].

For a test objective $\varphi_{\text{test},1}$ with progress propositional formula $\psi^f_{\text{test},1}$, let $\mathcal{I}$ be the set of states on G_{aux} at which $\psi^f_{\text{test},1}$ evaluates to true. Suppose the product state of the system and environment is some j steps away from a state $i \in \mathcal{I}$: $v \in \mathcal{V}^i_{j+1}$. If we want the test environment to guide the execution to two steps ahead to $\mathcal{V}^i_{j-1}$, the intermediate specification for the test environment is as follows.

$$\psi^i_j = (v \in \mathcal{V}^i_{j+1} \wedge \Phi \wedge \Box\varphi^s_{\text{sys}} \wedge \Box\Diamond\varphi^f_{\text{sys}}) \rightarrow (\Box\Diamond(\mu^i_{\text{visited},j-1}) \wedge \Box\varphi^s_{\text{test}} \wedge \Box\psi^s_{\text{test}} \wedge \Box\Phi), \tag{5.9}$$

where Φ is the invariant condition that ensures that ψ^i_j is realizable, and $\mu^i_{\text{visited},j-1}$ is an auxiliary variable which becomes true (and remains true) once the product state v has reached a state $j-1$ steps away from i: $v \in \mathcal{V}^i_{j-1}$. The construction of the invariant set Φ is given in [18]. It is sufficient to for the test environment to guide the execution to at least one node $i \in \mathcal{I}$, which can be formally stated as,

$$\Psi^{\mathcal{I}}_j = \vee_{i \in \mathcal{I}} \, \psi^i_j \, . \tag{5.10}$$

The set of states of G_{aux} from which the test environment has a strategy to satisfy equation (5.10) is denoted by $\mathcal{W}^{\mathcal{I}}_j$. This set serves as a short-horizon filter to guide the test strategy from j steps away to the goal set $\mathcal{I}$.

Consider the set of shortest paths: $\{Path(v, i) | v \in V, \, i \in \mathcal{I}\}$. Let j_{max} denote the length of the longest path in this set. The overall winning set filter is the union of individual winning sets:

$$\mathcal{W}^{\mathcal{I}} = \bigcup_{j=1}^{j_{\text{max}}} \mathcal{W}^{\mathcal{I}}_j \, . \tag{5.11}$$

Construction of $\mathcal{W}^{\mathcal{I}}$, and its use as a safety filter for finding test strategies using MCTS is outlined in Algorithm 8. For the merged test objective, $\mathcal{W}^{\mathcal{I}}$ is generated on G_{aux} where $\mathcal{I}$ is the set of states corresponding to $\psi^f_{\text{test},m}$. We will need the following notation to denote the graph induced by the set $\mathcal{W}^{\mathcal{I}}$. Let $G_{\mathcal{W}^{\mathcal{I}}} = (V_{\mathcal{W}}, E_{\mathcal{W}})$ be the subgraph of G_{aux} induced by $\mathcal{W}^{\mathcal{I}}$ such that $V_{\mathcal{W}} = \mathcal{W}^{\mathcal{I}} \subseteq V_{\text{aux}}$ and $E_{\mathcal{W}} = \{(u, v) \in E_{\text{aux}} \,|\, u \in \mathcal{W}^{\mathcal{I}} \wedge v \in \mathcal{W}^{\mathcal{I}}\}$.

On $\mathcal{W}^{\mathcal{I}}$ as a test strategy filter

Inspired by work on shield synthesis [127], we use the winning set $\mathcal{W}^{\mathcal{I}}$ as a filter to guide rollouts in the Monte Carlo Tree Search sub-routine for finding the test strategy. Since $\Psi_j^{\mathcal{I}}$ is a disjunction of short-horizon $GR(1)$ specifications, it is possible that an execution always satisfies $\Psi_j^{\mathcal{I}}$ without ever satisfying the progress requirement $\square \diamond \psi_{\text{test}}^f$. This happens when the test execution makes progress towards some $i \in \mathcal{I}$ but never actually reaches a goal in $\mathcal{I}$, resulting in a live lock. Further details addressing this are given in the Appendix. We *assume* that the graph is constructed such that there are no such cycles. In addition to using $W^{\mathcal{I}}$ to ensure that $\Psi_j^{\mathcal{I}}$ will always be satisfied, we enforce progress by only allowing the search procedure to take actions that will lead to a state which is closer to one of the goals $i \in \mathcal{I}$. Thus, the search procedure will ensure that for every state $v_l \in \mathcal{V}_j^i$, the control strategy for the next horizon will end in $v_{l'} \in \mathcal{V}_k^i$, such that $k \leq l$ for at least one goal $i \in \mathcal{I}$.

The following theorem and proof is taken from [126].

Theorem 5.1. Receding horizon synthesis of test filter $\mathcal{W}^{\mathcal{I}}$ is such that any test execution σ on $G_{\mathcal{W}^{\mathcal{I}}}$ starting from an initial state in $V_{\mathcal{W}} \cap V$ satisfies the test specification in equation (5.2).

Proof. For the recurrence formula of the merged test specification, $\square \diamond \psi_{\text{test},m}^f$, suppose there exists a single vertex on G_{aux} that satisfies $\psi_{\text{test},m}^f$. Then, it is shown in [18] that if there exists a partial order $(\{\mathcal{V}_p^i, \ldots, \mathcal{V}_0^i\}, \leq)$ on G_{aux}, we can find a set of vertices $\mathcal{W}^i \subseteq V_{\text{aux}}$, such that every test execution σ that remains in $\mathcal{W}^i$, will satisfy the safety requirements $\square \varphi_{\text{test}}^s$ and $\square \psi_{\text{test}}^s$, and the invariant Φ. Furthermore, given the partial order $(\{\mathcal{V}_p^i, \ldots, \mathcal{V}_0^i\}, \leq)$, one can find a test policy to ensure that the σ makes progress along the partial order such that for some $t > 0$, $\sigma_t \in \mathcal{V}_0^i$. However, in case of multiple vertices in G_{aux} that satisfy $\psi_{\text{test},m}^f$, we need to extend the receding horizon synthesis to specification $\Psi_j^{\mathcal{I}}$. We construct the filter $\mathcal{W}^{\mathcal{I}}$ and also check that for every test execution σ, there exists $i \in \mathcal{I}$ such that for every $k \geq 0$, $\sigma_k \in \mathcal{V}_j^i$ and $\sigma_{k+1} \in \mathcal{V}_{j'}^i$. Therefore, because the auxiliary game graph is assumed to not have cycles, the test execution makes progress on the partial order of at least one $i \in \mathcal{I}$ at each timestep, thus eventually satisfying $\psi_{\text{test},m}^f$. Thus every execution of our algorithm will satisfy equation (5.2). $\square$

Algorithm 8: Merge Unit Tests $(\varphi_{\text{test},1}, \varphi_{\text{test},2}, \varphi_{\text{sys}}, \mathcal{T}_{\text{sys}}, \mathcal{T}_{\text{test},1}, \mathcal{T}_{\text{test},2}, \rho)$

```
1: procedure MERGEUNITTESTS($(\varphi_{\text{test},1}, \varphi_{\text{test},2}, \varphi_{\text{sys}}, \mathcal{T}_{\text{sys}}, \mathcal{T}_{\text{test},1}, \mathcal{T}_{\text{test},2}, \rho)$)
     Input: Unit test specifications $\varphi_{\text{test},1}$ and $\varphi_{\text{test},2}$, system specification $\varphi_{\text{sys}}$, System $\mathcal{T}_{\text{sys}}$, unit test environments, $\mathcal{T}_{\text{test},1}$ and $\mathcal{T}_{\text{test},2}$, and quantitative metric of robustness $\rho$,
     Output: Merged test specification $\varphi_{\text{test},m}$, Merged test environment $\mathcal{T}_{\text{test},m}$, Merged test policy $\pi_{\text{test},m}$
2:     $\mathcal{C}_1$, $\mathcal{C}_2 \leftarrow$ Construct contracts for $\varphi_{\text{test},1}$ and $\varphi_{\text{test},2}$
3:     $\mathcal{T}_{\text{test}} \leftarrow \mathcal{T}_{\text{test},1} \times \mathcal{T}_{\text{test},2}$ Merged test environment
4:     $\mathcal{T}_{\text{prod}} \leftarrow \mathcal{T}_{\text{sys}} \times \mathcal{T}_{\text{test}}$ Product transition system
5:     $G \leftarrow$ Game graph from product transition system $\mathcal{T}_{prod}$
6:     $\mathcal{C}_m := (a_m, a_m \rightarrow g_m) \leftarrow$ strong merge($\mathcal{C}_1$, $\mathcal{C}_2$) Constructing the merged specification
7:     $\varphi_{\text{test},m} \leftarrow a_m \rightarrow g_m$ Merged test specification
8:     $G_{\text{aux}} \leftarrow$ Auxiliary game graph.
9:     $\mathcal{I} = \{s \in \mathcal{V}_{\text{aux}} | s \models \psi^f_{\text{test},m}\}$ Defining goal states and partial orders
10:    for $i \in \mathcal{I}$ do
11:        $\mathcal{P}^i := \{(\mathcal{V}^i_p, \dots, \mathcal{V}^i_0)\} \leftarrow$ Partial order for goal $i$
12:        $\psi^i_j \leftarrow$ Receding horizon specification for goal $i$ at distance $j$
13:    $\mathcal{W}^{\mathcal{I}} := \bigcup\{\mathcal{W}^i_j\} \leftarrow$ Test policy filter for goal $i$ at a distance of $j$
14:    $\pi_{\text{test},m} \leftarrow$ Searching for test policy guided by $\mathcal{W}^{\mathcal{I}}$
15:    return $\varphi_{\text{test},m}$, $\mathcal{T}_{\text{test},m}$, $\pi_{\text{test},m}$
```

