Reasoning over Test Specifications using Assume-Guarantee Contracts

Apurva Badithela*[1], Josefine B. Graebener*[1], Inigo Incer*[1,2], and Richard M. Murray[1]

[1]California Institute of Technology, USA
[2]University of California, Berkeley, USA
{apurva, jgraeben, inigo}@caltech.edu, murray@cds.caltech.edu

Abstract. We establish a framework to reason about test campaigns described formally. First, we introduce the notion of a test structure — an object 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. Secondly, we use the algebra of assume-guarantee contracts to reason about constructing tester specifications, comparing test structures and test campaigns, and combining and splitting test structures. Using the composition operator, we characterize the conditions on the constituent tester specifications and test objectives for feasibly combining test structures. We illustrate the different applications of the quotient operator to split the test objective, the system into subsystems, or both. 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. We anticipate that reasoning over test specifications would aid in generating optimal test campaigns.

Keywords: Testing Autonomous Systems · Assume-Guarantee Reasoning · Contracts

1 Introduction

Rigorous test campaigns have to be designed, implemented, and executed to aid in certification of safety-critical autonomous systems [30]. Testing complex autonomous systems is a key challenge that remains to be solved to achieve human confidence in the system's behavior in a real world setting, ranging from autonomous driving to military and space missions and beyond [9, 12, 19, 32]. Currently, test campaigns are designed by test engineers, who rely on their product know-how and experience, and the resulting test scenarios are fine-tuned using simulation-based falsification to find the desired test execution [4, 15]. Executing these test campaigns in real-world settings can be prohibitive for some systems, such as those involved in space missions, due to the immense cost and impracticality of the tests. Thus, carefully choosing the constituent tests of

* These authors contributed equally to this work.

a test campaign is necessary. Instead of the test engineer designing the entire test manually, we require them to specify the objective of the test. For example, an autonomous car required to operate safely at a busy T-intersection, while two tester cars arrive at the same time as the system, could be the test objective. Prior work in [2, 17], takes this high-level input from the test engineer, and provides algorithms for synthesizing test environments and tester strategies that meet the test objective.

We provide a brief overview of prior work that has used assume-guarantee reasoning for testing safety-critical systems. In [11], assume-guarantee contracts have been used for compositional verification of system models, and verified components have been reused in the certification process for new system architectures. In [7, 16], the authors use assume-guarantee reasoning to (i) generate component-level tests that convey system-level information, (ii) limit the scope of component testing by focusing on tests that meet a component's assumptions, and (iii) perform predictive testing. In [8], assume-guarantee reasoning is used in the context of input output conformance testing [31]. Assume-guarantee methodologies have been provided for testing web-services [13, 18, 10], and to distribute the burden of testing by augmenting subsystems with the ability to test their environment and neighboring subsystems during runtime [1].

We propose a framework grounded in assume-guarantee contract algebra to aid the test engineer in reasoning over a test campaign. We make use of a test objective in the form of a specification, which together with the system specification is used to characterize an assume guarantee contract for the test environment. This allows us to define tests as pairs of contracts, and reason over these tests using operators from contract theory. This approach reasons over the specifications for the system and the tester to construct a test specification which is then used in synthesizing a test environment. Overall, our approach is complementary to falsification — test environments synthesized for the tester specifications discussed in this paper could be used to seed falsification algorithms to find a worst-case test execution. We seek to address the following questions by using operators from contract algebra to reason over test objectives and system specifications.

(Q1) *Constructing Tests:* How do we generate a specification for the test environment so that a desired behavior, characterized by the test objective, is demonstrated? See Section 3.
(Q2) *Comparing Tests:* When can we say that one test is a refinement of the other, and define an ordering of tests? See Section 5.
(Q3) *Combining Tests:* Is it possible to check for multiple unit test objectives in a single test execution? See Section 4.
(Q4) *Splitting Tests:* Is it possible to derive unit test objectives from a more complex test objective? See Section 6.

The focus of this paper is on reasoning about tests at the specification level, not on synthesizing tests from these specifications. We illustrate different possible test executions for a combined and a split test on a discrete autonomous car example and a formation flying example.

2 Background

To reason about the specifications, we will make use of the contract-based-design framework first introduced as a design methodology for modular software systems [14, 23, 24] and later extended to complex cyber-physical systems [5, 29, 26]. We will adopt the mathematical framework presented by Benveniste et al. [6] and Passerone et al. [27].

Definition 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 $\neg A \subseteq G$.

Multiple operations are known for assume guarantee contracts — see [21]. Assume the following contracts are in canonical form. The meet or conjunction of two contracts exists [5] and is given by $\mathcal{C}_1 \wedge \mathcal{C}_2 = (A_1 \cup A_2, G_1 \cap G_2)$. Composition [6] 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 $\mathcal{C}_1 \parallel \mathcal{C}_2 \leq \mathcal{C}$. It is given by [20]: $\mathcal{C}/\mathcal{C}_1 = (A \cup G_1, (G \cap A_1) \cup \neg(A \cup G_1))$. Strong merging [27] yields a specification obeyed by a system that obeys 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) [25, 27] is a unary operation which inverts assumptions and guarantees: $\mathcal{C}^{-1} = (G, A)$.

(a) Composition and quotient.

(b) Order of operations.

