Constraint-Behavior Contracts: A Formalism for Specifying Physical Systems

Sheng-Jung Yu*, Inigo Incer*, Alberto Sangiovanni-Vincentelli*
*University of California, Berkeley
{shengjungyu, inigo, alberto}@berkeley.edu

***Abstract*—Contract-based design (CBD) is a system development methodology that addresses the ever-increasing complexity and heterogeneity of cyber-physical system design problems. In CBD, systems and subsystems are represented by assume-guarantee contracts, formal specifications that make a distinction between supported environments and guarantees offered by a component. While the assume-guarantee formalism offers compact encoding of specifications, the port directions, i.e., the classifications of ports as input and output ports, implied by its semantics make it difficult to express properties of physical systems and subsystems where the relation between inputs and outputs is implicit. In this paper, we propose *constraint-behavior contracts* to specify physical components without the need to classify ports. The operations and relations between constraint-behavior contracts are defined to facilitate system reasoning without port directions. The capability of constraint-behavior contracts to integrate with assume-guarantee contracts gives the user the choice of a formalism to use at different abstraction layers. A case study based on an Unmanned Aerial Vehicle design problem shows that the proposed constraint-behavior contracts can facilitate system verification by expressing physical components, reducing the number of contracts, and providing an intuitive encoding of contracts.**

I. Introduction

Cyber-Physical Systems (CPS) involve computation and physical processes whose behavior is defined by their interaction [1]. As the need for large-scale CPS increases in applications such as autonomous vehicles, Industry 4.0, and smart grids, the heterogeneous nature and complex interaction between the parts in CPS cause the prolonged and error-prone design process and thus result in prohibitively high costs. Contract-based design (CBD) is a system development methodology that relies on contracts—formal specifications that define the expected environments and implementations—to enable correct-by-construction design, incorporate different design aspects, and reduce design complexity [2]–[5]. To tackle the design challenges caused by this complexity and heterogeneity, the formalisms of contracts and contract-based design have attracted research interests [6]–[9]. Among many formalisms, assume-guarantee contracts are among the most promising candidates due to their compactness and ease of use. An assume-guarantee contract is a pair of assumption and guarantee sets $\mathcal{C} = (A, G)$ [10]. Its semantics states that when the environment behavior satisfies the assumption set A, the specified component produces behaviors within the guarantee set G. The acceptable behaviors allowed by the contracts are, therefore, $G \cup \overline{A}$, and the behaviors under the environments E are $E \cap (G \cup \overline{A})$. Assume-guarantee contracts have several operations which are relevant for system design [10]–[13].

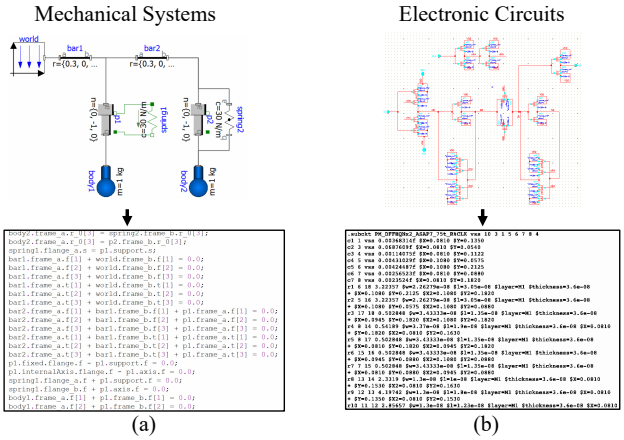

Fig. 1: Two examples of systems that use many implicit equations for modeling: (a) a Modelica example model of a spring mass system [14], and (b) a SPICE model of a parasitic extracted D Flip-Flop from the ASAP7 Design Kit [15].

In order to apply contract-based design in all steps of the CPS design process, it is necessary to formulate specifications as contracts for every component, including both the cyber and physical components. The cyber components, such as networks and control algorithms, monitor and control the physical components. Physical components, such as sensors and actuators, interact with the environment and create feedback loops for the computation components to perform specific actions. Therefore, the ability to express specifications for these diverse components and their interactions in contracts is crucial for fully leveraging the benefit of contract-based design.

The assume-guarantee contracts implicitly require port directions, and the assumption should be expressed only by the input ports. However, for many physical system applications, implicit equations are natural for system modeling [16]. Converting an implicit equation into an explicit form requires solving the equations either in a closed form or using a numerical algorithm where a closed form does not exist. As shown in Figure 1, electronic and mechanical systems are usually defined by a system of implicit equations with more than one thousand variables. Popular modeling tools are Simulink [17], which utilizes a signal-flow-based model requiring an explicit relation between input and output, and Modelica [14], which use equation-based models that do not need an explicit relation.

Figure 2 illustrates a challenge of writing assume-guarantee contracts for a resistor—a component allowing different port directions. The resistor, as shown in Figure 2a, can take voltage

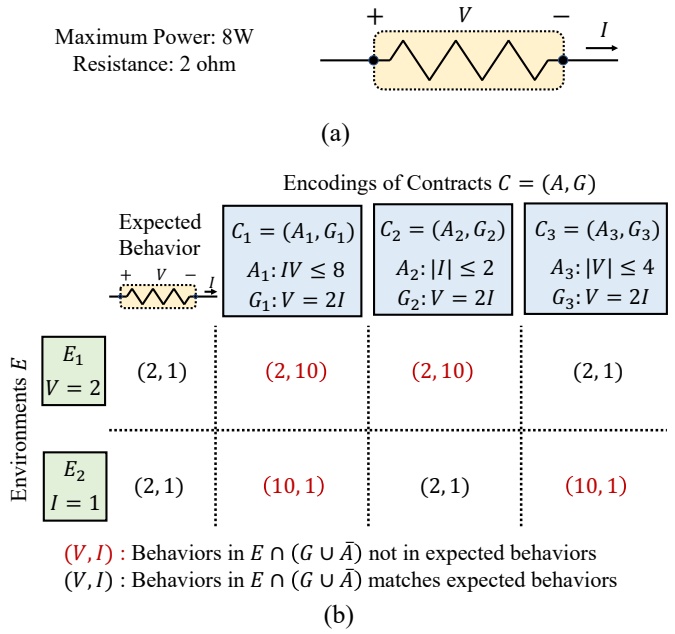

Fig. 2: Implicit port directions in assume-guarantee contracts: (a) a resistor with maximum power constraint as a motivating example component and (b) three assume-guarantee contract formulations for the resistor. The contract expresses the behaviors (V, I) correctly only when the actual input ports match the ports for defining the assumption.

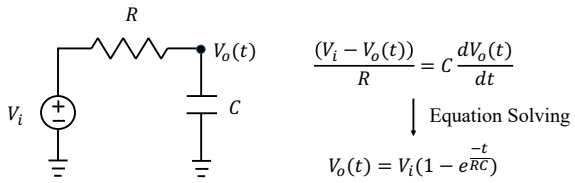

Fig. 3: An example showing that even when the port directions of individual components are known, the composed system is expressed in terms of implicit equations and requires solving equations to convert it to an explicit expression.

as input and current as output, and it can also take current as input and provide current as output. The port directions depend on the environment, meaning the component does not define its port directions. One may formulate the assume-guarantee contracts for the resistor as the three formulations shown in Figure 2b. However, all these formulations express the behaviors correctly only when the actual input ports match the ports for defining the assumption. When the actual input ports do not match the ports in the assumption, unexpected behaviors that cannot be produced by the specified components are allowed by the contracts.

Figure 3 shows an example in which the explicit relation of the output voltage to the input voltage requires solving an implicit equation. As a result, the ability to express the property of a physical system in implicit equations is important to avoid equation solving.

The requirements for port directions have several disadvantages. First, implicit equations need to be converted to explicit equations that reflect the input ports. For example, the assumptions of the formulations $\mathcal{C}_2$ and $\mathcal{C}_3$ in Figure 2 are not straightforward from the original bound on power. Second, the size of the contract library and the complexity of using contracts increases since a contract cannot be reused for environments with different inputs. The designers have to create multiple contracts with different combinations of the port directions and select the one that matches the inputs from the environments. Finally, contracts cannot be used when we are unable to define the port directions since we cannot select the contracts without the environment.

