Hypercontracts

Inigo Incer[1], Albert Benveniste[2], Alberto Sangiovanni-Vincentelli[1], and Sanjit A. Seshia[1]

[1] University of California, Berkeley, USA
[2] INRIA/IRISA, Rennes, France

Abstract. Contract theories have been proposed to formally support distributed and decentralized system design while ensuring safe system integration. In this paper we propose *hypercontracts*, a generic model with a richer structure for its underlying model of components, subsuming simulation preorders. While this new model remains generic, it provides a much more elegant and richer algebra for its key notions of refinement, parallel composition, and quotient, and it allows inclusion of new operations. On top of these foundations, we propose *conic hypercontracts*, which are still generic but come with a finite description.

1 Introduction

The need for compositional algebraic frameworks to design and analyze reactive systems is widely accepted. The aim is to support distributed and decentralized system design based on a proper definition of *interfaces* supporting the specification of subsystems having a partially specified context of operation, and subsequently guaranteeing safe system integration. Over the last few decades, we have seen the introduction of several formalisms to do this: interface automata [9,10,11,20,6], process spaces [22], modal interfaces [17,19,18,27,3], assume-guarantee (AG) contracts [4], rely-guarantee reasoning [15,16,8,13], and variants of these. The interface specifications state (*i*) what the component guarantees and (*ii*) what it assumes from its environment in order for those guarantees to hold, i.e., all these frameworks implement a form of assume-guarantee reasoning.

These algebraic frameworks have a notion of a component, of an environment, and of a specification, also called contract to stress the give-and-take dynamics between the component and its environment. They all have notions of satisfaction of a specification by a component, and of contract composition. Since many contract frameworks have been proposed, there have also been efforts to systematize this knowledge by building high-level theories of which existing contract theories are instantiations. Thus, Bauer et al. [2] describe how to build a contract theory if one has a specification theory available. Benveniste et al. [5] provide a meta-theory that builds contracts starting from an algebra of components. They provide several operations on contracts and show how this meta-theory can describe, among others, interface automata, assume-guarantee contracts, modal interfaces, and rely-guarantee reasoning. This meta-theory is, however, low-level, specifying contracts as unstructured sets of environments and implementations. As a consequence, important concepts such as parallel composition and quotient of contracts are expressed in terms that are considered too abstract—see [5], chapter 4. For example, no closed form formula is given for the quotient besides its abstract definition as adjoint of parallel composition. This paper introduces a theory, called *hypercontracts*, that will address these drawbacks.

Among the various contract theories proposed so far, assume-guarantee (AG) contracts [4] require users to state the assumptions and guarantees of the specification explicitly, while interface theories express a specification as a game played between the specification environments and implementations. Experience tells that engineers in industry find the explicit expression of a contract's assumptions and guarantees natural (see [5] chapter 12), while interface theories are perceived as a less intuitive mechanism for writing specifications; however, interface theories in general come with the most efficient algorithms, making them excellent candidates for internal representations of specifications. Some authors ([5] chapter 10) have therefore proposed to translate contracts expressed as pairs (assumptions, guarantees) into some interface model, where algorithms are applied. This approach has the drawback that results cannot be traced back to the original (assumptions, guarantees) formulation.

One of the difficulties with AG contracts is they only support environments and implementations that can be expressed using trace properties. While many attributes of interest can be expressed using trace properties, there are many important system attributes, such as non-interference, that need hyperproperties to be expressed. Hypercontracts allow environments and implementations to be expressed using arbitrary hyperproperties.

To elaborate on this point, the most basic definition of a property in the formal methods community is “a set of traces.” This notion is based on the *behavioral* approach to system modelling, in which we assume an underlying set of behaviors $\mathcal{B}$, and properties are defined as subsets of $\mathcal{B}$. In this approach, design elements or components are also defined as subsets of $\mathcal{B}$. The difference between components and properties is semantics: a component collects the behaviors that can be observed from that component, while a property collects the behaviors meeting some criterion of interest. We say a component M satisfies a property P, written $M \models P$, when $M \subseteq P$, that is, when each behavior of M is in the set of behaviors satisfying P. Properties of this sort are also called *trace properties*. Many design qualities are of this type, such as *safety*. But there are many system attributes that can only be determined by analyzing multiple traces such as mean response times, security attributes, and reliability. This suggests the need for a richer formalism for expressing design attributes: hyperproperties [7].

Hyperproperties are subsets of $2^{\mathcal{B}}$. Recall that each element of $2^{\mathcal{B}}$ defines a semantically-unique component. Thus, a component M satisfies a hyperproperty H if $M \in H$. One of our key contributions is an assume-guarantee theory that supports the expression of arbitrary hyperproperties. As we present our theory, we will use the following running example.

Example 1 (Running example). Consider the digital system shown in Figure 1a; this system is similar to those presented in [26,21] to illustrate the non-interference property in security. Here, we have an s-bit secret data input S and an n-bit public input P. The system has an output O. There is also an input H that is equal to zero when the system is being accessed by a user with low-privileges, i.e., a user not allowed to use the secret data, and equal to one otherwise. We wish the overall system to satisfy the property that for all environments with $H = 0$, the implementations can only make the output O depend on P, the public data, not on the secret input S.