Test Strategy Synthesis: Monte Carlo Tree Search is used to sample trajectories on G_{aux} after applying the safety filter $\mathcal{W}^{\mathcal{I}}$ to find a reactive test strategy $\pi_{\text{test},m}$ that satisfies the merged test objective. This procedure allows for optimizing for a metric of difficulty while also ensuring that all test strategies do not construct impossible tests for the system. Using MCTS with an upper confidence bound (UCB) was introduced in [128] as the upper confidence bound for trees (UCT) algorithm, which guarantees that given enough time and memory, the tree search converges to the optimal solution. We use MCTS to find $\pi^*_{\text{test},m}$, the optimal solution to Problem 5.1 for the merged test objective.

The following theorem and proof are taken from [126].

Theorem 5.2. Algorithm 1 is sound.

Proof. This follows by construction of the algorithm and the use of MCTS with UCB. Given a test policy π_{test} and a system policy π_{sys}, for every resulting execution $\sigma_{\pi_{\text{sys}} \times \pi_{\text{test}}}$ starting from an initial state in $\mathcal{W}^{\mathcal{I}}$, it is guaranteed that $\sigma \models \varphi_{\text{test},m}$

by Theorem 5.1. This is because for any action chosen by the test environment according to the policy π_{test} found by MCTS, we are guaranteed to remain in $\mathcal{W}^{\mathcal{I}}$ for any valid system policy π_{sys}. If $\mathcal{W}^{\mathcal{I}} = \emptyset$ or the initial state is not in $\mathcal{W}^{\mathcal{I}}$, the algorithm will terminate before any rollout is attempted and no policy is returned. It can be shown that the probability of selecting the optimal action converges to 1 as the limit of the number of rollouts is taken to infinity. For convergence analysis of MCTS, please refer to [128]. $\square$

Complexity: The time complexity of $GR(1)$ synthesis is in the order of $O(|N|^3)$, where N is the number of states needed to define the GR(1) formula. To improve scalability, our algorithm uses a receding horizon approach to synthesize the winning sets, which further reduces the time complexity. The upper confidence tree algorithm of MCTS is given as $O(ijkl)$ with j the number of rollouts, k the branching factor of the tree, l the depth of the tree, and i the number of iterations.

Simulation Experiments

This framework is illustrated on discrete gridworld examples where the system controller is non-deterministic and the test agents behave according to the synthesized test strategy. The Temporal Logic and Planning Toolbox (TuLiP) [109] is used for constructing winning sets [108], and an open-source script[1] for the online MCTS algorithm to find the test strategy. Simulation videos of at the linked GitHub repository[2].

Lane Change

For the lane change example, we define $\rho(\sigma)$ as the x-value of the cell in which the system finished its lane change maneuver.The test strategy is found to be consistent with the test objective in equation (5.7) while also maximizing by maximizing $\rho(\sigma)$. The metric ρ is the chosen metric of difficulty; the closer to the end of the lane, the fewer attempts the system will have for a successful lane change. Snapshots of the resulting test execution are depicted in Figure 5.5.

Unprotected left turn

In this example, the test environment consists of a pedestrian and a blue car, and the system is the red car, as illustrated in Figure 5.6. The unit tests correspond to

[1]https://gist.github.com/qpwo/c538c6f73727e254fdc7fab81024f6e1
[2]https://github.com/jgraeb/MergeUnitTests

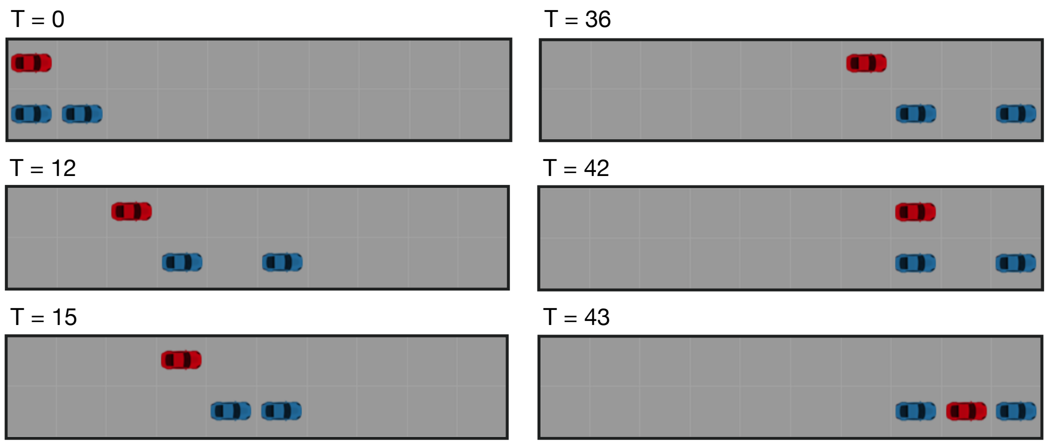


Figure 5.5: Snapshots during the execution of the test generated by our framework. The system under test (red car) needs to merge onto the lower lane between the two test agents (blue cars).

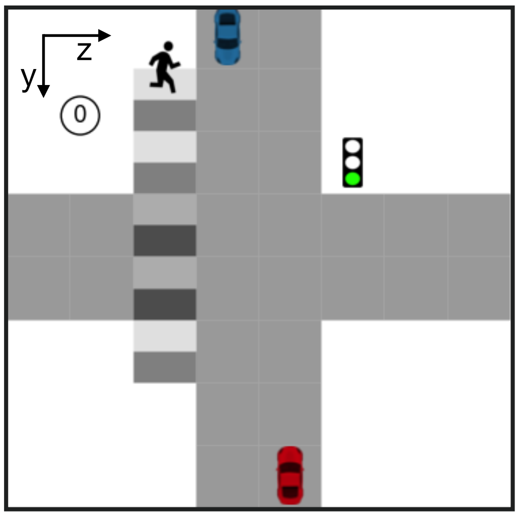


Figure 5.6: Layout of the unprotected left turn at intersection example. The system starts in cell (7,4) and wants to reach the goal cell (0,3), while the initial positions of the test agents are at the beginning of the road and crosswalk.

waiting for an oncoming car to pass the intersection, and waiting for a pedestrian to pass before taking a left turn.

The system requirement is to safely take an unprotected left turn. The unit specifications for waiting for the pedestrian are defined according to equation (5.2):

$$\varphi_{sys}^{\text{init}} = (\mathbf{x}_S \in I_S), \quad \varphi_{sys}^{f} = (\mathbf{x}_S \in \mathcal{S}_G), \quad \psi_{\text{test},ped}^{f} = (\mathbf{x}_S \in \mathcal{S}_P \wedge \mathbf{x}_P \in T_P)\,, \tag{5.12}$$

where $\mathbf{x}_S$ is the system state, I_S is the initial state of the system, $\mathcal{S}_G$ is the set of goal state following the left turn, $\mathbf{x}_P$ is the pedestrian state, and $\mathcal{S}_P$ are the states in which the car must wait for the pedestrian if the pedestrian state is in T_P. Similarly, the unit test objective for waiting for the test car is given as follows:

$$\varphi_{\text{sys}}^{\text{init}} = (\mathbf{x}_S \in I_S), \quad \varphi_{\text{sys}}^{f} = (\mathbf{x}_S \in \mathcal{S}_G), \quad \psi_{\text{test},car}^{f} = (\mathbf{x}_S \in \mathcal{S}_C \wedge \mathbf{x}_C \in T_C)\,, \tag{5.13}$$

where the C denotes the test agent car in blue. The coordinate system has origin in the upper left corner with cell $(y, z) = (0, 0)$, with the y-axis facing south and the z-axis facing east. The crosswalk locations are numbered from north to south, starting at 0.