This paper addresses these issues by proposing a new contract formalism called constraint-behavior contracts for expressing specifications of physical components using implicit equations. To the best of our knowledge, this is the *first* approach that explicitly considers contract formulations for physical components governed by implicit equations. The formalism has the following properties:

- Constraint-behavior contracts are invariant to port directions, which allows the contract library to be compactly created without solving implicit equations and enumerating combinations of port directions.
- The physical meaning of system requirements is preserved by the implicit equations in constraint-behavior contracts. Preserving physical meaning in contracts allows designers to formulate contracts and discover potential design faults more easily.
- Constraint-behavior contracts can be integrated with assume-guarantee contracts in the contract-based design process. The approach is exemplified by a demonstration based on an Unmanned Aerial Vehicle (UAV) system design verification problem.

The remainder of this paper is organized as follows. In Section II, we introduce contracts and assume-guarantee contracts. Then, in Section III, we present the Constraint-behavior contracts and discuss their properties. Section IV compares constraint-behavior contracts with assume-guarantee contracts. Next, in Section V, we introduce the verification methodology for using constraint-behavior contracts. We demonstrate the application of constraint-behavior contracts in a UAV propulsion system design verification in Section VI. In Section VII, we discuss related work and background development of the contract formalisms. Section VIII concludes this paper.

II. Preliminaries

This section introduces assume-guarantee contracts and their operations. This is the foundation of constraint-behavior contracts, as explained later.

A. Ports, behaviors, and components

A system interacts with its environment through the values on its ports. Each port p is associated with a port type that represents the set of possible values for the port. The system ports are denoted as $\mathcal{P}$. For example, in the case of a resistor, the system ports include the voltage V across it and the current I passing through it, represented as $\mathcal{P} = \{V, I\}$. The port types

for voltage and current are the sets of real numbers, denoted $\mathbb{R}$.

A port behavior refers to a finite or infinite successive assignment of values within the port type. The universe of port behaviors, denoted as $\mathcal{B}_p$, encompasses all possible behaviors of the port. A system behavior is a finite or infinite successive assignment of values to the system ports. The universe of system behaviors is defined as the direct product of the port behavior universes, i.e., $\mathcal{B}_\mathcal{P} = \prod_{p\in\mathcal{P}} \mathcal{B}_p$.

A component is characterized by a subset of the universe of system behaviors, denoted as $M \subseteq \mathcal{B}_\mathcal{P}$, which contains the system behaviors that the component can produce. For instance, in the case of a resistor with a resistance of 2 Ω, a valid behavior could be $(V, I) = (2, 1)$, indicating a voltage of 2 V and a current of 1 A. Conversely, $(V, I) = (2, 2)$ is not a valid behavior for the resistor, as it violates Ohm's law.

B. Assume-guarantee contracts

A contract [18] is a formal specification for a system defined by a pair of component sets $(\mathcal{E}, \mathcal{I})$. The environment set, denoted $\mathcal{E}$, consists of components that could serve as the system's environment. The implementation set, denoted $\mathcal{I}$, comprises components that can realize the specification. A contract can be represented in different forms and semantics to express its environment and implementation set. One such representation is the assume-guarantee contract—see Chapter 5 of [18].

An assume-guarantee contract $\mathcal{C}^{ag}$ represents a contract by a pair of sets (A, G), where A is the assumption set, and G is the guarantee set.

The assumption set and guarantee set are defined by the behaviors over the system ports $\mathcal{P}$. The assumption set describes the behaviors of the targeted environments where the designer expects the system to operate. The guarantee set specifies the acceptable behaviors the system should produce when operating under the targeted environment. When the system operates in an environment behavior not included in the assumption set, its behavior is not specified, and thus all behaviors are acceptable. As a result, the acceptable behavior set of a contract $\mathcal{C}^{ag} = (A, G)$ is $G \cup \overline{A}$. The environment set represented by an assume-guarantee contract is $\mathcal{E} = 2^A$, the powerset of the assumption set, and the implementation set is $\mathcal{I} = 2^{G\cup\overline{A}}$, the powerset of the acceptable behavior set. The derivation of the environment set can be obtained by observing that the behavior set of an environment component must be a subset of the assumption set. For the implementation set, any component $M_\mathcal{C}$ that satisfies $M_\mathcal{C} \cap A \subseteq G$ is an implementation of the assume-guarantee contracts according to the semantics. As a result, the implementation is any subset of the acceptable behavior set: $M_\mathcal{C} \subseteq G \cup \overline{A}$.

An assume-guarantee contract is *saturated* if it satisfies $G \cup \overline{A} = G$, meaning that the guarantee set includes all the acceptable behaviors. Any assume-guarantee contract can be saturated by the saturation operator

$$\text{sat}_{\text{ag}}(A, G) = (A, G \cup \overline{A}).$$

The semantics of the specification remains unchanged, as saturation does not change the environments or implementations of the contract.

C. Contract operations

Contracts facilitate system reasoning at the specification level through their operations and relations. This abstraction allows virtual integration without any implementations and is the foundation of contract-based design [3]. As design problems can be detected at the specification stage, the preliminary stage in the design process, the time and cost for design can be reduced by avoiding manufacturing and deploying components failing to satisfy the specification.

Refinement is a relation between contracts, and composition is an important operation for hierarchical reasoning. We give their formal definitions [18].

a) Refinement: Given two saturated contracts $\mathcal{C}_1^{ag} = (A_1, G_1)$ and $\mathcal{C}_2^{ag} = (A_2, G_2)$, $\mathcal{C}_2^{ag}$ refines $\mathcal{C}_1^{ag}$, denoted by $\mathcal{C}_1 \succeq \mathcal{C}_2$, if $A_2 \supseteq A_1$ and $G_2 \subseteq G_1$. Equivalently, we can say $\mathcal{C}_2^{ag}$ is a refinement of $\mathcal{C}_1^{ag}$, and $\mathcal{C}_1^{ag}$ is a relaxation of $\mathcal{C}_2^{ag}$. The refinement relation defines a partial order over contracts. The relaxed contract can be replaced by its refinement without violating the specification. All implementations based on the refined contract can satisfy the original contract.

b) Composition: The composition of two saturated contracts, denoted $\mathcal{C}_1^{ag} \parallel \mathcal{C}_2^{ag}$, can be computed as $((A_1 \cap A_2) \cup \overline{(G_1 \cap G_2)}, G_1 \cap G_2)$. The composition of contracts is the overall system specification considering all the subsystem specifications. The specification thus moves to a hierarchy for reasoning in the composed systems instead of individual subsystems.

III. Constraint-behavior Contracts

This section first discusses why assume-guarantee contracts implicitly impose port directions. Then we introduce constraint-behavior contracts to address the difficulties that implicit port directions present to physical components. Operations and relations for constraint-behavior contracts are also introduced.

A. Port sensitivity and implicit port directions

In our discussion of Figure 2 in Section I, we observed that assume-guarantee contracts may implicitly require port directions, which makes expressing behaviors in implicit equations challenging. We analyze the cause of implicit port directions in assume-guarantee contracts and then present the requirements for a contract formalism that does not imply port directions.

First, we introduce the notion of port sensitivity.

Definition 1. *A set of behaviors A is said to be* insensitive *to a port if the behavior of the port does not affect A. If it does then A is* sensitive *to the port.*

If a port is not used in the expression of the behavior set, the behavior set is insensitive to the port. For example, considering a system containing two resistors with resistances R_1 and R_2, respectively, the behavior set defined by Ohm's law $V_1 = I_1 R_1$ is sensitive to the voltage V_1 and current I_1. On the other hand,

the behavior set defined by Ohm's law of the other resistor $V_2 = I_2R_2$ is insensitive to the voltage V_1 and current I_1, as any values of V_1 and I_1 do not affect whether a system behavior is contained by the behavior set.