A prerequisite for writing this requirement is to be able to express the property that "the output O depends on P, the public data, not on the secret input S". We claim that this property cannot be captured by a trace property. To see this, suppose for simplicity that all variables are 1-bit-long. A trace property that aims at expressing the independence from the secret for $O = P$ is

$$P = \left\{ \begin{array}{l} (P = 1, S = 1, O = 1), \\ (P = 0, S = 1, O = 0), \\ (P = 1, S = 0, O = 1), \\ (P = 0, S = 0, O = 0) \end{array} \right\} .$$

A valid implementation $M \subseteq P$ is the following set of traces:

$$M = \left\{ \begin{array}{l} (P = 1, S = 1, O = 1), \\ (P = 0, S = 0, O = 0) \end{array} \right\} .$$

However, the component M leaks the value of S in its output. We conclude that the independence does not behave as a trace property, and therefore, neither does non-interference. To overcome this, simply list all the subsets of P that satisfy the independence requirement:

$$\left\{ \begin{array}{l} (P{=}1, S{=}1, O{=}1), \\ (P{=}0, S{=}1, O{=}0), \\ (P{=}1, S{=}0, O{=}1), \\ (P{=}0, S{=}0, O{=}0) \end{array} \right\}, \left\{ \begin{array}{l} (P{=}1, S{=}1, O{=}1), \\ (P{=}1, S{=}0, O{=}1) \end{array} \right\}, \left\{ \begin{array}{l} (P{=}0, S{=}1, O{=}0), \\ (P{=}0, S{=}0, O{=}0) \end{array} \right\}$$

This precisely defines a subset of $2^{\mathcal{B}}$, i.e., a hyperproperty.

In our development, we will use hypercontracts first to express this top-level, assume-guarantee requirement, and then to find a component that added to a partial implementation of the system results in a design that meets the top-level specification. $\square$

Contributions. In this paper, we provide a theory of contracts, called *hypercontracts*, which generalizes existing theories of contracts while treating assumptions and guarantees as first-class citizens. This new AG theory supports arbitrary structured hyperproperties, including, e.g., non-interference and robustness.

The theory of hypercontracts is built in three stages. We begin with a theory of components. Then we state what are the sets of components that our theory

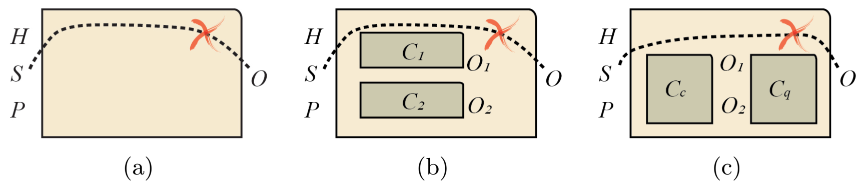

Fig. 1: (a) A digital system with a secret input S and a public input P. The overall system must meet the requirement that the secret input does not affect the value of the output O when the signal H is deasserted (this signal is asserted when a privileged user uses the system). Our agenda for this running example is the following: (b) we will start with two components C_1 and C_2 satisfying respective hypercontracts $\mathcal{C}_1$ and $\mathcal{C}_2$ characterizing information-flow properties of their own; (c) the composition of these two hypercontracts, $\mathcal{C}_c$, will be derived. Through the quotient hypercontract $\mathcal{C}_q$, we will discover the functionality that needs to be added in order for the design to meet the top-level information-flow specification $\mathcal{C}$.

can express; we call such objects compsets—compsets boil down to hyperproperties in behavioral formalisms [21]. From these compsets, we build hypercontracts. We provide closed-form expressions for hypercontract manipulations. Then we show how the hypercontract theory applies to two specific cases: downward-closed hypercontracts and interface hypercontracts (equivalent to interface automata). The main difference between hypercontracts and the meta-theory of contracts [5] is that hypercontracts are more structured: the meta-theory of contracts defines a theory of components, and uses these components in order to define contracts. Hypercontracts use the theory of components to define compsets, which are the types of properties that we are interested in representing in a specific theory. Hypercontracts are built out of compsets, not out of components.

To summarize, our contributions are the following: (i) a new model of *hypercontracts* possessing a richer algebra than the metatheory of [5] and capable of expressing any lattice of hyperproperties and (ii) a calculus of *conic hypercontracts* offering finite representations of downward-closed hypercontracts.

2 Preliminaries

Many concepts in this paper will be inherited from **preorders**. We recall that a preorder $(P, \leq)$ consists of a set P and a relation $\leq$ which is transitive (i.e., $a \leq b$ and $b \leq c$ implies that $a \leq c$ for all $a, b, c \in P$) and reflexive ($a \leq a$ for all $a \in P$). A partial order is a preorder whose relation is also antisymmetric (i.e., from $a \leq b$ and $b \leq a$ we conclude that $a = b$).

Our preorders will come equipped with a partial binary operation called composition, usually denoted $\times$. Composition is often understood as a means of connecting elements together and is assumed to be monotonic in the preorder, i.e., we assume composing with bigger elements yields bigger results: $\forall a, b, c \in P.\ a \leq b \Rightarrow a \times c \leq b \times c$. We will also be interested in taking elements apart. For a notion of composition, we can always ask the question, for $a, b \in P$, what is the largest element $b \in P$ such that $a \times b \leq c$? Such an element is called *quotient* or *residual*, usually denoted c/a. Formally, the definition of the quotient c/a is

$$\forall b \in P.\ a \times b \leq c \text{ if and only if } b \leq c/a, \tag{1}$$

which means that the quotient is the right adjoint of composition (in the sense of category theory). A synonym of this notion is to say that composing by a fixed element a (i.e., $b \mapsto a \times b$) and taking quotient by the same element (i.e., $c \mapsto c/a$) form a Galois connection. A description of the use of the quotient in many fields of engineering and computer science is given in [14].

A partial order for which every two elements have a well-defined LUB (aka join), denoted $\vee$, and GLB (aka meet), denoted $\wedge$, is a lattice. A lattice in which the meet has a right adjoint is called Heyting algebra. This right adjoint usually goes by the name exponential, denoted $\rightarrow$. In other words, the exponential is the notion of quotient if we take composition to be given by the meet, that is, for a Heyting algebra H with elements a, c, the exponential is defined as

$$\forall b \in H.\ a \wedge b \leq c \text{ if and only if } b \leq a \rightarrow c, \tag{2}$$

which is the familiar notion of implication in Boolean algebras.