The initial states of the test agents are $\mathbf{x}_C = (0, 3)$ and $\mathbf{x}_P = 0$, and the initial state of the system is $\mathbf{x}_S = (7, 4)$. The goal state for the system is $\mathbf{x}_G = (0, 3)$. In this example, $\mathbf{x}_G$ is the only element in $\mathcal{S}_G$. The state in which the system needs to wait for the pedestrian and the car, $\mathcal{S}_C$ and $\mathcal{S}_P$, respectively, are both $\mathbf{x} = (4, 4)$. When the test agent has not yet approached the intersection or has just approached the intersection, the system must wait. These states of the test agent are $\mathcal{T}_C = \{(0, 3), (1, 3), (2, 3), (3, 3)\}$. Similarly, the states of the pedestrian for which the system has to wait are $\mathcal{S}_P = \{1, 2, 3, 4, 5\}$, which represent the cells on the crosswalk, that map to grid coordinates. Note that if the pedestrian is in cell 0, the system is not required to wait for the pedestrian, as she is too far away from the road. The traffic light sequence is predetermined, the light will be green for a fixed number of time steps t_g, followed by t_y time steps of yellow and red for t_r time steps. We are assuming that the system designer supplied the robustness metric as the time until the traffic light turns red, resulting in a harder test the closer the light is to red once the system successfully takes the turn.

The robustness metric at a state is defined to be the time left until the traffic light changes to red, starting at the moment the system enters the intersection. The robustness over the entire trajectory is the minimum value of the robustness of all states in the trajectory. The smaller the value of this robustness, the more difficult the test for the reason that the system has fewer opportunities to successfully complete its task.

Additionally, this is an example in which all trajectories of the car taking a left turn while waiting for the pedestrian will also satisfy the condition of waiting for the test car and vice-versa. That is, $\sigma \models \Diamond\, \psi_{\text{test},ped}^{f} \iff \sigma \models \Diamond\, \psi_{\text{test},car}^{f}$. As a result, for this example, we add temporal constraints to the merged test objective to ensure that the two events do not entirely coincide.

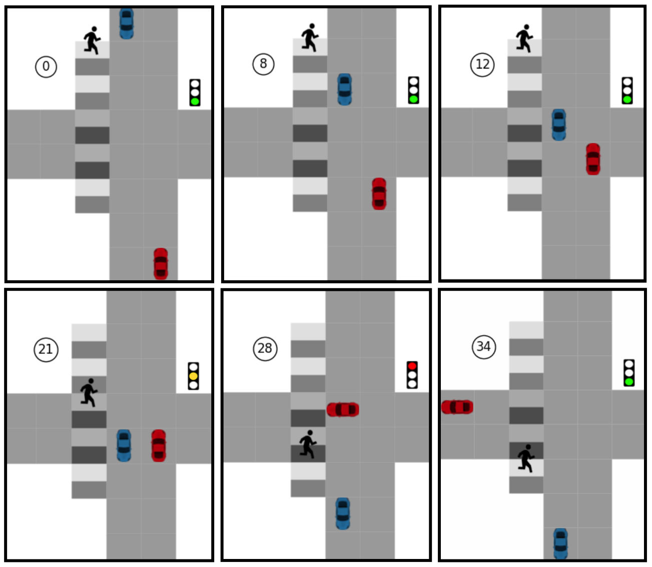


Figure 5.7: Snapshots during the execution of the unprotected left turn test generated by our framework. The autonomous vehicle (AV) under test (red) should take an unprotected left turn and wait for the pedestrian and the car (blue) individually, which are agents of the test environment. In the snapshots at time steps 8 and 12, the AV waits just for the car, and in time step 21 it waits just for the pedestrian.

The resulting test execution is shown in Figure 5.7. As expected, we see the system first waiting for the test car to pass the intersection. Even after the tester car passes, the pedestrian is still traversing the crosswalk, causing the system to wait for the pedestrian, satisfying the temporally constrained merged test objective.

5.5 Contract Theory for Formalizing Compositional Testing

So far, we have seen the use of the strong merge operator in constructing a single test from unit tests. In this part of the chapter, we explore the use of assume-guarantee contracts not only to combine tests, but also split complex tests into simpler unit tests on the overall system or on subsystems. We further explore the algebra of assume-guarantee contracts, and leverage contract operators to formalize this reasoning over test objectives. Finally, we illustrate test executions corresponding to the combined and split test structures in a discrete autonomous driving example and an aircraft formation-flying example. This work is a step towards formal methods to construct test campaigns from unit tests.

To apply concepts from this formalism, we introduce the test structure — a tuple that carries i) the formal specifications of the system under test, and ii) the test objective, which is specified by a test engineer. We build on test structures to define test campaigns and specifications for the tester. We address the following questions using the formalism of assume-guarantee contracts:

(Q1) *Comparing Tests:* Is it possible to define an ordering of tests? When is one test considered a refinement of another? See Section 5.8.

(Q2) *Combining Tests:* Can multiple unit test objectives be checked in a single test execution? See Section 5.7.

(Q3) *Splitting Tests:* From a complex test objective, can we split into component-level tests or split the test objective into simpler objectives? See Section 5.9.

5.6 Test Structures and Tester Specifications

For conducting a test, we need i) the system under test and its specification to be tested and ii) specifications for the test environment that ensure that a set of behaviors (specified by the test engineer) can be observed during the test. These sets of desired test behaviors are characterized by the test engineer in the form of a specification. The system specifications make some assumptions about the test environment. The test objective, together with the system specification, is used to synthesize a test environment and corresponding strategies of the tester agents. As a result, the test objective is not made known to the system since doing so would reveal the test strategy to the system. These concepts are formally defined below.

Definition 5.8. The *system specification* is the assume-guarantee contract denoted by $\mathcal{C}^{\text{sys}} = (A^{\text{sys}}, G^{\text{sys}})$, where A^{sys} are the assumptions that the system makes on its operating environment, and G^{sys} denotes the guarantees that it is expected to satisfy if A^{sys} evaluates to $\top$. In particular, A^{sys} are the assumptions requiring a safe test environment, and $\neg A_i^{\text{sys}} \cup G_i^{\text{sys}}$ are the guarantees on the specific subsystem that will be tested.

$$\mathcal{C}^{\text{sys}} = (A^{\text{sys}}, \neg A^{\text{sys}} \cup \bigcap_i (\neg A_i^{\text{sys}} \cup G_i^{\text{sys}})).$$

Definition 5.9. A *test objective* $\mathcal{C}^{\text{obj}} = (\top, G^{\text{obj}})$, where G^{obj} characterizes the set of desired test behaviors, is a formal description of the specific behaviors that the test engineer would like to observe during the test.

These contracts can be refined or relaxed using domain knowledge. Using definitions (5.8) and (5.9), we define a *test structure*, which is the unitary object that we use to establish our framework and for the analysis in the rest of the chapter.

Definition 5.10. A *test structure* is the tuple $\mathfrak{t} = (\mathcal{C}^{\text{obj}}, \mathcal{C}^{\text{sys}})$ comprising of the test objective and the system requirements for the test.

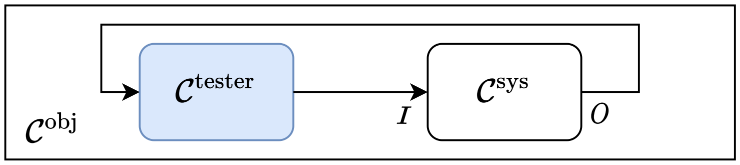

$\mathcal{C}^{\text{tester}}$
$\mathcal{C}^{\text{sys}}$
$\mathcal{C}^{\text{obj}}$
I
O

Figure 5.8: Block diagram showing contracts specifying the system specification $\mathcal{C}^{\text{sys}}$, the test objective $\mathcal{C}^{\text{obj}}$, and the test environment $\mathcal{C}^{\text{tester}}$.

Given the system specification and the test objective, we need to determine the specification for a valid test environment, which will ensure that if the system meets its specification, the desired test behavior will be observed. The resulting test execution will then enable reasoning about the capabilities of the system. If the test is executed successfully, the system passed the test, and conversely, if the test is failed, it is because the system violated its specification and not due to an erroneous test environment.