When an environment controls all the ports that the assumption set is sensitive to, the satisfaction of the assumption is determined by the behavior set of the environment on these ports. Let E denote the behavior set of the environment. If the environment satisfies the assumption, the resulting behavior must fall within the intersection of the behavior of the environment and the guarantees, denoted as $E \cap G$, since an assume-guarantee contract requires the system to ensure the specified guarantee when the assumption is satisfied. On the other hand, if the environment violates the assumption, denoted as $E \not\subseteq A$, the system is not required to provide any guarantees.

However, when some ports that the assumption set is sensitive to are not controlled by the environment, the satisfaction of the assumption depends on the behavior of the uncontrolled ports, which can be any behavior in the universe of the port behavior. Therefore, the specified component is allowed to produce any behavior, since the assumptions can be violated by the behaviors of the uncontrolled ports. The contract thus fails to represent the components for our purpose because it always contains behaviors that the component should not produce under the environment. As a result, assume-guarantee contracts implicitly require the component to define all ports that the assumption set is sensitive to as input ports.

In the example shown in Figure 2, the environment E_1 does not control the current I; hence, the assumptions in formulations $\mathcal{C}_1$ and $\mathcal{C}_2$ are sensitive to I. Consequently, these contracts allow behaviors such as $(V, I) = (2, 10)$ because they do not specify the component when the current values violate the assumptions. However, the resistor should only produce the behavior $(V, I) = (2, 1)$ when the voltage is 2 V.

Thus, it is necessary to define a contract formalism that allows us to specify components regardless of which ports are controlled by its environment. The requirements of such formalism are as follows:

1) The responsibilities of the components, i.e. the required behaviors, should be considered before checking the satisfaction of the working condition. This aspect is the main difference between specifications using implicit equations and explicit equations. By considering the responsibility of the components first, we obtain the possible behavior for the ports not controlled by the environment, instead of allowing arbitrary behaviors that may violate the specified conditions.
2) The contracts should have a compact encoding and be easy to use, similar to assume-guarantee contracts. As the specification is the initial design stage, designers need to write, interpret, and examine the specifications. Ease of use and compactness will enable designers to express their intentions clearly and identify potential faults effectively.
3) The new contract formalism should seamlessly integrate into the design flow with contracts having port directions, such as assume-guarantee contracts. Depending on the application's requirements, the designer can define port directions for the subsystem created by physical components. For example, in a propulsion system, the control values can be treated as inputs, while the generated thrust can be treated as its output. Integration of these specifications allows moving between different abstraction layers. The overall system can be specified based on the port directions that reflect its usage, even if the components do not have port directions.

B. Constraint-behavior contracts

Based on the requirements above, we propose *constraint-behavior contracts* as a formalism for specifying physical components.

We use *constraints* and *intrinsic behaviors* to describe physical components. The behaviors are the responsibility of the component, typically expressed in physical quantities. Constraints define the conditions under which the behaviors apply. For example, the resistor of Figure 2 imposes the behaviors $V = IR$, but it operates under the constraints $VI \leq P$.

As long as the system behavior following the intrinsic behaviors satisfies the constraints, the component functions as specified, producing the behavior according to its intrinsic behaviors. If the system behavior following the intrinsic behaviors does not satisfy the constraints, the component fails, and the resulting behavior becomes unspecified, except for the value controlled directly by the environment.

We define constraint-behavior contracts and their semantics as follows:

Definition 2. *Let $\mathcal{P}$ be the system ports, C be a set of behaviors called constraints, and B be a set of behaviors called intrinsic behaviors. A constraint-behavior contract is a pair of constraints and intrinsic behaviors denoted as $\mathcal{C}^{cb} = (C, B)$, where $C \subseteq \mathcal{B}_\mathcal{P}$ and $B \subseteq \mathcal{B}_\mathcal{P}$.*

The behaviors of a constraint-behavior contract under an environment $E \subseteq \mathcal{B}_\mathcal{P}$ are

$$ite(E \cap B \subseteq C, E \cap B, E),$$

where ite() is the IF-THEN-ELSE operator.

The semantics of constraint-behavior contracts differs from that of assume-guarantee contracts. Observe that the responsibility of the constraint-behavior contracts is applied first to verify that the specified behaviors satisfy the constraints, which define the working condition of the components. This fulfills our first requirement for the contract formalism for physical components.

The following example shows a constraint-behavior contract for specifying the resistor in Figure 2 (a):

Example 1. *We can write the constraint-behavior contract for the resistor as $\mathcal{C}_r^{cb} = (C_r, B_r)$:*

$$C_r : IV \leq 8 \text{ and } B_r : V = 2I$$

We can verify that its semantics align with our intuition of a resistor. When the environment provides $V = 2$, we apply the intrinsic behaviors to the environment's behaviors and observe that the constraints are satisfied: $E \cap B_r = (V, I) = (2, 1)$,

which is a subset of $C_r = (IV \leq 8)$*. Thus, the behaviors of the constraint-behavior contract under this environment are* $E \cap B_r = (V, I) = (2, 1)$*. Similarly, when the environment provides* $I = 1$*, we apply the intrinsic behaviors to the environment's behaviors and find that the constraints are satisfied:* $E \cap B_r = (V, I) = (2, 1)$*, which is a subset of* $C_r = (IV \leq 8)$*. Therefore, the resulting behavior is* $E \cap B_r = \{(V, I) = (2, 1)\}$*.*

This example also shows that constraint-behavior contracts can be applied to environments with different controlled ports.

Constraint-behavior contracts can be expressed as assume-guarantee contracts:

Proposition 1. *A constraint-behavior contract* $\mathcal{C}^{cb} = (C, B)$ *possesses the same semantics as the assume-guarantee contract* $\mathcal{C}^{ag} = (C \cup \overline{B}, B)$*.*

The equivalence of semantics can be shown by using the contract definitions introduced in II-B. The environment set $\mathcal{E}$ of a constraint-behavior contract consists of any components that satisfy the constraints when applied to the intrinsic behaviors. Therefore, $\mathcal{E} = \{E \mid E \cap B \subseteq C\} = 2^{C \cup \overline{B}}$. The implementation set $\mathcal{I}$ of constraint-behavior contracts should satisfy the $\mathcal{I} = \{M \mid \forall E \in \mathcal{E},\ M \cap E \subseteq B\}$, which simplifies to $\mathcal{I} = 2^B$. Since an assume-guarantee contract $\mathcal{C}^{ag} = (A, G)$ has an environment set $\mathcal{E} = 2^A$ and implementations $\mathcal{I} = 2^{G \cup \overline{A}}$, we can match the expressions from both semantics to obtain $A = C \cup \overline{B}$ and $G = B$.

The following example shows the corresponding assume-guarantee for the resistor in Figure 2 (a):

Example 2. *The assume-guarantee contract for the constraint-behavior contract* $\mathcal{C}_r^{cb}$*, according to Proposition 1, is* $\mathcal{C}_r^{ag} = (IV \leq 8 \vee V \neq 2I, V = 2I)$*.*

We observe that all behaviors of $V = 2$ *satisfy the assumption, as* $(V, I) = (2, 1)$ *satisfies* $IV \leq 8$*, and all other behaviors with* $I \neq 1$ *satisfy* $V \neq 2I$*. Therefore, the guarantee is always enforced, and it eliminates all the behaviors satisfying* $V \neq 2I$*. The only remaining behavior is* $(V, I) = (2, 1)$*. A similar derivation can be obtained with the environment* $I = 1$*.*

The example also highlights the counter-intuitive nature of encoding assume-guarantee contracts for physical components. In the derived assume-guarantee contract, all behaviors that violate the guarantee are initially accepted based on the assumption but later eliminated by the guarantee. The mixture of constraints and intrinsic behaviors complicates understanding the specification's intention, while the numerous illegal and subsequently discarded behaviors further complicate behavior derivation. Therefore, using constraint-behavior contracts allows designers to write specifications in an intuitive way by preserving the physical meaning of the components without considering the intermediate illegal behaviors.

C. Operations and relations

As introduced in Section II, the contract operations and relation can facilitate system reasoning at the specification level. To this end, we discuss the contract operation for constraint-behavior contracts.