3 The theory of hypercontracts

Our objective is to develop a theory of assume-guarantee reasoning for any kind of attribute of reactive systems. We do this in three steps:

1. we consider components coming with notions of preorder (e.g., simulation) and parallel composition;
2. we discuss the notion of a compset and give it substantial algebraic structure—unlike the unstructured sets of components considered in the metatheory of [5];
3. we build hypercontracts as pairs of compsets with additional structure—capturing environments and implementations.

In this section we describe how this construction is performed, and in the next we show specialized hypercontract theories.

3.1 Components

In the theory of hypercontracts, the most primitive concept is the component. Let $(\mathbb{M}, \leq)$ be a preorder. The elements $M \in \mathbb{M}$ are called *components*. We say that M is a subcomponent of M' when $M \leq M'$. If we represented components as automata, the statement "is a subcomponent of" is equivalent to "is simulated by."

There exists a partial binary operation, $\|: \mathbb{M}, \mathbb{M} \to \mathbb{M}$, monotonic in both arguments, called *composition*. If $M \parallel M'$ is not defined, we say that M and M' are *non-composable* (and *composable* otherwise). A component E is an environment for component M if E and M are composable. We assume that composition is associative and commutative.

Example 2 (running example, cont'd). In order to reason about possible decompositions of the system shown in Figure 1a, we introduce the internal variables O_1 and O_2, as shown in Figure 1b. They have lengths o_1 and o_2, respectively. The output O has length o. For simplicity, we will assume that the behaviors of the entire system are stateless. In that case, the set of components $\mathbb{M}$ is the union of the following sets:

– For $i \in \{1, 2\}$, components with inputs H, S, P, and output O_i, i.e., the sets $\{(H, S, P, O_1, O_2, O) \mid \exists f \in (2^1 \times 2^s \times 2^n \to 2^{o_i}).\, O_i = f(H, S, P)\}$.
– Components with inputs H, S, P, O_1, O_2, and output O, i.e., the set $\{(H, S, P, O_1, O_2, O) \mid \exists f \in (2^1 \times 2^s \times 2^n \times 2^{o_1} \times 2^{o_2} \to 2^o).\, O = f(H, S, P, O_1, O_2)\}$. We also consider components any subset of these components, as these correspond to restricting inputs to subsets of their domains.

In this theory of components, composition is carried out via set intersection. So for example, if for $i \in \{1, 2\}$ we have functions $f_i \in (2^1 \times 2^s \times 2^n \to 2^{o_i})$ and components $M_i = \{(H, S, P, O_1, O_2, O) \mid O_i = f_i(H, S, P)\}$, the composition of these objects is

$$M_1 \parallel M_2 = \left\{ (H, S, P, O_1, O_2, O) \,\middle|\, \begin{matrix} O_1 = f_1(H, S, P) \\ O_2 = f_2(H, S, P) \end{matrix} \right\},$$

which is the set intersection of the components's behaviors. □

3.2 Compsets

CmpSet is a lattice whose objects are sets of components, called *compsets*. Thus, compsets are equivalent to hyperproperties when the underlying component theory represents components as sets of behaviors. In general, not every set of components is necessarily an object of **CmpSet**.

CmpSet comes with a notion of satisfaction. Suppose $M \in \mathbb{M}$ and H is a compset. We say that M *satisfies* H or conforms to H, written $M \models H$, when $M \in H$. For compsets H, H', we say that H *refines* H', written $H \leq H'$, when $M \models H \Rightarrow M \models H'$, i.e., when $H \subseteq H'$.

Since we assume **CmpSet** is a lattice, the greatest lower bounds and least upper bounds of finite sets are defined. Observe, however, that although the partial order of **CmpSet** is given by subsetting, the meet and join of **CmpSet** are not necessarily intersection and union, respectively, as the union or intersection of any two elements are not necessarily elements of **CmpSet**.

Example 3 (Running example: non-interference). Non-interference, introduced by Goguen and Meseguer [12], is a common information-flow attribute, a prototypical example of a design quality which trace properties are unable to capture [7]. It can be expressed with hyperproperties, and is in fact one reason behind their introduction.

Suppose σ is one of the behaviors that our system can display, understood as the state of memory locations through time. Some of those memory locations we call *privileged*, some *unprivileged*. Let $L_0(\sigma)$ and $L_f(\sigma)$ be the projections of the behavior σ to the unprivileged memory locations of the system, at time zero, and at the final time (when execution is done). We say that a component M meets the non-interference hyperproperty when

$$\forall \sigma, \sigma' \in M.\ L_0(\sigma) = L_0(\sigma') \Rightarrow L_f(\sigma) = L_f(\sigma'),$$

i.e., if two traces begin with the unprivileged locations in the same state, the final state of the unprivileged locations matches.

Non-interference is a downward-closed hyperproperty [26,21], and a 2-safety hyperproperty—hyperproperties called *k-safety* are those for the refutation of which one must provide at least k traces. In our example, to refute the hyperproperty, it suffices to show two traces that share the same unprivileged initial state, but which differ in the unprivileged final state.

Regarding the system shown in Figure 1a, we require the top level component to generate the output O independently from the secret input S. We build our theory of compsets by letting the set $2^{\mathbb{M}}$ be the set of elements of **CmpSet**. This means that any set of components is a valid compset. The components meeting the top-level non-interference property are those belonging to the compset $\{(H, S, P, O_1, O_2, O) \mid \exists f \in (2^1 \times 2^n \to 2^o).\, O = f(H, P)\}$, i.e., those components for which H and P are sufficient to evaluate O. This corresponds exactly to those components that are insensitive to the secret input S. The join and meet of these compsets is given by set union and intersection, respectively. □

Composition and quotient. We extend the notion of composition to **CmpSet**:

$$H \parallel H' = \left\{ M \parallel M' \;\middle|\; \begin{array}{l} M \models H,\, M' \models H', \text{ and} \\ M \text{ and } M' \text{ are composable} \end{array} \right\}. \tag{3}$$

Composition is total and monotonic, i.e., if $H' \leq H''$, then $H \parallel H' \leq H \parallel H''$. It is also commutative and associative, by the commutativity and associativity, respectively, of component composition.