Now we need to find the specification of the test environment, the tester contract $\mathcal{C}^{\text{tester}}$, in which the system can operate and will satisfy the test objective according to Figure 5.8, with I, O denoting the inputs and outputs of the system contract. This contract can be computed as the mirror of the system contract, merged with the test objective, which is equivalent to computing the quotient of $\mathcal{C}^{\text{obj}}$ and $\mathcal{C}^{\text{sys}}$ [121]:

$$\mathcal{C}^{\text{tester}} = (\mathcal{C}^{\text{sys}})^{-1} \bullet \mathcal{C}^{\text{obj}} = \mathcal{C}^{\text{obj}} / \mathcal{C}^{\text{sys}}.$$

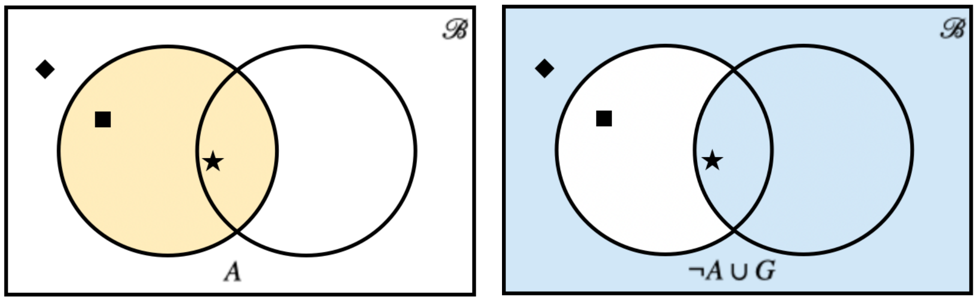
The tester contract can therefore directly be computed as

$$\mathcal{C}^{\text{tester}} = (G^{\text{sys}}, G^{\text{obj}} \cap A^{\text{sys}} \cup \neg G^{\text{sys}}). \tag{5.14}$$

Remark: Since it is the tester's responsibility to ensure a safe test environment, A^{sys}, a test is synthesized with respect to the following specification,

$$\bigcap_i (\neg A_i^{\text{sys}} \cup G_i^{\text{sys}}) \rightarrow A^{\text{sys}} \cap G^{\text{obj}}. \tag{5.15}$$

A successful test execution lies in the set of behaviors $A^{\text{sys}} \cap G^{\text{sys}} \cap G^{\text{obj}}$, and an unsuccessful test execution is the sole responsibility of the system being unable to satisfy its specification. Thus, any implementation of $\mathcal{C}^{\text{tester}}$ will be an environment in which the system can operate and satisfy $\mathcal{C}^{\text{obj}}$ if the system satisfies its specification, a geometric interpretation is shown in Figure 5.9.



(a) Assumptions A of the contract. (b) Guarantees $\neg A \cup G$ of the contract.

Figure 5.9: Geometric interpretation of an assume-guarantee contract (A, G) as a pair of sets of behaviors. The first element of the pair describes the set of behaviors for which the assumptions A hold, and the second element describes the set of behaviors for which G holds or A does not hold. The tester failing to provide the guarantees G (square) does not satisfy the contract. The set of desired test executions is in the intersection of the assumptions and guarantees (star), and the set of test executions that fall outside the assumptions (diamond) are because the system under test failed to satisfy its requirements.

5.7 Combining Tests

Earlier in the chapter, the strong merge operator was used to merge unit test objectives into a single objective. However, it required careful specification of assumptions and guarantees in a single GR(1) specification. Using the test structures introduced in the previous section, combining unit test contracts via the strong merge operator is equivalent to merging unit test specifications. The advantage of the new formalism is that it allows us to easily compose system and test objectives separately, without manual checking. The strong merge of test contracts is defined as follows.

Proposition 5.1. $\mathcal{C}_1^{\text{tester}} \bullet \mathcal{C}_2^{\text{tester}} = (\mathcal{C}_1^{\text{obj}} \parallel \mathcal{C}_2^{\text{obj}})/\left(\mathcal{C}_1^{\text{sys}} \parallel \mathcal{C}_2^{\text{sys}}\right)$.

Proof. Merging tester contracts yields

$$\begin{aligned}
\mathcal{C}_1^{\text{tester}} \bullet \mathcal{C}_2^{\text{tester}} &= (\mathcal{C}_1^{\text{obj}}/\mathcal{C}_1^{\text{sys}}) \bullet (\mathcal{C}_2^{\text{obj}}/\mathcal{C}_2^{\text{sys}}) \\
&= (\mathcal{C}_1^{\text{obj}} \bullet (\mathcal{C}_1^{\text{sys}})^{-1}) \bullet (\mathcal{C}_2^{\text{obj}} \bullet (\mathcal{C}_2^{\text{sys}})^{-1}) && \text{([129], Section 3.1)} \\
&= (\mathcal{C}_1^{\text{obj}} \bullet \mathcal{C}_2^{\text{obj}}) \bullet \Big((\mathcal{C}_1^{\text{sys}})^{-1}) \bullet ((\mathcal{C}_2^{\text{sys}})^{-1})\Big) \\
&= (\mathcal{C}_1^{\text{obj}} \bullet \mathcal{C}_2^{\text{obj}}) \bullet \left(\mathcal{C}_1^{\text{sys}} \parallel \mathcal{C}_2^{\text{sys}}\right)^{-1} && \text{([121], Table 6.1)} \\
&= (\mathcal{C}_1^{\text{obj}} \bullet \mathcal{C}_2^{\text{obj}})/\left(\mathcal{C}_1^{\text{sys}} \parallel \mathcal{C}_2^{\text{sys}}\right) \\
&= (\mathcal{C}_1^{\text{obj}} \parallel \mathcal{C}_2^{\text{obj}})/\left(\mathcal{C}_1^{\text{sys}} \parallel \mathcal{C}_2^{\text{sys}}\right), && (A_1^{\text{obj}} = A_2^{\text{obj}} = \top))
\end{aligned}$$

which is the list $(\mathcal{C}_1^{\text{obj}} \parallel \mathcal{C}_2^{\text{obj}}, \mathcal{C}_1^{\text{sys}} \parallel \mathcal{C}_2^{\text{sys}})$. □

The merged test constract is constructed from parallel compositions of the objective contracts and system contracts, separately. Composition of the system contracts should be interpreted as specifications on the subsystems. The composition of test structures is defined as:

Definition 5.11. Given test structures $\mathfrak{t}_i = (\mathcal{C}_i^{\text{obj}}, \mathcal{C}_i^{\text{sys}})$ for $i \in \{1, 2\}$, we define their *composition* $\mathfrak{t}_1 \parallel \mathfrak{t}_2$ as

$$(\mathcal{C}_1^{\text{obj}}, \mathcal{C}_1^{\text{sys}}) \parallel (\mathcal{C}_2^{\text{obj}}, \mathcal{C}_2^{\text{sys}}) = (\mathcal{C}_1^{\text{obj}} \parallel \mathcal{C}_2^{\text{obj}}, \mathcal{C}_1^{\text{sys}} \parallel \mathcal{C}_2^{\text{sys}}).$$

Example 5.3 (Car Pedestrian)**.** Recall the car-pedestrian example from Chapter 2, which we will adopt with slight modifications. Consider a test environment shown in Figure 5.10 consisting of a single lane road, a crosswalk with a pedestrian, and different visibility conditions. The system under test is an autonomous car driving on the road which must stop for the pedestrian at the crosswalk no matter the visibility conditions. The first test objective under low visibility is formalized by a test engineer as:

$$\mathcal{C}_1^{\text{obj}} = (\top, \quad \varphi_{\text{init}}^{\text{car}} \wedge \Box \varphi_{\text{low}}^{\text{vis}} \wedge \Diamond \varphi_{\text{cw}}^{\text{ped}} \wedge (\varphi_{\text{cw}}^{\text{ped}} \rightarrow \Diamond \varphi_{\text{cw}}^{\text{stop}})),$$

where $\varphi_{\text{low}}^{\text{vis}} := \varphi^{\text{vis}} \models \text{low}$, denotes low visibility conditions, $\varphi_{\text{init}}^{\text{car}}$ the initial conditions of the car (position x_{car} and velocity v_{car}), $\varphi_{\text{cw}}^{\text{ped}}$ denotes the pedestrian being on the crosswalk, and $\varphi_{\text{cw}}^{\text{stop}} := x_{\text{car}} \leq C_{\text{cw}-1} \wedge v_{\text{car}} = 0$ the stopping maneuver at one cell before the crosswalk cell C_{cw}. Similarly, the test objective contract under high visibility is also given as:

$$\mathcal{C}_2^{\text{obj}} = \Big(\top, \quad \varphi_{\text{init}}^{\text{car}} \wedge \Box \varphi_{\text{high}}^{\text{vis}} \wedge \Diamond \varphi_{\text{cw}}^{\text{ped}} \wedge (\varphi_{\text{cw}}^{\text{ped}} \rightarrow \Diamond \varphi_{\text{cw}}^{\text{stop}})\Big),$$

where $\varphi_{\text{high}}^{\text{vis}} := \varphi^{\text{vis}} \models \text{high}$ represents high visibility test environment. Finally, the dynamics of braking when a pedestrian is detected is given by the contract,

$$\mathcal{C}_3^{\text{obj}} = (\top, \quad \exists k : (v_{\text{car}} = V_{\max} \wedge x_{\text{car}} = C_k) \rightarrow \Diamond \varphi_{k+d}^{\text{stop}}),$$

where the car is required to drive at specified speed $V_{\max}$ in an arbitrary cell C_k, and then eventually stop within the stopping distance d specified by the user; $\varphi_{k+d}^{\text{stop}}$ specifies that the car must stop at or before cell C_{k+d}. Note that we assume that the stopping distance d is large enough such that the car moving at maximum speed can come to a stop within d steps. This test objective specifies the stopping requirement on the car irrespective of the environment. Note that none of the test objective contracts reason over the system's capabilities to detect a pedestrian, only requiring that the system needs to stop at the crosswalk if a pedestrian is in it. This is important since we do not want the test objective contract to have guarantees that depend on the performance of individual components (e.g., perception) of the system.