1) Composition: First, we define the composition of constraint-behavior contracts as follows:

Definition 3. *The composition of two constraint-behavior contracts* $\mathcal{C}_1^{cb} = (C_1, B_1)$*,* $\mathcal{C}_2^{cb} = (C_2, B_2)$*, denoted by* $\mathcal{C}_1^{cb} \parallel_{cb} \mathcal{C}_2^{cb}$*, is a constraint-behavior contract* $\mathcal{C}_{12}^{cb} = (C_{12}, B_{12})$*, where*

$$C_{12} = C_1 \cap C_2 \text{ and } B_{12} = B_1 \cap B_2.$$

The intuition of the composition operation is that the constraints of both contracts should be satisfied at the same time, and the intrinsic behaviors of both components are in force simultaneously.

We can show that the composition operation for constraint-behavior contracts aligns with the composition of assume-guarantee contracts. Considering the corresponding assume-guarantee contracts $\mathcal{C}_1^{ag} = (A_1, G_1) = (C_1 \cup \overline{B_1}, B_1)$ and $\mathcal{C}_2^{ag} = (A_2, G_2) = (C_2 \cup \overline{B_2}, B_2)$, we can use assume-guarantee contract composition to show that the composition result is equivalent to the one obtained following Definition 3:

$$\begin{aligned}
\mathcal{C}_{12}^{ag} &= \mathcal{C}_1^{ag} \parallel \mathcal{C}_2^{ag} \\
&= ((A_1 \cap A_2) \cup \overline{G_1} \cup \overline{G_2}, G_1 \cap G_2) \\
&= (((C_1 \cup \overline{B_1}) \cap (C_2 \cup \overline{B_2})) \cup \overline{B_1} \cup \overline{B_2}, B_1 \cap B_2) \\
&= ((C_1 \cap C_2) \cup \overline{(B_1 \cap B_2)}, B_1 \cap B_2) \\
&= (C_{12} \cup \overline{B_{12}}, B_{12}),
\end{aligned}$$

which is the corresponding assume-guarantee contract of the composed constraint-behavior contract $\mathcal{C}_{12}^{cb}$.

The following example illustrates the composition of two resistor specifications when the resistors are in parallel:

Example 3. *Consider the two contracts* $\mathcal{C}_1^{cb} = (C_{r1}, B_{r1})$ *and* $\mathcal{C}_2^{cb} = (C_{r2}, B_{r2})$ *for two resistors, where* $C_{r1} : I_1 V \leq 8$*,* $B_{r1} : V = 2I_1$*,* $C_{r2} : I_2 V \leq 16$*, and* $B_{r2} : V = 4I_2$*. The composition is* $\mathcal{C}_{12}^{cb} = (C_{r12}, B_{r12})$*, where* C_{r12} *is* $(I_1 V \leq 8) \wedge (I_2 V \leq 16)$ *and* B_{r12} *is* $(V = 2I_1) \wedge (V = 4I_2)$*.*

When the environment provides the voltage $V = 4$*, the resulting behavior is* $(V, I_1, I_2) = (4, 2, 1)$*.*

2) Refinement: We define the refinement relation of constraint-behavior contracts:

Definition 4. *Given two constraint-behavior contracts* $\mathcal{C}_1^{cb} = (C_1, B_1)$ *and* $\mathcal{C}_2^{cb} = (C_2, B_2)$*,* $\mathcal{C}_2^{cb}$ *is a refinement of* $\mathcal{C}_1^{cb}$*, denoted as* $\mathcal{C}_1^{cb} \succeq \mathcal{C}_2^{cb}$*, if they satisfy the following relation:*

$$C_1 \cup \overline{B_1} \subseteq C_2 \cup \overline{B_2} \text{ and } B_2 \subseteq B_1.$$

Similar to composition, we can show that the notion of refinement for constraint-behavior contracts aligns with assume-guarantee contract refinement. Consider their corresponding assume-guarantee contracts $\mathcal{C}_1^{ag} = (A_1, G_1) = (C_1 \cup \overline{B_1}, B_1)$ and $\mathcal{C}_2^{ag} = (A_2, G_2) = (C_2 \cup \overline{B_2}, B_2)$ and suppose they satisfy the refinement relation $\mathcal{C}_1^{ag} \succeq \mathcal{C}_2^{ag}$. Using assume-guarantee contract refinement as in Section II, the refinement relation requires that the following conditions hold:

$$C_1 \cup \overline{B_1} = A_1 \subseteq A_2 = C_2 \cup \overline{B_2}, B_2 = G_2 \subseteq G_1 = B_1$$

which is the same condition as the condition for constraint-behavior contract refinement.

Here we show an example of the contract refinement based on Example 3:

Example 4. *We want to check if the composition result in Example 3 is a refinement of the system contract* $\mathcal{C}^{cb}_{r3} = (C_{r3}, B_{r3}) = ((I_1 + I_2)V \leq 12, V = \frac{4}{3}(I_1 + I_2))$.

First, we check if the intrinsic behaviors satisfy the condition $B_{r12} \subseteq B_{r3}$. *By denoting any element in* R_{r12} *as* $(V, I_1, I_2) = (V, \frac{1}{2}V, \frac{1}{4}V)$, *the element must also be an element in* R_{r3} *since* $V = \frac{4}{3}(\frac{1}{2}V + \frac{1}{4}V)$. *Therefore, we get* $R_{r12} \subseteq R_{r3}$. *This derivation is equivalent to the derivation of the equivalent resistance for parallel resistors:* ${r_3}^{-1} = {r_1}^{-1} + {r_2}^{-1}$, *where* $r_3 = \frac{4}{3}$, $r_1 = 2$, *and* $r_2 = 4$ *are the resistances of the resistors.*

Then we check if the constraints satisfy the relation in Definition 4. We first get $C_{r12} \cup \overline{B_{r12}} = (I_1V \leq 8) \wedge (I_2V \leq 16) \vee (V \neq 2I_1) \vee (V \neq 4I_2)$. *An element in* $C_{r3} = (I_1 + I_2)V \leq 12$ *is an element of* C_{r12} *if* $V \neq 2I_1$ *or* $V \neq 4I_2$. *Therefore, we only need to check if all elements satisfying* $V = 2I_1$ *and* $V = 4I_2$ *in* C_{r3} *are elements of* $(I_1V \leq 8) \wedge (I_2V \leq 16)$. *Using the relation between voltages and currents, we know the ratio between* I_1V *and* I_2V *is always* 2. *Therefore, from* $(I_1 + I_2)V \leq 12$, *the maximum value of* I_1V *is* 8, *and the maximum value of* I_2V *is* 4, *which satisfies* $(I_1V \leq 8) \wedge (I_2V \leq 16)$, *and thus* $C_{r3} \subseteq C_{r12} \cup \overline{B_{r12}}$.

Then, as $B_{r12} \subseteq B_{r3}$, *we get* $\overline{B_{r3}} \subseteq \overline{B_{r12}}$ *and* $\overline{B_{r3}} \subseteq C_{r12} \cup \overline{B_{r12}}$. *Combining the results, we get* $C_{r3} \cup \overline{B_{r3}} \subseteq C_{r12} \cup \overline{B_{r12}}$, *which means the refinement relation holds.*

The refinement relation in the example shows that the composed resistor has the same equivalent resistance but a larger range of working regions than the system specification. As a result, the implementation based on the refinement result never fails if the environment always satisfies the specification of B_3. In this example, we can intuitively understand the intrinsic behaviors using the parallel resistance, but not for the constraint, i.e., the maximum power. If the system $r3$ requires a higher maximum power than 12, the refinement does not hold, as the maximum power constraint of r_1 could be violated.

IV. Constraint-behavior Contracts and Assume-Guarantee contracts

Constraint-behavior contracts and assume-guarantee contracts have different semantics and usage while they share some similarities in their forms and operations. As introduced in Proposition 1, every constraint-behavior contract has an equivalent assume-guarantee contract. Furthermore, the refinement relation and the composition operation of constraint-behavior contracts can be derived from assume-guarantee contracts, though having a slightly different form. This section discusses the similarities and differences between constraint-behavior contracts and assume-guarantee contracts.