Fig. 1: Contract operators and the partial order of their resulting objects.

To state the requirements on the system and the test, we will make use of linear temporal logic (LTL), although any specification formalism can be used. LTL is a temporal logic describing linear-time properties, allowing reasoning over the timing of events, where each point in time has a single successor. The use of LTL for formally verifying properties of computer programs was first introduced by Pnueli in 1977 [28].

Definition 2 (Linear Temporal Logic (LTL) [3])**.** The syntax of *linear temporal logic (LTL)* is given as:

$$\varphi ::= \top \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \mathcal{U} \varphi_2,$$

with $a \in AP$, where AP are the set of atomic propositions, the Boolean connectors conjunction $\wedge$ and negation $\neg$, and the temporal operators 'next' $\bigcirc$ and 'until' $\mathcal{U}$. From conjunction and negation, we can derive the entirety of propositional logic including disjunction $\vee$, implication $\rightarrow$, and equivalence $\leftrightarrow$. The temporal operators 'always' $\Box$ and 'eventually' $\Diamond$ can be derived from $\mathcal{U}$ as

$$\Diamond\varphi = \top \mathcal{U} \varphi, \quad \Box\varphi = \neg\Diamond\neg\varphi.$$

From these temporal operators we can derive 'always eventually' $\Box\Diamond$ and 'eventually always' $\Diamond\Box$, which specify that a proposition will be true infinitely often (progress) or eventually forever (stability) respectively. Let φ be an LTL formula over AP. The semantics of LTL formula φ are defined over an infinite word $\sigma = s_0 s_1 \cdots$ as follows

$\sigma \models \top$,
For $a \in AP$, $\sigma \models a$ iff $\sigma_0 \models a$,
$\sigma \models \varphi_1 \wedge \varphi_2$ iff $\sigma \models \varphi_1$ and $\sigma \models \varphi_2$,
$\sigma \models \neg\varphi$,
$\sigma \models \bigcirc\varphi$ iff $\sigma[1, \cdots] = s_1 s_2 \cdots \models \varphi$,
$\sigma \models \varphi_1 \mathcal{U} \varphi_1$ iff $\exists j \geq 0, \sigma[j, \cdots] \models \varphi_2$ and $\sigma[i, \cdots] \models \varphi_1$, for all $0 \leq i < j$,

where $\sigma[j, \cdots]$ denotes the word fragment $s_j s_{j+1} \cdots$.

3 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 3. 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 4. 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 (3) and (4), 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 paper.

Definition 5. 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.

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.

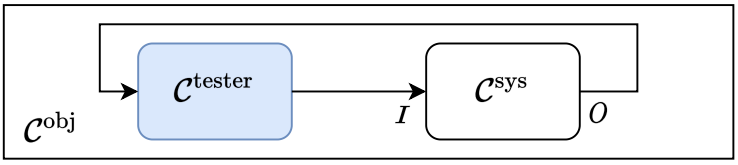


Fig. 2: Block diagram showing contracts specifying the system specification $\mathcal{C}^{\text{sys}}$, the test objective $\mathcal{C}^{\text{obj}}$, and the test environment $\mathcal{C}^{\text{tester}}$.

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 2, 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 $\mathcal{C}^{\text{sys}}$ [21]:

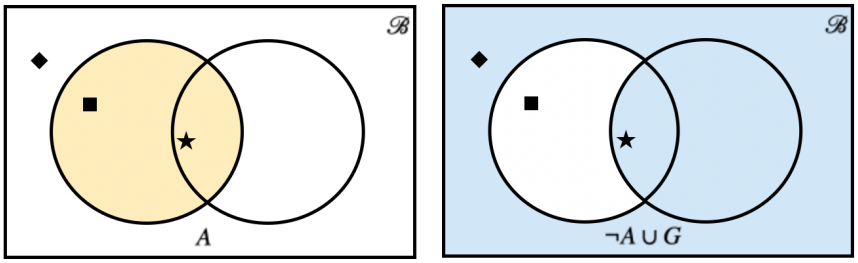
$$\mathcal{C}^{\text{tester}} = (\mathcal{C}^{\text{sys}})^{-1} \bullet \mathcal{C}^{\text{obj}} = \mathcal{C}^{\text{obj}} / \mathcal{C}^{\text{sys}}.$$

The tester contract can therefore directly be computed as

$$\mathcal{C}^{\text{tester}} = (G^{\text{sys}}, G^{\text{obj}} \cap A^{\text{sys}} \cup \neg G^{\text{sys}}). \tag{1}$$

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,

$$\bigcap_i (\neg A_i^{\text{sys}} \cup G_i^{\text{sys}}) \rightarrow A^{\text{sys}} \cap G^{\text{obj}}. \tag{2}$$



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

Fig. 3: 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.

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 3.

4 Combining Tests

We now provide a framework to combine unit test campaigns into a single system-level test structure. Suppose we have test structures $(\mathcal{C}_i^{\text{obj}}, \mathcal{C}_i^{\text{sys}})$ for $i \in \{1, 2\}$ with test environment (tester) contracts $\mathcal{C}_i^{\text{tester}}$. We interpret the specifications $\mathcal{C}_i^{\text{tester}}$ as viewpoints of the tester that apply to different specifications of the system. When we merge the tester specifications, we obtain a single test structure given as follows:

Proposition 1. $\mathcal{C}_1^{tester} \bullet \mathcal{C}_2^{tester} = (\mathcal{C}_1^{obj} \parallel \mathcal{C}_2^{obj}) / (\mathcal{C}_1^{sys} \parallel \mathcal{C}_2^{sys})$.