Requirements on the system are provided by the system designers and test engineers. Each of the following system contracts assume that the environment is safe (e.g., the environment agents will not adversarially crash into the car). This is denoted as $A^{\mathsf{sys}} = \square\varphi^{\text{ped}}_{\text{dyn}} \wedge \square\varphi^{\text{vis}}_{\text{dyn}}$, where $\varphi^{\text{ped}}_{\text{dyn}}$, and $\varphi^{\text{vis}}_{\text{dyn}}$ denote the dynamics of the pedestrian, and the visibility conditions, respectively.

$$\mathcal{C}_1^{\mathsf{sys}} = \Big(A^{\mathsf{sys}}, \quad \square\varphi^{\text{car}}_{\text{dyn}} \wedge \square\left(\varphi^{\text{vis}}_{\text{low}} \rightarrow v \leq V_{\text{low}}\right) \wedge \\ \square\left(\texttt{detectable}^{\text{ped}}_{\text{low}} \rightarrow \diamondsuit\, \varphi^{\text{stop}}_{\text{ped}}\right) \vee \neg A^{\mathsf{sys}}\Big),$$

where $\varphi^{\text{car}}_{\text{dyn}}$, describes the dynamics of the car. V_{low} is the maximum permissible speed of the car under low-visibility conditions. The expression $\texttt{detectable}^{\text{ped}}_{\text{low}}$ describes the pedestrian being in a buffer zone in front of the car, and is formally defined as,

$$\texttt{detectable}^{\text{ped}}_{\text{low}} := x_{\text{car}} + dist^{\text{low}}_{\text{min}} \leq x_{\text{ped}} \leq x_{\text{car}} + dist^{\text{low}}_{\text{max}},$$

where $dist^{\text{low}}_{\text{min}}$ is the minimum distance for the car to reach a full stop, and $dist^{\text{low}}_{\text{max}}$ is the maximum distance at which the car can detect a pedestrian in low visibility conditions. The second system objective contract describes driving in high visibility conditions:

$$\mathcal{C}_2^{\mathsf{sys}} = \Big(A^{\mathsf{sys}}, \quad \square\varphi^{\text{car}}_{\text{dyn}} \wedge \square\left(\varphi^{\text{vis}}_{\text{high}} \rightarrow v \leq V_{\text{max}}\right) \wedge \\ \square\left(\texttt{detectable}^{\text{ped}}_{\text{high}} \rightarrow \diamondsuit\, \varphi^{\text{stop}}_{\text{ped}}\right) \vee \neg A^{\mathsf{sys}}\Big),$$

where V_{max} is the maximum speed, and the expression $\texttt{detectable}^{\text{ped}}_{\text{high}}$, defined similarly to $\texttt{detectable}^{\text{ped}}_{\text{low}}$, denotes the pedestrian being detectable in the 'buffer' zone for high visibility conditions. The third system objective contract for the dynamics of the car,

$$\mathcal{C}_3^{\mathsf{sys}} = \Big(A^{\mathsf{sys}}, \quad \square\varphi^{\text{car}}_{\text{dyn}} \vee \neg A^{\mathsf{sys}}\Big),$$

where $\varphi^{\text{car}}_{\text{dyn}}$ describes the dynamics of the car, including the distance to come to a full stop as a function of car speed. For each pair of system and test objectives, a test can synthesized for the specification constructed by equation (5.15). We can find combinations of test structures $\mathfrak{t}_i = (\mathcal{C}_i^{\text{obj}}, \mathcal{C}_i^{\text{sys}})$ that can be executed instead of individual tests. Consider the combined test structure $\mathfrak{t} = \mathfrak{t}_2 \parallel \mathfrak{t}_3$. The corresponding combined test objective contract $\mathcal{C}^{\text{obj}}$ is:

$$\begin{aligned} \mathcal{C}^{\text{obj}} = \mathcal{C}_2^{\text{obj}} \parallel \mathcal{C}_3^{\text{obj}} = (\top, \quad & \varphi_{\text{init}}^{\text{car}} \wedge \square \varphi_{\text{high}}^{\text{vis}} \wedge \diamondsuit\, \varphi_{\text{cw}}^{\text{ped}} \wedge \varphi_{\text{cw}}^{\text{ped}} \rightarrow \diamondsuit\, \varphi_{\text{cw}}^{\text{stop}} \wedge \\ & \exists k : (v_{\text{car}} = V_{\max} \wedge x_{\text{car}} = C_k) \rightarrow \diamondsuit\, \varphi_{k+d}^{\text{stop}}). \end{aligned} \tag{5.16}$$

Likewise, the combined system objective contract is:

$$\mathcal{C}^{\text{sys}} = \mathcal{C}_2^{\text{sys}} \parallel \mathcal{C}_3^{\text{sys}} = (A^{\text{sys}} \cup \neg(G_2^{\text{sys}} \cap G_3^{\text{sys}}), \quad G_2^{\text{sys}} \cap G_3^{\text{sys}}).$$

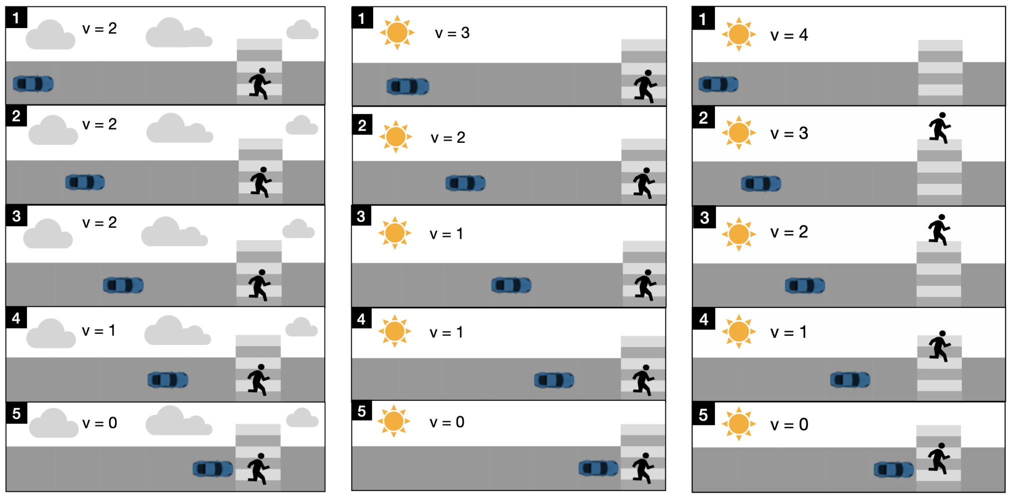
The term $\neg(G_2^{\text{sys}} \cap G_3^{\text{sys}})$ can be removed from the assumptions of $\mathcal{C}^{\text{sys}}$ to relax the system objective contract for ensuring that the assumptions conform to those required by Definition 5.8. Therefore, the system objective contract becomes:

$$\begin{aligned} \mathcal{C}^{\text{sys}} =& (A^{\text{sys}}, \quad \square \varphi_{\text{dyn}}^{\text{car}} \wedge \square\, (\varphi_{\text{high}}^{\text{vis}} \rightarrow v \leq V_{\max}) \wedge \\ & \square(\texttt{detectable}_{\text{high}}^{\text{ped}} \rightarrow \diamondsuit\, \varphi_{\text{ped}}^{\text{stop}}) \vee \neg A^{\text{sys}}). \end{aligned} \tag{5.17}$$

Equations (5.16) and (5.17) result in a test structure $\mathfrak{t} = (\mathcal{C}^{\text{obj}}, \mathcal{C}^{\text{sys}})$, and we can implement test environments to satisfy equation (5.15) with respect to the test structure $\mathfrak{t}$. The combined test structure $\mathfrak{t} = (\mathcal{C}^{\text{obj}}, \mathcal{C}^{\text{sys}}$ results in a test which requires the car to decelerate from $V_{\max}$ in high visibility conditions and come to a stop before the crosswalk.