We assume the existence of a second (but partial) binary operation on the objects of **CmpSet**. This operation is the right adjoint of composition: for compsets H and H', the residual H/H' (also called *quotient*), is defined by the universal property (1). From the definition of composition, we must have

$$H/H' = \{M \in \mathbb{M} \mid \{M\} \parallel H' \subseteq H\}. \tag{4}$$

Downward-closed compsets. The set of components was introduced with a partial order. We say that a compset H is *downward-closed* when $M' \leq M$

and $M \models H$ imply $M' \models H$, i.e., if a component satisfies a downward-closed compset, so does its subcomponent. Section 4.2 treats downward-closed compsets in detail.

3.3 Hypercontracts

Hypercontracts as pairs (environments, closed-system specification). A hypercontract is a specification for a design element that tells what is required from the design element when it operates in an environment that meets the expectations of the hypercontract. A hypercontract is thus a pair of compsets:

$$\mathcal{C} = (\mathcal{E}, \mathcal{S}) = (\text{environments, closed-system specification}).$$

$\mathcal{E}$ states the environments in which the object being specified must adhere to the specification. $\mathcal{S}$ states the requirements that the design element must fulfill when operating in an environment which meets the expectations of the hypercontract. We say that a component E *is an environment of hypercontract* $\mathcal{C}$, written $E \models^E \mathcal{C}$, if $E \models \mathcal{E}$. We say that a component M *is an implementation of* $\mathcal{C}$, written $M \models^I \mathcal{C}$, when $M \parallel E \models \mathcal{S}$ for all $E \models \mathcal{E}$. We thus define the set of implementations $\mathcal{I}$ of $\mathcal{C}$ as the compset containing all implementations, i.e., as the quotient:

$$\text{implementations} = \mathcal{I} = \mathcal{S}/\mathcal{E}.$$

A hypercontract with a nonempty set of environments is called *compatible*; if it has a nonempty set of implementations, it is called *consistent.* For $\mathcal{S}$ and $\mathcal{I}$ as above, the compset $\mathcal{E}'$ defined as $\mathcal{E}' = \mathcal{S}/\mathcal{I}$ contains all environments in which the implementations of $\mathcal{C}$ satisfy the specifications of the hypercontract. Thus, we say that a hypercontract is saturated if its environments compset is as large as possible in the sense that adding more environments to the hypercontract would reduce its implementations. This means that $\mathcal{C}$ satisfies the following fixpoint equation: $\mathcal{E} = \mathcal{S}/\mathcal{I} = \mathcal{S}/(\mathcal{S}/\mathcal{E})$.

At a first sight, this notion of saturation may seem to go against what for assume-guarantee contracts are called contracts in canonical or saturated form, as we make the definition based on the environments instead of on the implementations. However, the two definitions for AG contracts and hypercontracts agree. Indeed, for AG contracts, this notion means that the contract $\mathcal{C} = (A, G)$ satisfies $G = G \cup \neg A$. For this AG contract, we can form a hypercontract as follows: if we take the set of environments to be $\mathcal{E} = 2^A$ (i.e., all subsets of A) and the closed system specs to be $\mathcal{S} = 2^G$, we get a hypercontract whose set of implementations is $2^{G \cup \neg A}$, which means that the hypercontract $(2^A, 2^G)$ is saturated.

Hypercontracts as pairs (environments, implementations). Another way to interpret a hypercontract is by telling explicitly which environments and implementations it supports. Thus, we would write the hypercontract as $\mathcal{C} = (\mathcal{E}, \mathcal{I})$. Assume-guarantee theories can differ as to the most convenient representation for their hypercontracts. Moreover, some operations on hypercontracts find their most convenient expression in terms of implementations (e.g., parallel composition), and some in terms of the closed system specifications (e.g., strong merging).

The lattice **Contr** *of hypercontracts.* Just as with **CmpSet**, we define **Contr** as a lattice formed by putting together two compsets in one of the above two ways. Not every pair of compsets is necessarily a valid hypercontract. We will define soon the operations that give rise to this lattice.

Preorder. We define a preorder on hypercontracts as follows: we say that $\mathcal{C}$ *refines* $\mathcal{C}'$, written $\mathcal{C} \le \mathcal{C}'$, when every environment of $\mathcal{C}'$ is an environment of $\mathcal{C}$, and every implementation of $\mathcal{C}$ is an implementation of $\mathcal{C}'$, i.e., $E \models^E \mathcal{C}' \Rightarrow E \models^E \mathcal{C}$ and $M \models^I \mathcal{C} \Rightarrow M \models^I \mathcal{C}'$. We can express this as

$$\mathcal{E}' \le \mathcal{E} \text{ and } \mathcal{S}/\mathcal{E} = \mathcal{I} \le \mathcal{I}' = \mathcal{S}'/\mathcal{E}'.$$

Any two $\mathcal{C}, \mathcal{C}'$ with $\mathcal{C} \le \mathcal{C}'$ and $\mathcal{C}' \le \mathcal{C}$ are said to be *equivalent* since they have the same environments and the same implementations. We now obtain some operations using preorders which are defined as the LUB or GLB of **Contr**. We point out that the expressions we obtain are unique up to the preorder, i.e., up to hypercontract equivalence.

GLB and LUB. From the preorder just defined, the GLB of $\mathcal{C}$ and $\mathcal{C}'$ satisfies: $M \models^I \mathcal{C} \wedge \mathcal{C}'$ if and only if $M \models^I \mathcal{C}$ and $M \models^I \mathcal{C}'$; and $E \models^E \mathcal{C} \wedge \mathcal{C}'$ if and only if $E \models^E \mathcal{C}$ or $E \models^E \mathcal{C}'$.

Conversely, the least upper bound satisfies $M \models^I \mathcal{C} \vee \mathcal{C}'$ if and only if $M \models^I \mathcal{C}$ or $M \models^I \mathcal{C}'$, and $E \models^E \mathcal{C} \vee \mathcal{C}'$ if and only if $E \models^E \mathcal{C}$ and $E \models^E \mathcal{C}'$.