Proof. Merging tester contracts yields

$$
\begin{aligned}
\mathcal{C}_1^{\text{tester}} \bullet \mathcal{C}_2^{\text{tester}} &= (\mathcal{C}_1^{\text{obj}} / \mathcal{C}_1^{\text{sys}}) \bullet (\mathcal{C}_2^{\text{obj}} / \mathcal{C}_2^{\text{sys}}) \\
&= (\mathcal{C}_1^{\text{obj}} \bullet (\mathcal{C}_1^{\text{sys}})^{-1}) \bullet (\mathcal{C}_2^{\text{obj}} \bullet (\mathcal{C}_2^{\text{sys}})^{-1}) && \text{([22], Section 3.1)} \\
&= (\mathcal{C}_1^{\text{obj}} \bullet \mathcal{C}_2^{\text{obj}}) \bullet \left((\mathcal{C}_1^{\text{sys}})^{-1}) \bullet ((\mathcal{C}_2^{\text{sys}})^{-1}) \right) \\
&= (\mathcal{C}_1^{\text{obj}} \bullet \mathcal{C}_2^{\text{obj}}) \bullet (\mathcal{C}_1^{\text{sys}} \parallel \mathcal{C}_2^{\text{sys}})^{-1} && \text{([21], Table 6.1)} \\
&= (\mathcal{C}_1^{\text{obj}} \bullet \mathcal{C}_2^{\text{obj}}) / (\mathcal{C}_1^{\text{sys}} \parallel \mathcal{C}_2^{\text{sys}}) \\
&= (\mathcal{C}_1^{\text{obj}} \parallel \mathcal{C}_2^{\text{obj}}) / (\mathcal{C}_1^{\text{sys}} \parallel \mathcal{C}_2^{\text{sys}}), && (A_1^{\text{obj}} = A_2^{\text{obj}} = \top))
\end{aligned}
$$

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 resulting contract is the tester contract for the test structure given by the parallel compositions of the objective contracts and system contracts, separately. As we are defining the system specification as requirements on the subsystem to be tested, the composition of the system specifications represents a system consisting of the individual subsystems. We use Proposition 1 to define an operation on test structures directly:

Definition 6. 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

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

For the composition of the test structures to correspond to a valid test, we require the composed test objective and the resulting tester contract to be satisfiable.

Example 1. Consider a test setup with a single lane road and a pedestrian on a crosswalk. The agent under test is an autonomous car, which has to detect the pedestrian and come to a stop in front of the crosswalk under different visibility conditions. These requirements are encoded in the system specification and the test objective. The setup for this test is shown in Figure 4. Three unit test objective contracts are specified by the test engineer. The first test objective is as follows:

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

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 least one cell in front of the crosswalk cell C_{cw}. The second test objective is given as

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

where $\varphi_{\text{high}}^{\text{vis}} := \varphi^{\text{vis}} \models \text{high}$ denotes high visibility conditions; and lastly the third test objective is given as:

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

where the car has to drive at a specified speed of $V_{\max}$ in an arbitrary cell C_k and stop within the allowed braking distance d_{braking}. This test represents the mechanical requirement of stopping without specifying any interaction with a pedestrian. Note that neither of the test objective contracts hold information about the system's capabilities to detect a pedestrian, only that the system needs to stop in front of a pedestrian.

The system capabilities are encoded in the system specifications, which are provided by the system and test engineers. For each test objective, we are given the corresponding system specification, which describes the required capabilities of the system for that test objective (e.g. perception, mechanical requirements, etc.). Each system specification relies on the system being in a safe environment, where the transitions of the environment agents are ensured to be safe. This is denoted as $A^{\text{sys}} = \square\varphi^{\text{ped}}_{\text{dyn}} \wedge \square\varphi^{\text{vis}}_{\text{dyn}}$, where $\varphi^{\text{ped}}_{\text{dyn}}$, and $\varphi^{\text{vis}}_{\text{dyn}}$ denote the dynamics of the pedestrian, and the visibility conditions, respectively. We use the same notation for the set and formula A^{sys}, which can be inferred from context. The system contract $\mathcal{C}^{\text{sys}}_1$ corresponding to the first test objective is given as

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

where $\varphi^{\text{car}}_{\text{dyn}}$, describes the dynamics of the car. The maximum speed that the car is allowed to drive at in low visibility conditions is V_{low}, and $\texttt{detectable}^{\text{ped}}_{\text{low}}$ is defined as

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

which describes the pedestrian being in the 'buffer' zone in front of the car, where $dist^{\text{low}}_{\text{min}}$ denotes the minimum distance such that the car can come to a full stop, and $dist^{\text{low}}_{\text{max}}$ denotes the maximum distance at which the car can detect a pedestrian in low visibility conditions. The system specification for the second test objective, the system contract $\mathcal{C}^{\text{sys}}_2$, is given as

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

describing driving in high visibility conditions with a maximum speed of V_{max} and $\texttt{detectable}^{\text{ped}}_{\text{high}}$ denoting the pedestrian being detectable in the 'buffer' zone for high visibility conditions. The third system specification $\mathcal{C}^{\text{sys}}_3$ is given as