To determine when two test structures can be combined, we need to check if the combined test objective and the combined test structure are satisifiable. Therefore, two test structures cannot be combined if either of these conditions is untrue. For example, the combination $\mathfrak{t}_1 \parallel \mathfrak{t}_2$ is not possible because composition of the constituent test objectives $\mathcal{C}_1^{\text{obj}} \parallel \mathcal{C}_2^{\text{obj}}$ has an empty set of guarantees. This is because $\square \varphi_{\text{low}}^{\text{vis}}$ and $\square \varphi_{\text{high}}^{\text{vis}}$ is disjoint since visibility cannot be both *high* and *low* at the same time. Now consider test structures $\mathfrak{t}_1$ and $\mathfrak{t}_3$; while these test structures can be composed to have a non-empty set of guarantees, the resulting test structure is not realizable by any test environment. The composition $\mathfrak{t}_1 \parallel \mathfrak{t}_3$ results in a test structure with the test objective requiring a maximum speed of $V_{\max}$, but with the system constrained to a maximum speed of $V_{\text{low}} < V_{\max}$ in low visibility conditions. Therefore, $G^{\text{sys}} \cap G^{\text{obj}} = \emptyset$, and both the system and test objectives cannot be satisfied in a single trace of the system.

Figure 5.10 illustrates manually constructed test executions that satisfy test contracts corresponding to $\mathfrak{t}_1, \mathfrak{t}_2$, and $\mathfrak{t}_2 \parallel \mathfrak{t}_3$, respectively. The car controller is implemented on a discrete grid world; at some positive speed v the car moves forward by



(a) Low visibility with a stationary pedestrian. (b) High visibility with a stationary pedestrian. (c) High visibility with a reactive pedestrian.

Figure 5.10: Test execution snapshots of the car stopping for a pedestrian. Figure 5.10a shows a test execution satisfying $\mathcal{C}_1^{\text{tester}}$, Figure 5.10b satisfies $\mathcal{C}_2^{\text{tester}}$ and Figure 5.10c satisfies $\mathcal{C}_2^{\text{tester}}$ and $\mathcal{C}_3^{\text{tester}}$.

v cells. At each time step, the car can choose to continue at the same speed or to accelerate or decelerate.

In the low visibility setting, the car can drive at a maximum speed of $v = 2$ and it can detect a pedestrian up to two cells away as illustrated in Figure 5.10a. The car is able to detect the pedestrian and come to a full stop in front of the crosswalk. In a high visibility setting, the car can drive at a maximum speed of $v_{\text{max}} = 4$, and it can detect the pedestrian up to 5 cells ahead. In Figure 5.10b, we can see that the pedestrian is detected and the car slows down gradually until is reaches the cell in front of the crosswalk. The test for the combined test structure $\mathfrak{t} = \mathfrak{t}_2 \parallel \mathfrak{t}_3$ is shown in Figure 5.10c, where we see the pedestrian entering the crosswalk in high visibility conditions when the car is driving at its maximum speed of $v = 4$ and is 10 cells away from the crosswalk. This test execution now checks the test objective of detecting a pedestrian in high visibility conditions and executing the braking maneuver with the desired constant deceleration from its maximum speed down to zero. ■

5.8 Comparing Tests

A test campaign is a set of tests, each characterized by a test structure. Choosing a test campaign out of several possibilities requires a principled approach to comparing test campaigns. A more refined test campaign is preferable since the system will be tested for a more refined set of test objectives and possibly for a more stringent set of system specifications. Let $\mathfrak{t}_i = (\mathcal{C}_i^{\text{obj}}, \mathcal{C}_i^{\text{sys}})$ be test structures for $1 \leq i \leq n$. When generating tests for $\mathfrak{t}_i$, we want to ensure that our test execution satisfies the constraints set out by $\mathcal{C}_i^{\text{obj}}$ in the context of system behaviors defined by $\mathcal{C}_i^{\text{sys}}$. As seen in Section 5.6, the tester contract can be computed using the quotient operator. We characterize a test campaign, $\texttt{TC} = \{\mathfrak{t}_i\}_{i=1}^n$, as a finite list of test structures specified by the test engineer. Definition 5.12 allows us to generate a single test structure from a test campaign.

Definition 5.12. Given a test campaign $\texttt{TC} = \{\mathfrak{t}_i\}_{i=1}^n$, the *test structure generated by this campaign*, denoted $\tau(\texttt{TC})$, is

$$\tau(\texttt{TC}) = \mathfrak{t}_1 \parallel \ldots \parallel \mathfrak{t}_n.$$

A notion of ordering between test structures is necessary for establishing an ordering of test campaigns. This order is also important for defining the split of a test into unit tests, as we shall see later.

Definition 5.13. The test structure $(\mathcal{C}_1^{\text{obj}}, \mathcal{C}_1^{\text{sys}})$ *refines* the structure $(\mathcal{C}_2^{\text{obj}}, \mathcal{C}_2^{\text{sys}})$, written $(\mathcal{C}_1^{\text{obj}}, \mathcal{C}_1^{\text{sys}}) \leq (\mathcal{C}_2^{\text{obj}}, \mathcal{C}_2^{\text{sys}})$, if contract refinement occurs element-wise, i.e., if $\mathcal{C}_1^{\text{sys}} \leq \mathcal{C}_2^{\text{sys}}$ and $\mathcal{C}_1^{\text{obj}} \leq \mathcal{C}_2^{\text{obj}}$.

Finally, the order of refinement between test campaigns can be defined as follows. In a refined test campaign $\texttt{TC}$ of $\texttt{TC}'$, the system and test objective contracts of the test structure corresponding to $\texttt{TC}$ are more refined. That is, the test objective handles a larger set of system behaviors with stricter requirements (i.e., more constraints) on what the desired test execution should look like. In addition, the system might potentially be required to satisfy stricter guarantees on its behavior under a larger set of assumptions. For these reasons, it is preferable to choose a refined test campaign.

Definition 5.14. Given two test campaigns $\texttt{TC}$ and $\texttt{TC}'$, we say that $\texttt{TC} \leq \texttt{TC}'$ if $\tau(\texttt{TC}) \leq \tau(\texttt{TC}')$.

5.9 Splitting Tests

In this section, we explore the notion of splitting test structures. One of our motivations for doing this is failure diagnostics, in which we wish to look for root causes of a system-level test failure. To split test structures, we look for the existence of a quotient — see [129]. Suppose there exists a test structure $\mathfrak{t}$ that we want to split, and suppose one of the pieces of this decomposition, $\mathfrak{t}_1$, is given to us. Our objective is to find $\mathfrak{t}_2$ such that $\mathfrak{t}_1 \parallel \mathfrak{t}_2 \leq \mathfrak{t}$. The following result tells how to compute the optimum $\mathfrak{t}_2$. This optimum receives the name *quotient of test structures*.

Proposition 5.2. Let $\mathfrak{t} = (\mathcal{C}^{\text{obj}}, \mathcal{C}^{\text{sys}})$ and $\mathfrak{t}_1 = (\mathcal{C}_1^{\text{obj}}, \mathcal{C}_1^{\text{sys}})$ be two test structures and let $\mathfrak{t}_q = (\mathcal{C}^{\text{obj}}/\mathcal{C}_1^{\text{obj}}, \mathcal{C}^{\text{sys}}/\mathcal{C}_1^{\text{sys}})$. For any test structure $\mathfrak{t}_2 = (\mathcal{C}_2^{\text{obj}}, \mathcal{C}_2^{\text{sys}})$, we have

$$\mathfrak{t}_2 \parallel \mathfrak{t}_1 \leq \mathfrak{t} \quad \text{if and only if} \quad \mathfrak{t}_2 \leq \mathfrak{t}_q.$$

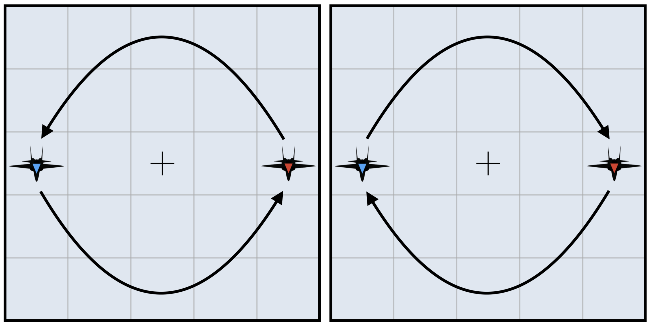
We say that $\mathfrak{t}_q$ is the quotient of $\mathfrak{t}$ by $\mathfrak{t}_1$, and we denote it as $\mathfrak{t}/\mathfrak{t}_1$.