The lattice **Contr** has hypercontracts for objects (up to contract equivalence), and meet and join as just described.

Parallel composition. The composition of hypercontracts $\mathcal{C}_i = (\mathcal{E}_i, \mathcal{I}_i)$ for $1 \le i \le n$, denoted $\|_i\ \mathcal{C}_i$, is the smallest hypercontract $\mathcal{C}' = (\mathcal{E}', \mathcal{I}')$ (up to equivalence) meeting the following requirements:

– any composition of implementations of all $\mathcal{C}_i$ is an implementation of $\mathcal{C}'$; and
– for any $1 \le j \le n$, any composition of an environment of $\mathcal{C}'$ with implementations of all $\mathcal{C}_i$ (for $i \ne j$) yields an environment for $\mathcal{C}_j$.

These requirements were stated for the first time by Abadi and Lamport [1]. Using our notation, this composition principle becomes

$$\begin{aligned}\mathcal{C} \parallel \mathcal{C}' &= \bigwedge \left\{ \begin{matrix} (\mathcal{E}', \mathcal{I}') \\ \in \textbf{Contr} \end{matrix} \;\middle|\; \begin{bmatrix} \mathcal{I}_1 \parallel \ldots \parallel \mathcal{I}_n \le \mathcal{I}', \text{ and} \\ \mathcal{E}' \parallel \mathcal{I}_1 \parallel \ldots \parallel \hat{\mathcal{I}}_j \parallel \ldots \parallel \mathcal{I}_n \le \mathcal{E}_j \\ \text{for all } 1 \le j \le n \end{bmatrix} \right\} \\ &= \bigwedge \left\{ \begin{matrix} (\mathcal{E}', \mathcal{I}') \\ \in \textbf{Contr} \end{matrix} \;\middle|\; \begin{bmatrix} \mathcal{I}_1 \parallel \ldots \parallel \mathcal{I}_n \le \mathcal{I}', \text{ and} \\ \mathcal{E}' \le \bigwedge_{1 \le j \le n} \frac{\mathcal{E}_j}{\mathcal{I}_1 \parallel \ldots \parallel \hat{\mathcal{I}}_j \parallel \ldots \parallel \mathcal{I}_n} \end{bmatrix} \right\},\end{aligned} \tag{5}$$

where the notation $\hat{\mathcal{I}}_j$ indicates that the composition $\mathcal{I}_1 \parallel \ldots \parallel \hat{\mathcal{I}}_j \parallel \ldots \parallel \mathcal{I}_n$ includes all terms $\mathcal{I}_i$, except for $\mathcal{I}_j$.

Example 4 (Running example, parallel composition). Coming back to the example shown in Figure 1, we want to state a requirement for the top-level component that for all environments with $H = 0$, the implementations can only make the output O depend on P, the public data. We will write a hypercontract for the top-level. We let $\mathcal{C} = (\mathcal{E}, \mathcal{I})$, where

$$\begin{aligned}\mathcal{E} =& \{M \in \mathbb{M} \mid \forall (H, S, P, O_1, O_2, O) \in M. H = 0\}\\ \mathcal{I} =& \{M \in \mathbb{M} \mid \exists f \in (2^n \to 2^o). \forall (H, S, P, O_1, O_2, O) \in M. H = 0 \to O = f(P)\}.\end{aligned}$$

The environments are all those components only defined for $H = 0$. The implementations are those such that the output is a function of P when $H = 0$.

Let $f^* : 2^n \to 2^o$. Suppose we have two hypercontracts that require their implementations to satisfy the function $O_i = f^*(P)$, one implements it when $S = 0$, and the other when $S \neq 0$. For simplicity of syntax, let s_1 and s_2 be the propositions $S = 0$ and $S \neq 0$, respectively. Let the two hypercontracts be $\mathcal{C}_i = (\mathcal{E}_i, \mathcal{I}_i)$ for $i \in \{1, 2\}$. We won't place restrictions on the environments for these hypercontracts, so we obtain $\mathcal{E}_i = \mathbb{M}$ and

$$\mathcal{I}_i = \{M \in \mathbb{M} \mid \forall (H, S, P, O_1, O_2, O) \in M. s_i \to O_i = f^*(P)\}.$$

We now evaluate the composition of these two hypercontracts: $\mathcal{C}_c = \mathcal{C}_1 \parallel \mathcal{C}_2 = (\mathcal{E}_c, \mathcal{I}_c)$, yielding $\mathcal{E}_c = \mathbb{M}$ and

$$\begin{aligned}\mathcal{I}_c = \{M \in \mathbb{M} \mid& \forall (H, S, P, O_1, O_2, O) \in M.\\ & (s_1 \to O_1 = f^*(P)) \wedge (s_2 \to O_2 = f^*(P))\}.\end{aligned}$$

Mirror or reciprocal. We assume we have an additional operation on hypercontracts, called both mirror and reciprocal, which flips the environments and implementations of a hypercontract: $\mathcal{C}^{-1} = (\mathcal{E}, \mathcal{I})^{-1} = (\mathcal{I}, \mathcal{E})$ and $\mathcal{C}^{-1} = (\mathcal{E}, \mathcal{S})^{-1} = (\mathcal{S}/\mathcal{E}, \mathcal{S})$. This notion gives us, so to say, the hypercontract obeyed by the environment. The introduction of this operation assumes that for every hypercontract $\mathcal{C}$, its reciprocal is also an element of **Contr**. Moreover, we assume that, when the infimum of a collection of hypercontracts exists, the following identity holds:

$$\left(\bigwedge_i \mathcal{C}_i\right)^{-1} = \bigvee_i \mathcal{C}_i^{-1}. \tag{6}$$

Hypercontract quotient. The *quotient* or residual for hypercontracts $\mathcal{C} = (\mathcal{E}, \mathcal{I})$ and $\mathcal{C}'' = (\mathcal{E}'', \mathcal{I}'')$, written $\mathcal{C}''/\mathcal{C}$, has the universal property (1), namely $\forall \mathcal{C}'.\ \mathcal{C} \parallel \mathcal{C}' \leq \mathcal{C}''$ if and only if $\mathcal{C}' \leq \mathcal{C}''/\mathcal{C}$. We can obtain a closed-form expression using the reciprocal:

Proposition 1. *The hypercontract quotient obeys* $\mathcal{C}''/\mathcal{C} = \left((\mathcal{C}'')^{-1} \parallel \mathcal{C}\right)^{-1}$.