A. Semantic and practical usage

Both constraint-behavior contracts and assume-guarantee contracts are defined as a pair of behavior sets $2^{\mathcal{B}_\mathcal{P}} \times 2^{\mathcal{B}_\mathcal{P}}$. However, the two contracts have to be understood and used differently in practical applications for the same behavior sets. In assume-guarantee contracts, the assumption directly specifies the environment behaviors where the component is expected to function, and the guarantee states the component's responsibility in those environments. The assumption checks the working conditions independently of the contract's guarantee. On the other hand, constraint-behavior contracts describe the relationship between ports through intrinsic behaviors, with constraints indicating the conditions under which the established relationship no longer holds. In this case, both the intrinsic behaviors and constraints are involved to derive the uncontrolled part behavior and check the satisfaction of the component's working conditions.

The semantics define different orders for applying the behavior sets, which results in different practical usage. As shown in Section III-A, the implicit port directions from the interpretations of the assume-guarantee contracts make it hard and counter-intuitive to specify physical components.

We refer to these two interpretations of the behavior sets as the assume-guarantee contract semantics and the constraint-behavior contract semantics.

B. Equivalent saturated form in both semantics

According to the refinement relation, we can define the saturation of a constraint-behavior contract to reason about the partial order between the contracts:

Definition 5. *The saturation of a constraint-behavior contract, denoted by the operator* $sat_{cb}()$, *is defined as*

$$sat_{cb}(C, B) = (C \cup \overline{B}, B).$$

The saturation results in a maximal contract, based on partial order defined by refinement, which represents the same specification.

A saturated constraint-behavior contract denotes the same specification as the saturated assume-guarantee contract of the same form, meaning that the interpretation results of the two semantics are common for saturated contracts. We can denote a saturated constraint-behavior contract as $\mathcal{C}^{cb} = (C, B) = (S_1, S_2)$ and a saturated assume-guarantee contract $\mathcal{C}^{ag} = (A, G) = (S_1, S_2)$, where S_1 and S_2 are behavior sets.

In the following, we show their equivalence. First, we show that any pair of behavior sets (S_1, S_2) being a saturated constraint-behavior contract is also a saturated assume-guarantee contract under the assume-guarantee contract semantics, and vice versa:

Lemma 1. *Given a pair of behavior sets* $(S_1, S_2) \in 2^{\mathcal{B}_\mathcal{P}} \times 2^{\mathcal{B}_\mathcal{P}}$, $S_2 = S_2 \cup \overline{S_1}$ *if and only if* $S_1 \cup \overline{S_2} = S_1$.

Proof. $S_2 \cup \overline{S_1} = S_2 \Leftrightarrow \overline{S_1} \subseteq S_2 \Leftrightarrow \overline{S_2} \subseteq S_1 \Leftrightarrow S_1 \cup \overline{S_2} = S_1$. □

Therefore, any saturated constraint-behavior contract is also a saturated assume-guarantee contract if we interpret its pair of behavior sets in the assume-guarantee contract semantics.

Then we show that the saturated contracts in the two semantics represent the same specification using the mathematical meta-theory of contracts [18].

Proposition 2. *Given a pair of behavior sets $(S_1, S_2) \in 2^{\mathcal{B}_\mathcal{P}} \times 2^{\mathcal{B}_\mathcal{P}}$, if $S_2 = S_2 \cup \overline{S_1}$, then the pair expresses the same specification under the constraint-behavior contract semantics and the assume-guarantee contract semantic, i.e., specifying the same sets of environments and implementations.*

Proof. First, we show that the two semantics result in the same environment set: $\mathcal{E}^{cb} = \{E \in \mathcal{B}_\mathcal{P} \mid E \cap S_2 \subseteq S_1\} = \left\{E \;\middle|\; E \subseteq S_1 \cup \overline{S_2} = S_1\right\} = 2^{S_1} = \mathcal{E}^{ag}$.

Then we show that the two semantics represent the same implementation set: $\mathcal{I}^{cb} = \left\{I \subseteq \mathcal{B}_V \;\middle|\; \forall E \in \mathcal{E}^{cb}, I \cap E \subseteq S_2\right\} = \{I \subseteq \mathcal{B}_V \mid I \cap S_1 \subseteq S_2\} = \left\{I \subseteq \mathcal{B}_V \;\middle|\; I \subseteq S_2 \cup \overline{S_1}\right\} = 2^{S_2 \cup \overline{S_1}} = \mathcal{I}^{ag}$, where the second equality is obtained by $E \subseteq S_1$ since $\mathcal{E}^{cb} = 2^{S_1}$. Therefore, the two semantics on the pair of behavior sets express the same specification since they specify identical environment and implementation sets. □

This property allows us to derive the formula for operations of constraint-behavior contracts, like the derivation in Section III-B. Furthermore, algorithms based on saturated assume-guarantee contracts can be applied to saturated constraint-behavior contracts, allowing integration of the two contract semantics.

C. Unsaturated composition with set intersection

The previous parts detail the close relationship between assume-guarantee contracts and constraint-behavior contracts. However, the constraint-behavior contracts have a property that the assume-guarantee contracts do not have: the same composition formula for saturated and unsaturated constraint-behavior contracts.

The composition in Definition 3, which involves only simple set intersections, does not require the contracts to be saturated, and the resulting contract might also be unsaturated. However, assume-guarantee contracts do not possess a similar property. The composition formula introduced in Section II cannot be applied to unsaturated assume-guarantee contracts. Consider unsaturated contracts $\mathcal{C}_1^{ag} = (A_1, G_1)$ and $\mathcal{C}_2^{ag} = (A_2, G_2)$, where A_1, G_1, A_2, and G_2 are unconstrained sets. We can obtain their composition as follows:

$$\begin{aligned}\mathcal{C}_1^{ag} \parallel \mathcal{C}_2^{ag} &= \text{sat}_{\text{ag}}(\mathcal{C}_1^{ag}) \parallel \text{sat}_{\text{ag}}(\mathcal{C}_2^{ag}) \\ &= (A_1, G_1 \cup \overline{A_1}) \parallel (A_2, G_2 \cup \overline{A_2}) \\ &= ((A_1 \cap A_2) \cup (\overline{G_1} \cap A_1) \cup (\overline{G_2} \cap A_2), \\ &\quad (G_1 \cup \overline{A_1}) \cap (G_2 \cup \overline{A_2})).\end{aligned}$$

Note that no further simplifications can be made since all sets are unconstrained. Observing the obtained composition formula, the composition of unsaturated assume-guarantee contracts still involves the saturation operation such as $G_1 \cup \overline{A_1}$ and $G_2 \cup \overline{A_2}$. On the other hand, constraint-behavior contracts have a common composition formula for saturated and unsaturated contracts. Considering two unsaturated contracts $\mathcal{C}_1^{cb} = (C_1, B_1)$ and $\mathcal{C}_2^{cb} = (C_2, B_2)$, where C_1, B_1, C_2, and B_2 are unconstrained sets, we can obtain their composition as follows:

$$\begin{aligned}\mathcal{C}_1^{cb} \parallel_{cb} \mathcal{C}_2^{cb} &= \text{sat}_{\text{cb}}(\mathcal{C}_1^{cb}) \parallel \text{sat}_{\text{cb}}(\mathcal{C}_2^{cb}) \\ &= (C_1 \cup \overline{B_1}, B_1) \parallel (C_2 \cup \overline{B_2}, B_2) \\ &= (((C_1 \cup \overline{B_1}) \cap (C_2 \cup \overline{B_2})) \cup \overline{B_1} \cup \overline{B_2}, \\ &\quad B_1 \cap B_2) \\ &= (((C_1 \cap C_2) \cup \overline{(B_1 \cap B_2)}, B_1 \cap B_2) \\ &= \text{sat}_{\text{cb}}(\mathcal{C}_1^{cb} \parallel_{cb} \mathcal{C}_2^{cb}),\end{aligned}$$

where we first saturate the constraint-behavior contracts to get their equivalent assume-guarantee contacts and then perform the assume-guarantee contract composition.