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

with the braking distance as a function of speed being part of the car's dynamics denoted by $\varphi^{\text{car}}_{\text{dyn}}$. For each pair of system specifications and test objectives, we can synthesize the test environment according to equation (2). Now we will find combinations of these tester structures $\mathfrak{t}_i = (\mathcal{C}^{\text{obj}}_i, \mathcal{C}^{\text{sys}}_i)$, that we can use instead of executing all tests individually. We will start by computing the combined test structure $\mathfrak{t} = \mathfrak{t}_2 \parallel \mathfrak{t}_3$. The combined test objective contract $\mathcal{C}^{\text{obj}}$ is computed as

$$\mathcal{C}^{\text{obj}} = \mathcal{C}^{\text{obj}}_2 \parallel \mathcal{C}^{\text{obj}}_3 = \big(\top, \quad \varphi^{\text{car}}_{\text{init}} \wedge \square\varphi^{\text{vis}}_{\text{low}} \wedge \Diamond\varphi^{\text{ped}}_{\text{cw}} \wedge \varphi^{\text{ped}}_{\text{cw}} \rightarrow \Diamond\varphi^{\text{stop}}_{\text{cw}} \wedge \\ \exists k : (v_{\text{car}} = V_{\text{max}} \wedge x_{\text{car}} = C_k) \rightarrow \Diamond\varphi^{\text{stop}}_{k+d_{\text{braking}}}\big). \tag{3}$$

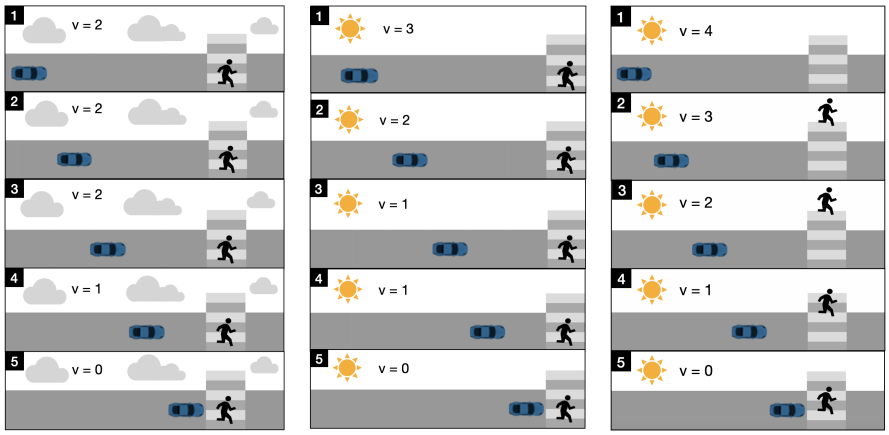
The combined system contract is computed as

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

We will relax this system contract by removing $\neg(G_2^{\text{sys}} \cap G_3^{\text{sys}})$ from the assumptions to ensure that the assumptions are in the same form as we require for the system contract in Definition 3. Consequently, the tester contract resulting from this system contract is more refined. So the system contract becomes

$$\begin{aligned}\mathcal{C}^{\text{sys}} = \big(&A^{\text{sys}},\quad \square\varphi_{\text{dyn}}^{\text{car}} \wedge \square\,(\varphi_{\text{high}}^{\text{vis}} \rightarrow v \leq V_{\max}) \wedge \\ &\square(\texttt{detectable}_{\text{high}}^{\text{ped}} \rightarrow \Diamond\varphi_{\text{ped}}^{\text{stop}}) \vee \neg A^{\text{sys}}\big).\end{aligned} \tag{4}$$

From equations (3) and (4), we construct the test structure $\mathfrak{t} = (\mathcal{C}^{\text{obj}}, \mathcal{C}^{\text{sys}})$, where every implementation that satisfies to equation (2) describes a valid test environment for this combined test. This merged tester specification describes a test environment where we will see the car decelerate from $V_{\max}$ and stop in front of the crosswalk in high visibility conditions.



(a) Low visibility with a stationary pedestrian.

(b) High visibility with a stationary pedestrian.

(c) High visibility with a reactive pedestrian.

Fig. 4: Test execution snapshots of the car stopping for a pedestrian. Figure 4a shows a test execution satisfying $\mathcal{C}_1^{\text{tester}}$, Figure 4b satisfies $\mathcal{C}_2^{\text{tester}}$ and Figure 4c satisfies $\mathcal{C}_2^{\text{tester}}$ and $\mathcal{C}_3^{\text{tester}}$.

To ensure that test structures can be combined, we need to check whether the resulting test objective, and the corresponding tester contract are satisfiable. We will now explain which combinations of the given test structures cannot be implemented for either of these reasons. Computing the composition $\mathfrak{t}_1 \parallel \mathfrak{t}_2$ is not possible, as the composition of the test objectives $\mathcal{C}_1^{\text{obj}} \parallel \mathcal{C}_2^{\text{obj}}$ results in a contract with empty guarantees. This is the case, because $\square\varphi_{\text{low}}^{\text{vis}}$ and $\square\varphi_{\text{high}}^{\text{vis}}$ are disjoint, as the visibility conditions cannot be *high* and *low* at the same

time. Thus these two test structures are not composable with each other. The composition $\mathtt{t}_1 \parallel \mathtt{t}_3$, does not result in a feasible test — the test objective requires a maximum speed of $V_{\max}$, but the system is constrained to a maximum speed of $V_{\text{low}} < V_{\max}$ in low visibility conditions, resulting in $G^{\text{sys}} \cap G^{\text{obj}} = \emptyset$.