Example 5 (Running example, quotient). We use the quotient to find the specification of the component that we need to add to the system shown in Figure 1c in order to meet the top level contract $\mathcal{C}$. To compute the quotient, we use (10). We let $\mathcal{C}/\mathcal{C}_c = (\mathcal{E}_q, \mathcal{I}_q)$ and obtain $\mathcal{E}_q = \mathcal{E} \wedge \mathcal{I}_c$ and

$$\begin{aligned}\mathcal{I}_q = \{M \in \mathbb{M} \mid& \exists f \in (2^n \to 2^o) \forall (H, S, P, O_1, O_2, O)\\ & \in M. ((s_1 \to O_1 = f^*(P)) \wedge (s_2 \to O_2 = f^*(P))) \to (H = 0 \to O = f(P))\}.\end{aligned}$$

We can refine the quotient by lifting any restrictions on the environments, and picking from the implementations the term with $f = f^*$. Observe that f^* is a valid choice for f. This yields the hypercontract $\mathcal{C}_3 = (\mathcal{E}_3, \mathcal{I}_3)$, defined as $\mathcal{E}_3 = \mathbb{M}$ and

$$\begin{aligned}\mathcal{I}_3 = &\{M \in \mathbb{M} \mid \forall (H, S, P, O_1, O_2, O) \in M.\\ &\quad ((s_1 \rightarrow O_1{=}f^*(P)) \wedge (s_2 \rightarrow O_2{=}f^*(P))) \rightarrow O{=}f^*(P)\}.\end{aligned}$$

A further refinement of this hypercontract is $\mathcal{C}_r = (\mathcal{E}_r, \mathcal{I}_r)$, where $\mathcal{E}_r = \mathbb{M}$ and

$$\mathcal{I}_r = \{M \in \mathbb{M} \mid \forall (H, S, P, O_1, O_2, O) \in M. ((s_1 \rightarrow O{=}O_1) \wedge (s_2 \rightarrow O{=}O_2))\}.$$

By the properties of the quotient, composing this hypercontract, which knows nothing about f^*, with $\mathcal{C}_c$ will yield a hypercontract which meets the non-interference hypercontract $\mathcal{C}$. Note that this hypercontract is consistent, i.e., it has implementations (in general, refining may lead to inconsistency). $\square$

Merging. The composition of two hypercontracts yields the specification of a system comprised of two design objects, each adhering to one of the hypercontracts being composed. Another important operation on hypercontracts is viewpoint merging, or *merging* for short. It can be the case that the same design element is assigned multiple specifications corresponding to multiple viewpoints, or design concerns [4,23] (e.g., functionality and a performance criterion). Suppose $\mathcal{C}_1 = (\mathcal{E}_1, \mathcal{S}_1)$ and $\mathcal{C}_2 = (\mathcal{E}_2, \mathcal{S}_2)$ are the hypercontracts we wish to merge. Two slightly different operations can be considered as candidates for formalizing viewpoint merging:

– A *weak merge* which is the GLB; and
– A *strong merge* which states that environments of the merger should be environments of both $\mathcal{C}_1$ and $\mathcal{C}_2$ and that the closed systems of the merger are closed systems of both $\mathcal{C}_1$ and $\mathcal{C}_2$. If we let $\mathcal{C}_1 \bullet \mathcal{C}_2 = (\mathcal{E}, \mathcal{I})$, we have

$$\begin{aligned}\mathcal{E} &= \vee \{\mathcal{E}' \in \mathbf{CmpSet} \mid \mathcal{E}' \leq \mathcal{E}_1 \wedge \mathcal{E}_2 \text{ and } \exists \mathcal{C}'' = (\mathcal{E}'', \mathcal{I}'') \in \mathbf{Contr}.\ \mathcal{E}' = \mathcal{E}''\}\\ \mathcal{I} &= \vee \left\{ \mathcal{I}' \in \mathbf{CmpSet} \;\middle|\; \begin{array}{l} \mathcal{I}' \leq (\mathcal{S}_1 \wedge \mathcal{S}_2)/\mathcal{E} \text{ and} \\ (\mathcal{E}, \mathcal{I}) \in \mathbf{Contr} \end{array} \right\}.\end{aligned}$$

The difference is that, whereas the commitment to satisfy $\mathcal{S}_2$ survives under the weak merge when the environment fails to satisfy $\mathcal{E}_1$, no obligation survives under the strong merge. This distinction was proposed in [28] under the name of weak/strong assumptions.