As a result, constraint-behavior contracts can be composed simply by set intersection, even though they are unsaturated. This property also demonstrates the ease of use of our proposed constraint-behavior contracts. The composition is straightforward and intuitive for designers to express their intentions and identify potential faults.

V. Verification using Constraint-behavior Contracts and Assume-guarantee Contracts

By comparing constraint-behavior and assume-guarantee contracts, we have shown the possibility of integrating contracts from different semantics into the contract-based design process. Based on this, this section introduces a verification methodology that utilizes constraint-behavior contracts and assume-guarantee contracts.

Depending on the application's need, a system may be specified with clear port directions while the underlying components can be expressed using implicit equations. In this case, the designers may choose assume-guarantee contracts to specify the system while utilizing constraint-behavior contracts for its components. Therefore, a verification methodology for such an integration is necessary, as verification of these specifications requires the refinement relation to be established under different contract formalisms.

With Proposition 2, we can derive the refinement relation and verify system specification for this case. First, we show the conditions for contract refinement for an assume-guarantee contract and a constraint-behavior contract as follows:

Definition 6. *Given a constraint-behavior contract $\mathcal{C}^{cb} = (C, B)$ and an assume-guarantee contract $\mathcal{C}^{ag} = (A, G)$. $\mathcal{C}^{cb}$ refines $\mathcal{C}^{ag}$, denoted as $\mathcal{C}^{cb} \preceq \mathcal{C}^{ag}$, if the following condition is satisfied:*

$$A \cap B \subseteq C \cap G.$$

The intuition behind the definition is to ensure that all intrinsic behaviors under the specified environments ($A \cap B$) result in behaviors satisfying the constraints of the components and the system guarantees ($C \cap G$).

We can use Proposition 2 to show that the definition aligns with saturated contract refinement. First, we define the assume-guarantee contract $\mathcal{C}_2^{ag} = (C \cup \overline{B}, B)$, which expresses the same specification as $\mathcal{C}^{cb}$. According to assume-guarantee contract refinement, the conditions for contract refinement are

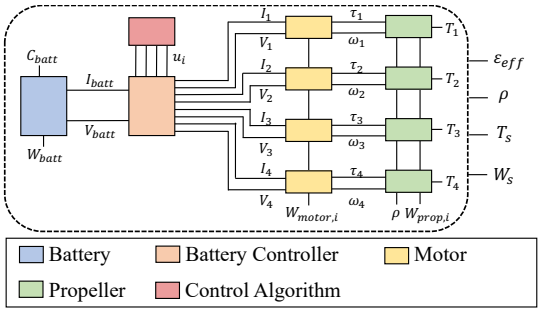

Fig. 4: System Diagram of a UAV propulsion system with four propellers.

$A \subseteq C \cup \overline{B}$ and $B \subseteq G \cup \overline{A}$. By rewriting $A \subseteq C \cup \overline{B}$, we obtain the equivalent condition $A \cap B \subseteq C$. Similarly, rewriting $B \subseteq G \cup \overline{A}$ yields the equivalent condition $A \cap B \subseteq G$. Combining these two results, we get $A \cap B \subseteq C \cap G$, which aligns with the defined refinement relation.

Here we sketch a verification process integrating constraint-behavior contracts and assume-guarantee contracts using the contract refinement as follows: first, the designer specifies the system requirement in assume-guarantee contracts. Then the designer gathers the component specifications written in constraint-behavior contracts and assume-guarantee contracts. After gathering all the contracts, the composed contract is computed by applying contract composition to the component contracts. Finally, the designer can verify the system by checking contract refinement following Definition 6.

VI. Demonstration: UAV Electrical System Design

To demonstrate the effectiveness of constraint-behavior contracts in addressing design problems with physical components, we apply them to a UAV propulsion system verification problem. The system specification includes two requirements: 1) the UAV must tolerate the maximum voltage of the batteries, and 2) the UAV must be able to stay in the air for a given time t_req. The demonstration involves formulating constraint-behavior contracts for the components and utilizing the verification process that combines constraint-behavior contracts with assume-guarantee contracts.

A. System details

The UAV propulsion system is a heterogeneous system spanning the electrical, mechanical, and control domains. Figure 4 shows the system overview. The components in the UAV propulsion system with N propellers include a battery pack, N motors, N propellers, a battery controller, and a battery control algorithm [19]. Each individual component of the same type may have a different component model, which has the same ports while the behaviors are different based on the parameter values of the component model. In the following, we detail each component.

- *Battery.* The battery provides electrical power to the propulsion system. A battery has four ports: C_{batt}, I_{batt}, V_{batt}, and W_{batt}. Port C_{batt} indicates the maximum capacity. I_{batt} is the battery current. V_{batt} is the battery voltage. W_{batt} is the weight of the battery.
- *Motor.* The motor converts the electrical energy from batteries to mechanical energy. The motor has five ports: V_i, I_i, τ_i, ω_i, and $W_{motor,i}$. The subscript i denotes the i^{th} component in the system. Port V_i is the voltage across the motor. I_i is the current passing through the motor. τ_i is the torque of the motor. ω_i is the angular velocity of the motor. $W_{motor,i}$ is the weight of the motor.
- *Propellers.* The propeller produces thrust, an upward force for the UAV to fly against gravity. A propeller has five ports: τ_i, ω_i, T_i, ρ, and $W_{prop,i}$. Port τ_i is the torque of the propeller. ω_i is the angular velocity of the propeller. T_i is the thrust generated by the propeller. ρ is the air density. $W_{prop,i}$ is the weight of the propeller.
- *Battery Controller.* The battery controller determines the power drawn from the batteries and its distribution to the motors. A battery controller contains $3N+2$ ports: V_{batt}, I_{batt}, u_i, V_i, and I_i, for all $i = 1$ to N. Port V_{batt} is the voltage from the battery. Port I_{batt} is the current from the battery. Ports u_i are the control inputs indicating the ratio between V_i and V_{batt}. V_i is the electromotive force provided by the controller. I_i are the current sent by the controller.
- *Control Algorithm.* A control algorithm applies a strategy to control the battery controller such that the UAV can fly to achieve the design goals. Ideally, a control algorithm should send the control signal based on the status of the UAV. However, as we do not focus on the control algorithm in this demonstration, the status can be abstract as any possible values. As a result, the control algorithm contains N ports: u_i for $i = 1$ to N, where u_i is the control value for the voltage to the motor.

B. Contract formulation

As outlined in Section V, the designers first specify the system requirements in assume-guarantee contracts and gather the component specifications in constraint-behavior contracts. Here we show the contract formulation for the system requirements and individual components.

1) System contract: The first system requirement indicates the UAV must be able to fly, i.e., not crash onto the ground, at the maximum battery voltage. Therefore, we can write the assume-guarantee contract $\mathcal{C}^{ag}_{fly} = (A_{fly}, G_{fly})$ as follows:

A_{fly}	G_{fly}
$\rho = 1.225$ $W_s = W_{battery} + W_{body} +$ $\sum_{i=1}^{n}(W_{prop,i} + W_{motor,i})$ $T_s = \sum_{i=1}^{n} T_i$ $u_i = 1 \ \ \forall i = 1...n$	$T_s \geq W_s$

The assumption sets the air density, defines auxiliary variables for total weight and thrust, and then sets the control output to 1 to operate the UAV at the maximum battery voltage. The guarantee requires that the thrust exceeds the weight, preventing the UAV from crashing onto the ground due to insufficient thrust. In the formulation, W_{body} is a parameter for specifying the weight of the UAV frame and its payload.

The second system requirement indicates that the UAV can stay flying for at least t_req seconds before depleting the battery power. Thus, we can formulate the assume-guarantee contract $\mathcal{C}^{ag}_{t\_req} = (A_{t\_req}, G_{t\_req})$ for this requirement as follows:

A_{t_req}	G_{t_req}
$\rho = 1.225$ $W_s = W_{battery} + W_{body} +$ $\sum_{i=1}^{n}(W_{prop,i} + W_{motor,i})$ $T_s = \sum_{i=1}^{n} T_i$ $T_s = W_s$	$I_{batt} \leq \frac{C_{batt}}{t\_req}$

The assumption sets the air density, defines auxiliary variables for total weight and thrust, and then ensures that the thrust is equivalent to the weight to maintain hovering. The guarantee requires that the current withdrawn from the battery can continuously supply power for at least t_{req} seconds before depleting its stored energy.