Figure 4 shows snapshots of manually constructed test executions satisfying the tester contracts corresponding to $\mathtt{t}_1, \mathtt{t}_2$, and $\mathtt{t}_2 \parallel \mathtt{t}_3$. The simulation is in a grid world setting, where the car will move one cell forward if it has a positive speed v, and can accelerate or decelerate by one unit during every time step, meaning if the car is driving at a higher speed, it will take more cells to come to a stop. 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. So in Figure 4a it 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 $v_{\max} = 4$, and it can detect the pedestrian up to 5 cells ahead. In Figure 4b we can see that the pedestrian is detected and the car slows down gradually until is reaches the cell in front of the crosswalk. Figure 4c shows a test for the tester contract corresponding to $\mathtt{t}_2 \parallel \mathtt{t}_3$, 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 exactly $d_{\text{braking}} = 4$ cells away from the pedestrian. 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. ∎

Remark: Sometimes in addition to the combined test contract, the test executions must satisfy further constraints, informed by domain knowledge, to provide useful information to the test engineer. In the case of combining tests, a metric can be useful in determining whether we get the desired information from the execution of the combined test. In [17], to ensure that a merged test execution respects causality in satisfying all unit guarantees, temporal constraints are added to refine the merged test objective. Instead of refining the test structure, such additional constraints can also be handled during test environment synthesis. This can be helpful in determining if and how tests can be combined for a given available environment and the desired test information.

5 Comparing Test Campaigns

Justifying the choice of a test campaign from a list of possibilities requires a method of comparing test campaigns. A more refined test campaign is preferable to execute, because the system will be tested for a more refined set of test objectives and possibly for a more stringent set of system specifications. Let $\mathtt{t}_i = (\mathcal{C}_i^{\text{obj}}, \mathcal{C}_i^{\text{sys}})$ be test structures for $1 \leq i \leq n$. When generating tests for $\mathtt{t}_i$, 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 3, the tester contract can be computed using the quotient operator. We characterize a test campaign, $\mathtt{TC} = \{\mathtt{t}_i\}_{i=1}^n$, as a finite list of test structures specified by the

test engineer. Definition 7 allows us to generate a single test structure from a test campaign.

Definition 7. Given a test campaign $\mathtt{TC} = \{\mathsf{t}_i\}_{i=1}^n$, the *test structure generated by this campaign*, denoted $\tau(\mathtt{TC})$, is

$$\tau(\mathtt{TC}) = \mathsf{t}_1 \parallel \ldots \parallel \mathsf{t}_n.$$

To define a notion of order for test campaigns, we need a notion of order for test structures. Comparing two test structures becomes important for defining the quotient of test structures (see Section 6) for splitting tests.

Definition 8. We say that 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 $\mathcal{C}_1^{\text{obj}} \leq \mathcal{C}_2^{\text{obj}}$.

We use the order between test structures (see Definition 8) to know when a test campaign is more refined than another (see Definition 9). A test campaign can be replaced by a more refined test campaign because the refined test campaign includes more stringent specifications in more stringent settings.

Definition 9. Given two test campaigns $\mathtt{TC}$ and $\mathtt{TC}'$, we say that $\mathtt{TC} \leq \mathtt{TC}'$ if $\tau(\mathtt{TC}) \leq \tau(\mathtt{TC}')$.

6 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 [22]. Suppose there exists a test structure t that we want to split, and suppose one of the pieces of this decomposition, t_1, is given to us. Our objective is to find t_2 such that $\mathsf{t}_1 \parallel \mathsf{t}_2 \leq \mathsf{t}$. The following result tells how to compute the optimum t_2. This optimum receives the name *quotient of test structures*.

Proposition 2. *Let* $\mathsf{t} = (\mathcal{C}^{obj}, \mathcal{C}^{sys})$ *and* $\mathsf{t}_1 = (\mathcal{C}_1^{obj}, \mathcal{C}_1^{sys})$ *be two test structures and let* $\mathsf{t}_q = (\mathcal{C}^{obj}/\mathcal{C}_1^{obj}, \mathcal{C}^{sys}/\mathcal{C}_1^{sys})$. *For any test structure* $\mathsf{t}_2 = (\mathcal{C}_2^{obj}, \mathcal{C}_2^{sys})$, *we have*

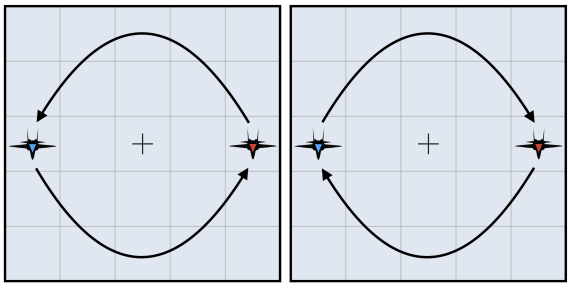
$$\mathsf{t}_2 \parallel \mathsf{t}_1 \leq \mathsf{t} \quad \textit{if and only if} \quad \mathsf{t}_2 \leq \mathsf{t}_q.$$

We say that t_q *is the quotient of* t *by* t_1, *and we denote it as* t/t_1.