3.4 An example on robustness

Now we explore assume-guarantee specifications of autonomous vehicles. We will deal with their safety and the robustness of their perception components. In order to consider the perception components, we will build our model using a pair (X, O), where $X \in S$ is the input image, belonging to a set S of images, and $O \in CS$ is the classification of the image X, an element of the classification space CS. To deal with safety, we will consider pairs $(v, \Delta s)$, where v represents the state of the vehicle with domain SP, and Δs is the maximum amount of time that it takes the vehicle to come to a full stop. Thus, every component $M \in \mathbb{M}$ is of the form

$$M = \left\{(X, O, v, \Delta_s) \in S \times CS \times SP \times \mathbb{R}^+ \;\middle|\; \exists f \in S \to CS.\ O = f(X)\right\}.$$

As discussed in Seshia et al. [31], certain robustness properties of data-driven components are hyperproperties. Robustness properties usually take the form $d(x, y) < \delta \Rightarrow D(f(x), f(y)) < \varepsilon$, where d and D are distance functions. The property says that points that are close should have similar classifications. As two points are needed to provide evidence that a function is not robust, these are 2-safety hyperproperties. We will state a specification for our vehicles that requires their perception components to be robust. Suppose the input space S is partitioned in sets S_i. We want our vehicle to meet the following top-level specification:

$$\mathcal{C} = \left(\mathbb{M}, \left\{ M \in \mathbb{M} \;\middle|\; \begin{array}{l} \forall (x_k, o_k, v_k, \Delta s_k), (x_l, o_l, v_l, \Delta s_l) \in M. \\ \bigwedge_i x_k, x_l \in S_i \to |o_k - o_l| \leq \varepsilon \end{array} \right\}\right).$$

Suppose our vehicle obeys the specification $\mathcal{C}_a$ given by

$$\mathcal{C}_a = \left(\mathbb{M}, \left\{ M \in \mathbb{M} \;\middle|\; \begin{array}{l} \forall (x_k, o_k, v_k, \Delta s_k), (x_l, o_l, v_l, \Delta s_l) \in M. \\ \bigwedge_i x_k, x_l \in S_i \to |o_k - o_l| \leq \varepsilon_i \end{array} \right\}\right).$$

This specification says that the perception component in each region S_i should have a robustness ε_i. Suppose that there is a $j \in \mathbb{N}$ such that $\varepsilon_i \leq \varepsilon$ for all $i \leq j$ and $\varepsilon_i > \varepsilon$ otherwise. The contract quotient is $\mathcal{C}_q = (\mathcal{E}_q, \mathcal{I}_q)$, where $\mathcal{E}_q = \mathcal{I}_a$ and

$$\mathcal{I}_q = \frac{\left\{ M \in \mathbb{M} \;\middle|\; \begin{array}{l} \forall (x_k, o_k, v_k, \Delta s_k), (x_l, o_l, v_l, \Delta s_l) \in M. \\ \bigwedge_i x_k, x_l \in S_i \to |o_k - o_l| \leq \varepsilon \end{array} \right\}}{\left\{ M \in \mathbb{M} \;\middle|\; \begin{array}{l} \forall (x_k, o_k, v_k, \Delta s_k), (x_l, o_l, v_l, \Delta s_l) \in M. \\ \bigwedge_i x_k, x_l \in S_i \to |o_k - o_l| \leq \varepsilon_i \end{array} \right\}},$$

where we used the horizontal bar to denote the compset quotient. By the definition of the contract quotient, any refinement of $\mathcal{C}_q$ is a solution to our problem, namely, what is the specification that we have to compose with a specification $\mathcal{C}_a$ in order for the result to meet a goal specification $\mathcal{C}$. We thus compute a refinement of the quotient that we just obtained:

$$\mathcal{C}_b = \left(\mathbb{M}, \left\{ M \in \mathbb{M} \;\middle|\; \begin{array}{l} \forall (x_k, o_k, v_k, \Delta s_k), (x_l, o_l, v_l, \Delta s_l) \in M. \\ \bigwedge_{i > j} x_k, x_l \in S_i \to |o_k - o_l| \leq \varepsilon \end{array} \right\}\right).$$

Observe how using the quotient we were able to obtain a specification $\mathcal{C}_b$ that contains exactly what needs to be fixed in the component adhering to hypercontract $\mathcal{C}_a$ in order for it to meet the top-level specification $\mathcal{C}$. Moreover, the specification $\mathcal{C}_b$ does not contain any information about $\mathcal{C}_a$.

One of the uses of hypercontracts is in handling multiple viewpoints. Suppose that the robust perception specification is given to a vehicle on top of other specifications, such as safety. For example, suppose there is a specification that says that if the state of the vehicle v is inside a safety set T, then the amount of time Δs that it takes the vehicle to come to a full stop is a most P. We can write the spec

$$\mathcal{C}_s = (v \in T, \Delta s < P)\,.$$

By using strong merging, we can get into a single top-level hypercontract the specification of the perception and the safety viewpoints, as follows:

$$\left(v \in T, \left\{ M \in \mathbb{M} \;\middle|\; \begin{array}{l} \forall (x_k, o_k, v_k, \Delta s_k), (x_l, o_l, v_l, \Delta s_l) \in M. \\ \Delta s_k, \Delta s_l < P \wedge \bigwedge_i x_k, x_l \in S_i \rightarrow |o_k - o_l| \leq \varepsilon \end{array} \right\} \right).$$

This specification summarizes the perception and safety viewpoints of the vehicle. As robustness is a hyperproperty, we cannot use AG contracts to reason about the specifications in this example, but hypercontracts enable us to do so.

4 Behavioral modeling

In the behavioral approach to system modeling, we start with a set $\mathcal{B}$ whose elements we call behaviors. Components are defined as subsets of $\mathcal{B}$. They contain the behaviors they can display. A component M is a subcomponent of M' if M' contains all the behaviors of M, i.e., if $M \subseteq M'$. Component composition is given by set intersection: $M \times M' \stackrel{\text{def}}{=} M \cap M'$. If we represent the components as $M = \{b \in \mathcal{B} \mid \phi(b)\}$ and $M' = \{b \in \mathcal{B} \mid \phi'(b)\}$ for some constraints ϕ and ϕ', then composition is $M \times M' = \{b \in \mathcal{B} \mid \phi(b) \wedge \phi'(b)\}$, i.e., the behaviors that simultaneously meet the constraints of M and M'. This notion of composition is independent of the connection topology: the topology is inferred from the behaviors of the components.

We will consider two contract theories we can build with these components. The first is based on unconstrained hyperproperties; the second is based on downward-closed hyperproperties.