Proof. $\mathfrak{t}_2 \leq \mathfrak{t}_q \quad \Leftrightarrow \quad \mathcal{C}_2^{\text{sys}} \leq \mathcal{C}^{\text{sys}}/\mathcal{C}_1^{\text{sys}}$ and $\mathcal{C}_2^{\text{obj}} \leq \mathcal{C}^{\text{obj}}/\mathcal{C}_1^{\text{obj}} \Leftrightarrow \quad (\mathcal{C}_2^{\text{obj}} \parallel \mathcal{C}_1^{\text{obj}}, \mathcal{C}_2^{\text{sys}} \parallel \mathcal{C}_1^{\text{sys}}) \leq (\mathcal{C}^{\text{obj}}, \mathcal{C}^{\text{sys}}) \quad \Leftrightarrow \quad \mathfrak{t}_2 \parallel \mathfrak{t}_1 \leq \mathfrak{t}.$ $\square$

Remark: The method of constructing the quotient test structure in Proposition 5.2 involves taking the quotient of the system contracts as well as the test objectives, meaning that we remove a subsystem from the overall system, and remove a part of the test objective. Depending on the use case, we can consider two further situations, where we can define the test structure $\mathfrak{t}_1$ such that: i) only removing a subsystem from the overall system, which gives the quotient $\mathfrak{t}_q = (\mathcal{C}^{\text{obj}}, \mathcal{C}^{\text{sys}}/\mathcal{C}_1^{\text{sys}})$; and ii) only separating a part of the test objective: $\mathfrak{t}_q = (\mathcal{C}^{\text{obj}}/\mathcal{C}_1^{\text{obj}}, \mathcal{C}^{\text{sys}})$. The quotient test structures of type (i) could be useful in adding further test harnesses to monitor sub-systems under for the same test objective, and test structures of type (ii) could be useful in monitoring overall system behavior under a more unit test objective. In future work, we will study automatically choosing the relevant quotient test structure for specific use cases.

Label	Formula
φ_{setgoal}	$\square(x_{init,i} = R_1 \rightarrow x_{g,i} = R_2) \wedge \square(x_{init,i} = R_2 \rightarrow x_{g,i} = R_1)$
$\texttt{execute}^{\text{ccw}}_{\text{swap}}(a_i)$	$\diamondsuit(x_i = g_i) \wedge \square(x_i = g_i \rightarrow \bigcirc(x_i = g_i)) \wedge \square\varphi^{\text{ccw}}_{\text{traj,i}}$
$\texttt{execute}^{\text{cw}}_{\text{swap}}(a_i)$	$\diamondsuit(x_i = g_i) \wedge \square(x_i = g_i \rightarrow \bigcirc(x_i = g_i)) \wedge \square\varphi^{\text{cw}}_{\text{traj,i}}$
$\varphi^{\text{cw}}_{\text{swap,i}}$	$\square(\texttt{directive}^{\text{cw}}_{\text{swap}}(a_i) \rightarrow \diamondsuit(x_i = g_i))$
$\varphi^{\text{ccw}}_{\text{swap,i}}$	$\square(\texttt{directive}^{\text{ccw}}_{\text{swap}}(a_i) \rightarrow \diamondsuit(x_i = g_i))$
φ^{cw}	$\diamondsuit\, \texttt{directive}^{\text{cw}}_{\text{swap}}(a_1) \wedge \diamondsuit\, \texttt{directive}^{\text{cw}}_{\text{swap}}(a_2)$
φ^{ccw}	$\diamondsuit\, \texttt{directive}^{\text{ccw}}_{\text{swap}}(a_1) \wedge \diamondsuit\, \texttt{directive}^{\text{ccw}}_{\text{swap}}(a_2)$

Table 5.1: Subformulas for constructing G^{sys} and G^{obj}.


(a) Executions satisfying the original test structure.

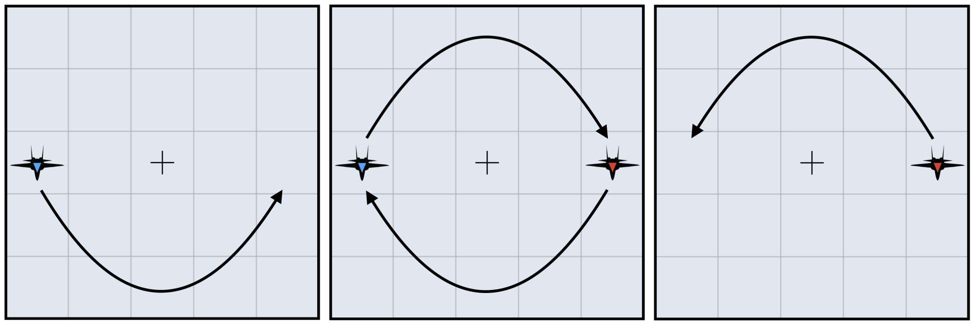

(b) Left: Given unit test. Center and right: Possible executions for the split test.

Figure 5.11: Front view of test executions satisfying the original test structure and the split test structure.

Example 5.4. Consider two aircraft, a_1 and a_2, flying parallel to each other undergoing a formation flying test shown in Figure 5.11a where two aircraft need to swap positions longitudinally in a clockwise or counterclockwise spiral motion. Assume that during this test execution a system-level failure has been observed, but it is unknown which aircraft is responsible for the failure during which stage of the maneuver. We will make use of our framework to split test structures to help identify the subsystem responsible for the failure.

The aircraft communicate with a centralized computer that issues waypoint directives to each aircraft in a manner consistent to the directives issued to other aircraft to ensure that there are no collisions. The dynamics of aircraft a_i on the gridworld is specified by G_i^{dyn}, and the safety or no collision requirement on all aircraft is given in G^{safe}. The swap requirement, G_i^{swap}, specifies the maneuver that each aircraft must take in the event that a directive is issued.

$$\begin{aligned} G_i^{\text{swap}} =& \square(\texttt{directive}_{\text{swap}}^{\text{cw}}(a_i) \rightarrow \texttt{execute}_{\text{swap}}^{\text{cw}}(a_i)) \wedge \\ & \square(\texttt{directive}_{\text{swap}}^{\text{ccw}}(a_i) \rightarrow \texttt{execute}_{\text{swap}}^{\text{ccw}}(a_i)). \end{aligned}$$

For example, in the case of a counter-clockwise swap directive issued to aircraft a_1 starting in region R_1, the aircraft must eventually reach the counter-clockwise swap goal, R_2, by traveling in the counter-clockwise direction, and upon reaching the goal must stay there as long as no new directive is issued. These maneuvers are specified in the `execute` subformulas in Table 5.1. The swap goals, g_i, for the aircraft are determined by their respective positions, $x_{\text{init,i}}$, when the directives are issued (see Table 5.1).

In this example, the tester fills the role of the supervisor. If the tester decides on all aircraft swapping clockwise, then the clockwise directives to each aircraft will be issued: $\varphi^{\text{cw}} = \Diamond\, \texttt{directive}^{\text{cw}}_{\text{swap}}(a_1) \wedge \Diamond\, \texttt{directive}^{\text{cw}}_{\text{swap}}(a_2)$. Similarly, φ^{ccw} denotes the eventual issue of counter-clockwise swap directives to both aircraft. All the temporal logic formulas required to construct the test structure associated with this example are summarized in Table 5.1. Moreover, no new directives are issued until all current directives are issued and all aircraft have completed the swap executions corresponding to the current directives (labeled as $G^{\texttt{dir}}_{\text{limit}}$). Finally, the aircraft are never issued conflicting swap directions — all aircraft are instructed to go clockwise or counterclockwise (labeled as $G^{\texttt{dir}}_{\text{safe}}$). For simplicity, we choose not to write out $G^{\texttt{dir}}_{\text{limit}}$ and $G^{\texttt{dir}}_{\text{safe}}$ in their extensive forms. Thus, the requirements for the system under test are as follows:

$$\mathcal{C}^{\text{sys}} = (A^{\text{sys}}, G^{\text{sys}}) = (G^{\texttt{dir}}_{\text{limit}} \wedge G^{\texttt{dir}}_{\text{safe}},\, G^{\text{safe}} \wedge \bigwedge_i G^{\text{swap}}_i \wedge G^{\text{dyn}}_i). \tag{5.18}$$

That is, assuming that the supervisor issues consistent directives, and issues new directives only when all aircraft have completed the executions corresponding to the current round of directives, the aircraft system is required to guarantee safety and successful execution of the swap maneuver corresponding to the current directive. If we were to write the system requirements for a single aircraft, the corresponding contract would be similar:

$$\mathcal{C}^{\text{sys}}_i = (A^{\text{sys}}_i, G^{\text{sys}}_i) = (G^{\texttt{dir}}_{\text{limit}} \wedge G^{\texttt{dir}}_{\text{safe}},\, G^{\text{swap}}_i \wedge G^{\text{dyn}}_i). \tag{5.19}$$

$$\begin{aligned}\mathcal{C}^{\text{obj}} &= (\top, G^{\text{obj}}),\\ G^{\text{obj}} &= \Box((\texttt{directive}^{cw}_{\text{swap}}(a_1) \wedge \texttt{directive}^{cw}_{\text{swap}}(a_2)) \vee (\texttt{directive}^{ccw}_{\text{swap}}(a_2)\\ &\quad \wedge \texttt{directive}^{ccw}_{\text{swap}}(a_1)) \rightarrow \Diamond(x_1 = R_2 \wedge x_2 = R_1)).\end{aligned} \tag{5.20}$$

Observe that G^{obj} represents the tester issuing either clockwise or counter-clockwise swap directives. One of the unit tests is to the have the aircraft a_1 starting at