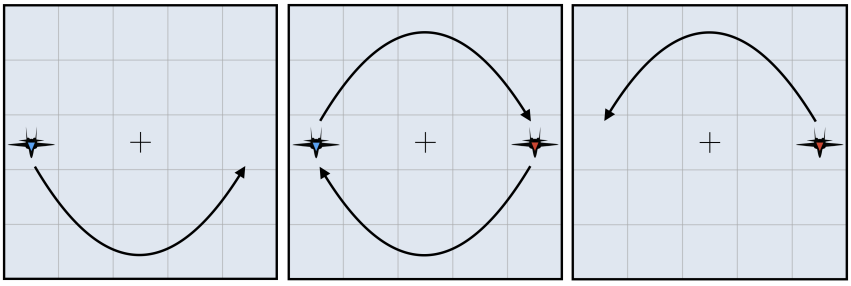
Proof. $\mathsf{t}_2 \leq \mathsf{t}_q \quad \Leftrightarrow \quad \mathcal{C}_2^{\text{sys}} \leq \mathcal{C}^{\text{sys}}/\mathcal{C}_1^{\text{sys}}$ and $\mathcal{C}_2^{\text{obj}} \leq \mathcal{C}^{\text{obj}}/\mathcal{C}_1^{\text{obj}} \Leftrightarrow \quad (\mathcal{C}_2^{\text{obj}} \parallel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

Remark: The method of constructing the quotient test structure in Proposition 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.



(a) Executions satisfying the original test structure.



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

Fig. 5: Front view of test executions satisfying the original test structure and the split test structure.

Example 2. Consider two aircraft, a_1 and a_2, flying parallel to each other undergoing a formation flying test shown in Figure 5a 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.

$$\begin{aligned} G_i^{\text{swap}} = &\square\big(\texttt{directive}^{\text{cw}}_{\text{swap}}(a_i) \rightarrow \texttt{execute}^{\text{cw}}_{\text{swap}}(a_i)\big) \wedge \\ &\square\big(\texttt{directive}^{\text{ccw}}_{\text{swap}}(a_i) \rightarrow \texttt{execute}^{\text{ccw}}_{\text{swap}}(a_i)\big). \end{aligned} \tag{5}$$

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 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 1).

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

Table 1: Subformulas for constructing G^{sys} and G^{obj}.

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: $\varphi^{\text{cw}} = \lozenge\texttt{directive}^{\text{cw}}_{\text{swap}}(a_1) \wedge \lozenge\texttt{directive}^{\text{cw}}_{\text{swap}}(a_2)$. 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 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:

$$\mathcal{C}^{\text{sys}} = (A^{\text{sys}}, G^{\text{sys}}) = (G^{\texttt{dir}}_{\text{limit}} \wedge G^{\texttt{dir}}_{\text{safe}},\ G^{\text{safe}} \wedge \bigwedge_i G_i^{\text{swap}} \wedge G_i^{\text{dyn}}). \tag{6}$$

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:

$$\mathcal{C}_i^{\text{sys}} = (A_i^{\text{sys}}, G_i^{\text{sys}}) = (G^{\texttt{dir}}_{\text{limit}} \wedge G^{\texttt{dir}}_{\text{safe}},\ G_i^{\text{swap}} \wedge G_i^{\text{dyn}}). \tag{7}$$

$$\begin{aligned}\mathcal{C}^{\text{obj}} &= (\top, G^{\text{obj}}), \quad \text{where}\\ G^{\text{obj}} &= (\varphi^{\text{cw}} \wedge \neg\varphi^{\text{ccw}}) \vee (\varphi^{\text{ccw}} \wedge \neg\varphi^{\text{cw}}).\end{aligned} \tag{8}$$

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,

$$\mathcal{C}_1^{\text{obj}} = (\top, G_1^{\text{obj}}) = (\top, \Diamond \texttt{directive}_{\text{swap}}^{\text{ccw}}(a_1)) \tag{9}$$

$$\mathcal{C}_1^{\text{sys}} = (A_1^{\text{sys}}, G_1^{\text{sys}}) = (G_{\text{limit}}^{\texttt{dir}} \wedge G_{\text{safe}}^{\texttt{dir}}, G_1^{\text{swap}} \wedge G_1^{\text{dyn}}). \tag{10}$$

Following Proposition 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 [20]. 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 $\varphi_1 = \left(\Diamond \texttt{directive}_{\text{swap}}^{\text{ccw}}(a_2) \wedge \neg\varphi^{\text{cw}}\right)$ 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:

$$\mathcal{C}_{a_2}^{\text{obj}} = (\top, \varphi_1 \vee \varphi_2) \leq (\top, \neg G_1^{\text{obj}} \vee \varphi_1 \vee \varphi_2). \tag{11}$$