2) Component contract: After formulating the assume-guarantee contracts for the system requirements, we proceed to specify the behavior of the components by listing the constraint-behavior contract formulations.

a) Battery: The contract for a battery of the component model b, denoted as $\mathcal{C}^{cb}_{batt,b} = (C_{batt,b}, B_{batt,b})$, is formulated as follows:

$C_{batt,b}$	$B_{batt,b}$
$I_{batt} \leq I_{max,b}$	$C_{batt} = C_b$ $V_{batt} = V_b$ $W_{batt} = W_b$

In the above equations, the following parameters are constants specified by the battery model: $I_{max,b}$ is the maximum allowable current, C_b is the capacity, V_b is the voltage, and W_b is the weight of the battery model.

b) Motors: The contract of a motor of the component model m with index i, denoted as $\mathcal{C}^{cb}_{motor,m,i} = (C_{motor,m,i}, B_{motor,m,i})$, is defined as follows:

$C_{motor,m,i}$	$B_{motor,m,i}$
$V_i I_i < P_{max,m}$	$I_i B_{w,m} = \frac{V_i - \omega_i}{K_{v,m}}$
$I_i < I_{max,m}$	$\tau_i = \frac{K_{t,m}}{B_{w,m}}(V_i - B_{w,m} \times I_{idle,m} - \frac{\omega_i}{K_{v,m}})$ $W_{motor,i} = W_m$

In the above equations, the following parameters are constants specified by the motor model: $P_{max,m}$ is the maximum allowable power, $B_{w,m}$ is the internal resistance, $K_{v,m}$ is the motor velocity constant, $I_{idle,m}$ is the idle current, $K_{t,m}$ is the motor torque constant, and W_m is the weight of the motor model.

c) Propellers: The contract of a propeller of the component model p with index i, denoted as $\mathcal{C}^{cb}_{prop,p,i} = (C_{prop,p,i}, B_{prop,p,i})$, is defined as follows:

$C_{prop,p,i}$	$B_{prop,p,i}$
True	$\tau_i = \frac{\rho C_{p,i} \omega_i^2 \times D_p^5}{(2\pi)^3}$ $T_i = \frac{\rho C_{t,i} \omega_i^2 \times D_p^4}{(2\pi)^2}$ $\omega_i \geq 0$ $C_{p,i} \in [C_{pmin,p}, C_{pmax,p}]$ $C_{t,i} \in [C_{tmin,p}, C_{tmax,p}]$ $W_{prop,i} = W_p$

In the above equations, the following parameters are constants specified by the propeller model: D_p is the propeller diameter, $C_{pmin,p}$ and $C_{pmax,p}$ define the range of the power coefficient, $C_{tmin,p}$, $C_{tmax,p}$ define the range of the thrust coefficient, and W_p is the weight of the propeller model.

d) Battery Controller: The contract of the battery controller for battery controller model c, denoted as $\mathcal{C}^{cb}_{batcont,c} = (C_{batcont,c}, B_{batcont,c})$, is defined as follows ($\epsilon_{eff,c}$ is the conversion efficiency of the battery controller):

$C_{batcont,c}$	$B_{batcont,c}$
$V_{motor,i} \leq V_{battery}\ \forall i = 1...n$	$I_{batt} = \epsilon_{eff,c} \sum_{i=1}^{n} I_i$ $V_i = u_i V_{battery}$

e) Control Algorithm: As we abstract sensors and all states of the UAV, the control output can be any value between 1 and 0. The value denotes the ratio of the battery voltage sent to the motor. For example, a control output of 0.4 and a battery voltage of $22.2V$ means the motor gets $8.88V$. Therefore, the contract of the control algorithm, denoted as $\mathcal{C}^{cb}_{cont} = (C_{cont}, B_{cont})$ is defined as follows:

C_{cont}	B_{cont}
True	$u_i \in [0, 1]\ \forall i = 1...n$

C. Designs for verification

The benchmark designs for verification are based on five designs developed under the DARPA SDCPS project [20]. Among the designs, Designs 1 and 3 are manually designed quadcopters, while the remaining designs are randomly generated by specifying a random number of components and assigning random component models. Table I provides a summary of the statistics of the designs and component models used in each design.

D. Verification settings and results

We implemented the verification process of contracts using the SMT solver Z3 [21] in the Python programming language, with polynomial arithmetic as the background theory. For designs with multiple batteries, we modify the contract by multiplying the capacity and weight by the number of batteries and dividing the maximum current by the number of batteries, assuming a parallel connection of the batteries. The parameter t_{req} was set to $200s$, and W_{body} was set to $19.62N$ for all benchmark designs. The results of the verification, including the parameter settings, verification outcomes, and reasons for not passing the requirements, are summarized in Table II. The reason for not passing is obtained by analyzing the counter-example provided by the solver.

	#motor	#battery	Propeller Model	Motor Model	Battery Model
Design1	4	3	apc_propellers_17x6	t_motor_AT4130KV300	TurnigyGraphene6000mAh6S75C
Design2	4	2	apc_propellers_16x6E	t_motor_AT4130KV230	TattuPlus15C16000mAh12S1Pcompact
Design3	4	2	apc_propellers_6x4E	t_motor_AT2312KV1400	TurnigyGraphene1000mAh2S75C
Design4	6	3	apc_propellers_20x10E	t_motor_AT4130KV230	TurnigyGraphene1400mAh4S75C
Design5	4	1	apc_propellers_11x4_6SF	kde_direct_KDE700XF_535_G3	TattuPlus25C22000mAh12S1PAGRI

TABLE I: The statistics of the benchmark designs, including the number of batteries in the battery pack (#battery), the number of motors (#motors), and the component models for propellers, motors, and batteries.

	t_req	W_{body}	$\mathcal{C}^{ag}_{fly}$	$\mathcal{C}^{ag}_{t\_req}$	Reason
Design1	200 s	19.62 N	O	O	
Design2	200 s	19.62 N	X	–	P_motor violated
Design3	200 s	19.62 N	X	–	Not enough thrust
Design4	200 s	19.62 N	O	X	Low capacity
Design5	200 s	19.62 N	O	O	

TABLE II: The requirement parameters (t_req and W_{body}) of the UAV and the verification result. The second contract is denoted as "–" if the fly requirement is not met as there is no need to verify the requirement.

The verification results show that Designs 1 and 5 passed both design requirements, while Designs 2, 3, and 4 violated at least one of the requirements. Design 2 exceeded the maximum power constraint of the motor, resulting in a violation of the guarantee in $\mathcal{C}^{ag}_{fly}$. Design 3 failed to provide sufficient thrust to lift the UAV, thus violating the guarantee of the contract $\mathcal{C}^{ag}_{t\_req}$. Design 4 required a current greater than the specified value and, as a result, failed to provide guarantees in the contract $\mathcal{C}^{ag}_{t\_req}$ to maintain hovering for the required period.

E. Discussion

The constraint-behavior contracts enable contract formulations using implicit equations without considering the port directions. This prevents the need to solve implicit equations, reduces the size of the contract library, and provides an intuitive encoding.

For example, let's consider the contract $\mathcal{C}^{cb}_{motor,m,i}$. To verify the first requirement, the motor's input should be the voltage V_i since the requirement implies that the control parameters are system inputs, which directly control the motor's voltage. On the other hand, to verify the second requirement, the motor's inputs should be the ports connected to the propeller, namely ω_i and τ_i, as the requirement indicates that the thrust is a system input.