$x_{init,1} = R_1$ (and as a result, $x_g = R_2$) get the counter-clockwise swap directive to reach $x_g = R_2$. The corresponding unit test structure $\mathfrak{t}_1 = (\mathcal{C}_1^{\text{obj}}, \mathcal{C}_1^{\text{sys}})$ can be written as follows:

$$\mathcal{C}_1^{\text{obj}} = (\top, G_1^{\text{obj}}) = (\top, \Diamond\, \texttt{directive}_{\text{swap}}^{\text{ccw}}(a_1)) \tag{5.21}$$

$$\mathcal{C}_1^{\text{sys}} = (A_1^{\text{sys}}, G_1^{\text{sys}}) = (G_{\text{limit}}^{\texttt{dir}} \wedge G_{\text{safe}}^{\texttt{dir}},\ G_1^{\text{swap}} \wedge G_1^{\text{dyn}}). \tag{5.22}$$

Following Proposition 5.2, the second unit test structure can be derived by separately applying the quotient operator on the test objectives and the system contract. Applying the quotient on the test objective, we substitute $\top$ for the assumptions to simplify, and we refine the quotient contract $\mathcal{C}^{\text{obj}}/\mathcal{C}_1^{\text{obj}}$ by replacing its assumptions with $\top$:

$$\begin{aligned}\mathcal{C}^{\text{obj}}/\mathcal{C}_1^{\text{obj}} &= (A \cap G_1^{\text{obj}}, G \cap A_1^{\text{obj}} \cup \neg(A \cap G_1^{\text{obj}}))\\ &= (G_1^{\text{obj}}, G \cup \neg G_1^{\text{obj}}) \geq (\top, G^{\text{obj}} \cup \neg G_1^{\text{obj}}).\end{aligned}$$

Designer input is important for refining this contract resulting from applying the quotient; a similar observation has been documented for quotient operators in previous work [123]. Domain knowledge can be helpful in refining the contracts. Using $\neg G_1^{\text{obj}}$ as context, the contract $(\top, G^{\text{obj}} \cup \neg G_1^{\text{obj}})$ can be simplified to $(\top, \neg G_1^{\text{obj}} \vee \varphi_1 \vee \varphi_2)$, where $\varphi_1 = (\Diamond\, \texttt{directive}_{\text{swap}}^{\text{ccw}}(a_2) \wedge \neg\varphi^{\text{cw}})$ and $\varphi_2 = \varphi^{\text{cw}} \wedge \neg\varphi^{\text{ccw}}$. Then, $\neg G_1^{\text{obj}}$ is discarded and the test objective of the second unit test can be defined as a refinement of this simplified contract arising from the quotient:

$$\mathcal{C}_{a_2}^{\text{obj}} = (\top, \varphi_1 \vee \varphi_2) \leq (\top, \neg G_1^{\text{obj}} \vee \varphi_1 \vee \varphi_2). \tag{5.23}$$

In equation (5.23), there are two types of test executions that would be the unit contract obtained by applying the quotient operator: i) A counter-clockwise directive is issued to aircraft a_2 and no clockwise directives are issued to either aircraft, or ii) Both aircraft are issued clockwise directives and no counter-clockwise directives. Note that φ_1 and φ_2 cannot be implemented in the same test by construction. Finally, the unit system contract can also by found by applying the quotient operator:

$$\begin{aligned}\mathcal{C}^{\text{sys}}/\mathcal{C}_1^{\text{sys}} =& (A^{\text{sys}} \cap G_1^{\text{sys}}, G^{\text{sys}} \cap A_1^{\text{sys}} \cup \neg(A^{\text{sys}} \cap G_1^{\text{sys}}))\\ =& (G_{\text{limit}}^{\text{dir}} \wedge G_{\text{safe}}^{\text{dir}} \wedge G_1^{\text{swap}} \wedge G_1^{\text{dyn}}, (G^{\text{safe}} \wedge G_2^{\text{swap}} \wedge G_2^{\text{dyn}})\\ & \vee \neg(G_1^{\text{swap}} \wedge G_1^{dyn} \wedge G_{\text{limit}}^{\text{dir}} \wedge G_{\text{safe}}^{\text{dir}}))\\ =& (G_{\text{limit}}^{\text{dir}} \wedge G_{\text{safe}}^{\text{dir}} \wedge G_1^{\text{swap}} \wedge G_1^{\text{dyn}}, (G^{\text{safe}} \wedge G_2^{\text{swap}} \wedge G_2^{\text{dyn}})).\end{aligned} \tag{5.24}$$

We refine the quotient contract by keeping the assumptions to be *true*.

$$\mathcal{C}_{a_2}^{\text{sys}} = (\top, \neg(G_1^{\text{comm}} \wedge G_1^{\text{dyn}}) \vee (G^{\text{safe}} \wedge \bigwedge_i G_i^{\text{comm}} \wedge G_i^{\text{dyn}})) \tag{5.25}$$

$$= (\top, \neg(G_1^{\text{comm}} \wedge G_1^{\text{dyn}}) \vee (G^{\text{safe}} \wedge G_2^{\text{dyn}} \wedge G_2^{\text{dyn}})). \tag{5.26}$$

Remark: Observe that equation (5.24) carries the swap and dynamics requirements of aircraft a_1 in its assumptions. Since we choose to separate aircraft a_1 from the overall aircraft system, this quotient contract can be satisfied by making aircraft a_1 a part of the tester. For a test execution of $\mathfrak{t}_2$, the tester can choose to keep aircraft a_1 as a part of the test harness for the operational test involving aircraft a_2, or choose to not deploy a_1 during the test execution. Assuming that aircraft a_1 satisfies its swap requirements, and that the supervisor satisfies the requirements on the directives, $G_{\text{limit}}^{\text{dir}}$ and $G_{\text{safe}}^{\text{dir}}$, then this unit system contract guarantees that the aircraft a_2 satisfies its swap requirements, and all the aircraft together satisfy the safety requirements.

The system requirement $\mathcal{C}_2^{\text{sys}} = \mathcal{C}^{\text{sys}}/\mathcal{C}_1^{\text{sys}}$ and the test objective together result in the following possible tester specifications,

$$\begin{aligned}\mathcal{C}_{\varphi_1}^{\text{tester}} = \Big(&G^{\text{safe}} \wedge G_2^{\text{swap}} \wedge G_2^{\text{dyn}},\ G_{\text{limit}}^{\text{dir}} \wedge G_{\text{safe}}^{\text{dir}} \wedge G_1^{\text{swap}} \wedge G_1^{\text{dyn}} \\ &\wedge \Diamond\, \texttt{directive}_{\text{swap}}^{\text{ccw}}(a_2) \wedge \neg\varphi^{\text{cw}}\Big).\end{aligned} \tag{5.27}$$

$$\begin{aligned}\mathcal{C}_{\varphi_2}^{\text{tester}} = \Big(&G^{\text{safe}} \wedge G_2^{\text{swap}} \wedge G_2^{\text{dyn}},\ G_{\text{limit}}^{\text{dir}} \wedge G_{\text{safe}}^{\text{dir}} \wedge G_1^{\text{swap}} \wedge G_1^{\text{dyn}} \\ &\wedge \Diamond\, \texttt{directive}_{\text{swap}}^{\text{cw}}(a_1) \wedge \Diamond\, \texttt{directive}_{\text{swap}}^{\text{cw}}(a_2) \wedge \neg\varphi^{\text{ccw}}\Big).\end{aligned} \tag{5.28}$$

From equation (5.27), we see that the tester does not require aircraft a_1 for any dynamic maneuvers, so it need not be deployed. In equation (5.28), even though aircraft a_1 would be a part of the test harness, it needs to be deployed for the tester contract, $\mathcal{C}_{\varphi_2}^{\text{tester}}$, to be satisfied. These tests resulting from the quotient test structure will help with determining the source of the failure that arose in the more complex test. ■

5.10 Conclusions and Future Work

In this chapter, we covered how assume-guarantee contract operations can aid in merging, comparing, and splitting specifications that define individual tests. While the previous chapter synthesized a test environment and strategy from the system abstraction, here we assume that such an environment is already synthesized. The ideas in this chapter are preliminary, and further research is needed for practical and large-scale construction of test campaigns while exploiting notions of compositionality. Yet, our framework based on the mathematical foundations of assume-guarantee contracts provides a useful formalism to reason about sets of behaviors that are covered by a test objective. An interesting direction of future work is to investigate how *coverage* arguments can be built from synthezing tests in this manner. Given a set of behaviors covered by a test structure, one could optimize for the worst-case test strategy using a robustness metric, preliminary versions of which were illustrated earlier in the chapter. This can be significantly expanded to systems with dynamics and specifications with timing constraints. Additionally, we would need to derive a guarantee that evaluations and conclusions from running the most difficult test for a test contract determines with high probability the success of possible other test executions in the same test contract.

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.