In equation (11), 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:

$$\begin{aligned}\mathcal{C}^{\text{sys}}/\mathcal{C}_1^{\text{sys}} =& \left(A^{\text{sys}} \cap G_1^{\text{sys}}, G^{\text{sys}} \cap A_1^{\text{sys}} \cup \neg(A^{\text{sys}} \cap G_1^{\text{sys}})\right)\\ =& \big(G_{\text{limit}}^{\text{dir}} \wedge G_{\text{safe}}^{\text{dir}} \wedge G_1^{\text{swap}} \wedge G_1^{\text{dyn}}, (G^{\text{safe}} \wedge G_2^{\text{swap}} \wedge G_2^{\text{dyn}})\\ & \vee \neg(G_1^{\text{swap}} \wedge G_1^{dyn} \wedge G_{\text{limit}}^{\text{dir}} \wedge G_{\text{safe}}^{\text{dir}})\big)\\ =& (G_{\text{limit}}^{\text{dir}} \wedge G_{\text{safe}}^{\text{dir}} \wedge G_1^{\text{swap}} \wedge G_1^{\text{dyn}}, (G^{\text{safe}} \wedge G_2^{\text{swap}} \wedge G_2^{\text{dyn}})).\end{aligned} \tag{12}$$

Remark: Observe that equation (12) 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.

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,

$$\begin{aligned}\mathcal{C}_{\varphi_1}^{\text{tester}} = \Big(&G^{\text{safe}} \wedge G_2^{\text{swap}} \wedge G_2^{\text{dyn}},\ G_{\text{limit}}^{\text{dir}} \wedge G_{\text{safe}}^{\text{dir}} \wedge G_1^{\text{swap}} \wedge G_1^{\text{dyn}} \\ &\wedge \Diamond \texttt{directive}_{\text{swap}}^{\text{ccw}}(a_2) \wedge \neg\varphi^{\text{cw}}\Big).\end{aligned} \tag{13}$$

$$\begin{aligned}\mathcal{C}_{\varphi_2}^{\text{tester}} = \Big(&G^{\text{safe}} \wedge G_2^{\text{swap}} \wedge G_2^{\text{dyn}},\ G_{\text{limit}}^{\text{dir}} \wedge G_{\text{safe}}^{\text{dir}} \wedge G_1^{\text{swap}} \wedge G_1^{\text{dyn}} \\ &\wedge \Diamond \texttt{directive}_{\text{swap}}^{\text{cw}}(a_1) \wedge \Diamond \texttt{directive}_{\text{swap}}^{\text{cw}}(a_2) \wedge \neg\varphi^{\text{ccw}}\Big).\end{aligned} \tag{14}$$

From equation (13), we see that the tester does not require aircraft a_1 for any dynamic maneuvers, so it need not be deployed. In equation (14), 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. ∎

7 Conclusion and Future Work

We have developed formal notions for constructing, comparing, combining, and splitting test structures. We reason at the specification level of the test structures to find more refined test campaigns, and derive the tester specification from which a test environment can be synthesized. We give the conditions for when test structures are composable, allowing for simultaneous execution of test objectives when possible.

We briefly discussed the use of splitting tests for the application of failure diagnosis to find the root cause of a system-level test failure. Using the splitting operation on the test structure, we can isolate the components and verify their operation, assuming we can make use of a test harness, allowing us to monitor certain subsystem inputs and outputs. For future work, we can annotate which sub-component was used in context to satisfy a formula and thus use this additional information to track potential sources of a system-level failure. Additionally, we aim to construct an algorithm that will find a refined test campaign from a given test campaign that is optimal for a certain user-defined metric, e.g., test time or cost or coverage, while also accounting for test environment constraints.

Acknowledgements. The authors acknowledge funding from AFOSR Test and Evaluation program, grant FA9550-19-1-0302, and NSF and ASEE through an eFellows postdoctoral fellowship. The contents are solely the responsibility of the authors and do not necessarily represent the views of the sponsors.