By using the constraint-behavior contract for motors, these two requirements can be directly verified without the need to solve implicit equations involving ω_i, I_i, and V_i for various inputs. Additionally, it eliminates the need to store different contracts for the same components solely based on different port directions. As a result, the number of contracts is reduced and independent of the combination of port directions. Furthermore, implicit equations enable designers to formulate contracts using their physical intuition, as the component modeling in [19]. This capability helps prevent specification faults and makes it easier for designers to identify mistakes.

Overall, the demonstration shows the contract formulation, verification process, and the benefits of using constraint-behavior contracts over assume-guarantee contracts.

VII. Related Work

The concept of a contract originated in software engineering, specifically in the context of specification and verification. Meyer [22] introduced the term "contract" in software engineering, making an analogy to business contracts between the caller of a function and the function being specified. In the methodology, preconditions and postconditions differentiate the responsibilities between the caller and the program method. This methodology was later applied to cyber-physical design for system verification and hierarchical reasoning due to its similarities with concurrent programs. Abadi *et al.* [23], [24] represented specifications as assumptions and guarantees for transition systems and introduced a composition principle for assume-guarantee specifications. Building on this, Sangiovanni-Vincentelli *et al.* [3] developed the contract-based design methodology, which leverages contracts in platform-based cyber-physical system designs.

In the fields of software engineering and cyber-physical systems, various contract formalisms have been proposed. Assume-guarantee contracts [10] use assumption and guarantee sets, similar to preconditions and postconditions, to define the responsibilities of the environment and the component, respectively. This formalism has been extended for optimization of controller designs [25], stochastic systems [26], and hyperproperties [27]. Rely-guarantee reasoning [28] extends the concept by adding rely-conditions and guarantee conditions for concurrent programs, where rely-conditions account for changes of global states made by other processes and guarantee conditions define the changes of the global states the programs can make. Input-Output automata [29] describe a component using states, an initial state, actions, and transition relations. The transition relation ensures a property called input-enabled, meaning that every input action is acceptable in any state. Interface automata [30] are an extension of Input-Output Automata that do not require the transition relation to be input-enabled, allowing the separation of responsibilities between the environment and the component. Despite their diverse forms and applications, these contract formalisms are elegantly expressed in a unified algebraic theory by Benveniste *et al.* [18]. We refer interested readers to the monograph for the detailed mathematical definition of contracts and the connections between the contract formalisms mentioned above.

However, most contract formalisms, except for assume-guarantee contracts, explicitly require port directions, assuming

receptive systems. Although assume-guarantee contracts are more general, the implicit port directions make it counter-intuitive to express assumptions as implicit equations containing all ports. Therefore, this work presents an intuitive approach for specifying physical components in implicit equations without the need to solve the equations based on port directions.

VIII. Conclusion

We have presented constraint-behavior contracts as specifications for physical components in cyber-physical systems. Unlike assume-guarantee contracts, the intuitive implicit equations in the constraint-behavior contract eliminate the need for equation solving and reduce the number of required contracts. With the developed properties, the proposed verification process can integrate specifications in constraint-behavior contracts and assume-guarantee contracts. The demonstration based on the UAV propulsion system design problem has provided examples of contract formulation, the verification process in an actual design problem, and the benefits brought by the capability of implicit equations.

References

[1] E. A. Lee and S. A. Seshia, *Introduction to embedded systems: A cyber-physical systems approach*. MIT Press, 2016.

[2] A. Sangiovanni-Vincentelli, "Quo vadis, SLD? reasoning about the trends and challenges of system level design," *Proceedings of the IEEE*, vol. 95, no. 3, pp. 467–506, 2007.

[3] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming Dr. Frankenstein: Contract-based design for cyber-physical systems," *European journal of control*, vol. 18, no. 3, pp. 217–238, 2012.

[4] P. Nuzzo, "From electronic design automation to cyber-physical system design automation: A tale of platforms and contracts," in *Proceedings of the International Symposium on Physical Design (ISPD)*, p. 117–121, 2019.

[5] S. A. Seshia, S. Hu, W. Li, and Q. Zhu, "Design automation of cyber-physical systems: Challenges, advances, and opportunities," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, no. 9, pp. 1421–1434, 2017.

[6] P. Nuzzo, H. Xu, N. Ozay, J. B. Finn, A. L. Sangiovanni-Vincentelli, R. M. Murray, A. Donzé, and S. A. Seshia, "A contract-based methodology for aircraft electric power system design," *IEEE Access*, vol. 2, pp. 1–25, 2013.

[7] W. Damm, H. Hungar, B. Josko, T. Peikenkamp, and I. Stierand, "Using contract-based component specifications for virtual integration testing and architecture design," in *Design, Automation & Test in Europe Conference Exhibition (DATE)*, pp. 1–6, 2011.

[8] P. Nuzzo, A. Sangiovanni-Vincentelli, X. Sun, and A. Puggelli, "Methodology for the design of analog integrated interfaces using contracts," *IEEE Sensors Journal*, vol. 12, no. 12, pp. 3329–3345, 2012.

[9] S. Spellini, R. Chirico, M. Panato, M. Lora, and F. Fummi, "Virtual prototyping a production line using assume–guarantee contracts," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 9, pp. 6294–6302, 2021.

[10] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis, "Multiple viewpoint contract-based specification and design," in *International Symposium on Formal Methods for Components and Objects*, pp. 200–225, 2007.

[11] I. Incer, A. Sangiovanni-Vincentelli, C.-W. Lin, and E. Kang, "Quotient for assume-guarantee contracts," in *2018 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, pp. 1–11, IEEE, 2018.

[12] R. Passerone, I. Incer, 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. 5, pp. 1–23, 2019.

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

[14] P. Fritzson and V. Engelson, "Modelica-a unified object-oriented language for system modeling and simulation," in *ECOOP*, vol. 98, pp. 67–90, Citeseer, 1998.

[15] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[16] J. C. Willems, "The behavioral approach to open and interconnected systems," *IEEE Control Systems Magazine*, vol. 27, no. 6, pp. 46–99, 2007.

[17] The Mathwork Inc., "Simulink." Available at https://www.mathworks.com/products/simulink.html.

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

[19] J. D. Walker, F. M. Heim, B. Surampudi, P. Bueno, A. Carpenter, S. Chocron, J. Cutshall, R. Lammons, T. Bapty, B. Swenson, and S. Whittington, "A flight dynamics model for exploring the distributed electrical evtol cyber physical design space," in *2022 IEEE Workshop on Design Automation for CPS and IoT (DESTION)*, pp. 7–12, 2022.

[20] DARPA, "SDCPS Project." Available at https://www.darpa.mil/program/symbiotic-design-for-cyber-physical-systems.

[21] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," TACAS'08/ETAPS'08, p. 337–340, 2008.

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

[23] M. Abadi, L. Lamport, and P. Wolper, "Realizable and unrealizable specifications of reactive systems," in *International Colloquium on Automata, Languages, and Programming*, pp. 1–17, 1989.

[24] M. Abadi and L. Lamport, "Composing specifications," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 15, no. 1, pp. 73–132, 1993.

[25] C. Oh, E. Kang, S. Shiraishi, and P. Nuzzo, "Optimizing assume-guarantee contracts for cyber-physical system design," in *Design, Automation & Test in Europe Conference Exhibition (DATE)*, pp. 246–251, 2019.

[26] J. Li, P. Nuzzo, A. Sangiovanni-Vincentelli, Y. Xi, and D. Li, "Stochastic contracts for cyber-physical system design under probabilistic requirements," in *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*, pp. 5–14, 2017.

[27] I. Incer, A. Benveniste, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Hypercontracts," in *NASA Formal Methods*, pp. 674–692, 2022.

[28] C. B. Jones, "Tentative steps toward a development method for interfering programs," *ACM Transactions on Programming Languages and Systems*, vol. 5, p. 596–619, oct 1983.

[29] N. A. Lynch and M. R. Tuttle, "Hierarchical correctness proofs for distributed algorithms," in *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '87, (New York, NY, USA), p. 137–151, Association for Computing Machinery, 1987.

[30] L. de Alfaro and T. A. Henzinger, "Interface automata," in *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-9, (New York, NY, USA), p. 109–120, Association for Computing Machinery, 2001.