References

1. Atkinson, C., Groß, H.G.: Built-in contract testing in model-driven, component-based development. In: Proc. of ICSR-7 Workshop on Component-Based Development Processes (2002)
2. Badithela, A., Graebener, J.B., Ubellacker, W., Mazumdar, E.V., Ames, A.D., Murray, R.M.: Synthesizing reactive test environments for autonomous systems: Testing reach-avoid specifications with multi-commodity flows. arXiv preprint arXiv:2210.10304 (2022)
3. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
4. Beer, A., Ramler, R.: The role of experience in software testing practice. In: 2008 34th Euromicro Conference Software Engineering and Advanced Applications. pp. 258–265. IEEE (2008)
5. Benveniste, A., Caillaud, B., Ferrari, A., Mangeruca, L., Passerone, R., Sofronis, C.: Multiple viewpoint contract-based specification and design. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.P. (eds.) Formal Methods for Components and Objects: 6th International Symposium, FMCO 2007, Amsterdam, The Netherlands, October 24-26, 2007, Revised Lectures. pp. 200–225. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-92188-2_9, https://doi.org/10.1007/978-3-540-92188-2_9
6. Benveniste, A., Caillaud, B., Nickovic, D., Passerone, R., Raclet, J.B., Reinkemeier, P., Sangiovanni-Vincentelli, A., Damm, W., Henzinger, T.A., Larsen, K.G., et al.: Contracts for system design. Foundations and Trends® in Electronic Design Automation **12**(2-3), 124–400 (2018)
7. Blundell, C., Giannakopoulou, D., Pundefinedsundefinedreanu, C.S.: Assume-guarantee testing. p. 1–es. SAVCBS '05, Association for Computing Machinery, New York, NY, USA (2005). https://doi.org/10.1145/1123058.1123060, https://doi.org/10.1145/1123058.1123060
8. Brandán Briones, L.: Assume-guarantee reasoning with ioco testing relation. on Testing Software and Systems: Short Papers p. 103 (2010)
9. Brat, G., Jonsson, A.: Challenges in verification and validation of autonomous systems for space exploration. In: Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005. vol. 5, pp. 2909–2914. IEEE (2005)
10. Bruno, M., Canfora, G., Di Penta, M., Esposito, G., Mazza, V.: Using test cases as contract to ensure service compliance across releases. In: Benatallah, B., Casati, F., Traverso, P. (eds.) Service-Oriented Computing - ICSOC 2005. pp. 87–100. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
11. Cofer, D., Gacek, A., Miller, S., Whalen, M.W., LaValley, B., Sha, L.: Compositional verification of architectural models. In: NASA Formal Methods Symposium. pp. 126–140. Springer (2012)
12. Dahm, W.J.: Technology horizons vision for the air force during 2010-2030 (video). Tech. rep., Chief Scientist (Air Force) Washington, DC (2011)
13. Dai, G., Bai, X., Wang, Y., Dai, F.: Contract-based testing for web services. In: 31st Annual International Computer Software and Applications Conference (COMPSAC 2007). vol. 1, pp. 517–526 (2007). https://doi.org/10.1109/COMPSAC.2007.100
14. Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Communications of the ACM **18**(8), 453–457 (1975)
15. Fremont, D.J., Kim, E., Pant, Y.V., Seshia, S.A., Acharya, A., Bruso, X., Wells, P., Lemke, S., Lu, Q., Mehta, S.: Formal scenario-based testing of autonomous vehicles:

From simulation to the real world. In: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC). pp. 1–8. IEEE (2020)
16. Giannakopoulou, D., Păsăreanu, C., Blundell, C.: Assume-guarantee testing for software components. IET Software **2**(6), 547–562 (2008)
17. Graebener, J.B., Badithela, A., Murray, R.M.: Towards better test coverage: Merging unit tests for autonomous systems. In: NASA Formal Methods Symposium. pp. 133–155. Springer (2022)
18. Heckel, R., Lohmann, M.: Towards contract-based testing of web services. Electronic Notes in Theoretical Computer Science **116**, 145–156 (2005). https://doi.org/https://doi.org/10.1016/j.entcs.2004.02.073, https://www.sciencedirect.com/science/article/pii/S1571066104052831, proceedings of the International Workshop on Test and Analysis of Component Based Systems (TACoS 2004)
19. Helle, P., Schamai, W., Strobel, C.: Testing of autonomous systems–challenges and current state-of-the-art. In: INCOSE international symposium. vol. 26, pp. 571–584. Wiley Online Library (2016)
20. Incer, I., Sangiovanni-Vincentelli, A.L., Lin, C.W., Kang, E.: Quotient for assume-guarantee contracts. In: 16th ACM-IEEE International Conference on Formal Methods and Models for System Design. pp. 67–77. MEMOCODE'18 (October 2018). https://doi.org/10.1109/MEMCOD.2018.8556872
21. Incer, I.: The Algebra of Contracts. Ph.D. thesis, EECS Department, University of California, Berkeley (May 2022)
22. Incer, I., Mangeruca, L., Villa, T., Sangiovanni-Vincentelli, A.: The quotient in preorder theories. arXiv:2009.10886 (2020)
23. Lamport, L.: win and sin: Predicate transformers for concurrency. ACM Transactions on Programming Languages and Systems (TOPLAS) **12**(3), 396–428 (1990)
24. Meyer, B.: Applying 'design by contract'. Computer **25**(10), 40–51 (1992)
25. Negulescu, R.: Process spaces. In: Palamidessi, C. (ed.) CONCUR 2000 — Concurrency Theory. pp. 199–213. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
26. Nuzzo, P., Sangiovanni-Vincentelli, A.L., Bresolin, D., Geretti, L., Villa, T.: A platform-based design methodology with contracts and related tools for the design of cyber-physical systems. Proceedings of the IEEE **103**(11), 2104–2132 (2015)
27. Passerone, R., Incer, I., Sangiovanni-Vincentelli, A.L.: Coherent extension, composition, and merging operators in contract models for system design. ACM Transactions on Embedded Computing Systems (TECS) **18**(5s), 1–23 (2019)
28. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977). pp. 46–57. IEEE (1977)
29. Sangiovanni-Vincentelli, A.L., Damm, W., Passerone, R.: Taming Dr. Frankenstein: Contract-based design for cyber-physical systems. Eur. J. Control **18**(3), 217–238 (2012). https://doi.org/10.3166/ejc.18.217-238, https://doi.org/10.3166/ejc.18.217-238
30. Seshia, S.A., Sadigh, D., Sastry, S.S.: Towards verified artificial intelligence. arXiv preprint arXiv:1606.08514 (2016)
31. Tretmans, J.: Model based testing with labelled transition systems. In: Formal methods and testing, pp. 1–38. Springer (2008)
32. Weiss, L.G.: Autonomous robots in the fog of war. IEEE Spectrum **48**(8), 30–57 (2011)