4.1 General hypercontracts

The most expressive behavioral theory of hypercontracts is obtained when we place no restrictions on the structure of compsets and hypercontracts. In this case, the elements of **CmpSet** are all objects $H \in 2^{2^{\mathcal{B}}}$, i.e., all hyperproperties. The meet and join of compsets are set intersection and union, respectively, and their composition and quotient are given by (3) and (4), respectively. Hypercontracts are of the form $\mathcal{C} = (\mathcal{E}, \mathcal{I})$ with all extrema achieved in the binary operations, i.e., for a second hypercontract $\mathcal{C}' = (\mathcal{E}', \mathcal{I}')$, the meet, join, and composition (5) are, respectively, $\mathcal{C} \wedge \mathcal{C}' = (\mathcal{E} \cup \mathcal{E}', \mathcal{I} \cap \mathcal{I}')$, $\mathcal{C} \vee \mathcal{C}' = (\mathcal{E} \cap \mathcal{E}', \mathcal{I} \cup \mathcal{I}')$, and $\mathcal{C} \parallel \mathcal{C}' = \left(\frac{\mathcal{E}'}{\mathcal{I}} \cap \frac{\mathcal{E}}{\mathcal{I}'}, \mathcal{I} \parallel \mathcal{I}'\right)$. From these follow the operations of quotient, and merging.

4.2 Conic (or downward-closed) hypercontracts

We assume that **CmpSet** contains exclusively downward-closed hyperproperties. Let $H \in$ **CmpSet**. We say that $M \models H$ is a maximal component of H when H contains no set bigger than M, i.e., if $\forall M' \models H.\ M \leq M' \Rightarrow M' = M$.

We let $\overline{H}$ be the set of maximal components of H:

$$\overline{H} = \{M \models H \mid \forall M' \models H.\ M \leq M' \Rightarrow M' = M\}.$$

Due to the fact H is downward-closed, the set of maximal components is a unique representation of H. We can express H as

$$H = \bigcup\nolimits_{M \in \overline{H}} 2^M.$$

We say that H is *k-conic* if the cardinality of $\overline{H}$ is finite and equal to k, and we write this

$$H = \langle M_1, \ldots, M_k \rangle, \text{ where } \overline{H} = \{M_1, \ldots, M_k\}.$$

Order. The notion of order on **CmpSet** can be expressed using this notation as follows: suppose $H' = \langle M' \rangle_{M' \in \overline{H}'}$. Then

$$H' \leq H \text{ if and only if } \forall M' \in \overline{H}'\ \exists M \in \overline{H}.\ M' \leq M.$$

Composition. Composition in **CmpSet** becomes

$$H \times H' = \bigcup_{\substack{M \in \overline{H} \\ M' \in \overline{H}'}} 2^{M \cap M'} = \langle M \cap M' \rangle_{\substack{M \in \overline{H} \\ M' \in \overline{H}'}}. \tag{7}$$

Therefore, if H and H' are, respectively, k- and k'-conic, $H \times H'$ is at most kk'-conic.

Quotient. Suppose H_q satisfies

$$H' \times H_q \leq H.$$

Let $M_q \in \overline{H}_q$. We must have

$$M_q \times M' \models H \text{ for every } M' \in \overline{H}',$$

which means that for each $M' \in \overline{H}'$ there must exist an $M \in \overline{H}$ such that $M_q \times M' \leq M$; let us denote by $M(M')$ a choice $M' \mapsto M$ satisfying this condition. Therefore, we have

$$M_q \leq \bigwedge\nolimits_{M' \in \overline{H}'} \frac{M(M')}{M'}, \tag{8}$$

Clearly, the largest such M_q is obtained by making (8) an equality. Thus, the cardinality of the quotient is bounded from above by $k^{k'}$ since we have

$$H_q = \left\langle \bigwedge_{M' \in \overline{H}'} \frac{M(M')}{M'} \right\rangle_{\substack{M(M') \in \overline{H} \\ \forall M' \in \overline{H}'}} . \tag{9}$$

Contracts. Now we assume that the objects of **CmpSet** are pairs of *downward-closed compsets.* If we have two hypercontracts $\mathcal{C} = (\mathcal{E}, \mathcal{I})$ and $\mathcal{C}' = (\mathcal{E}', \mathcal{I}')$, their composition and quotient are, respectively,

$$\mathcal{C} \parallel \mathcal{C}' = \left(\frac{\mathcal{E}}{\mathcal{I}'} \wedge \frac{\mathcal{E}'}{\mathcal{I}}, \mathcal{I} \times \mathcal{I}' \right) \text{ and } \mathcal{C}/\mathcal{C}' = \left(\mathcal{E} \times \mathcal{I}', \frac{\mathcal{I}}{\mathcal{I}'} \wedge \frac{\mathcal{E}'}{\mathcal{E}} \right). \tag{10}$$

5 Conclusions

We proposed hypercontracts, a generic model of contracts providing a richer algebra than the metatheory of [5]. We started from a generic model of components equipped with a simulation preorder and parallel composition. On top of them, we considered compsets (or hyperproperties, for behavioral formalisms), which are lattices of sets of components equipped with parallel composition and quotient; compsets are our generic model formalizing "properties." Hypercontracts are then defined as pairs of compsets specifying the allowed environments and either the obligations of the closed system or the set of allowed implementations—both forms are useful.

We specialized hypercontracts by restricting them to conic hypercontracts, whose environments and closed systems are described by a finite number of components. Conic hypercontracts include assume-guarantee contracts as a specialization. We illustrated the versatility of our model on the definition of contracts for information flow in security and robustness of data-driven components.

The flexibility and power of our model suggests that a number of directions that were opened in [5], but not explored to their end, can now be re-investigated with more powerful tools: contracts and testing, subcontract synthesis (for requirement engineering), contracts and abstract interpretation, contracts in physical system modeling.[3]

Furthermore, results on contracts were also recently obtained in the domain of control systems. In particular, Phan-Minh and Murray [25,24] introduced the notion of reactive contracts. Saoud et al. [29,30] proposed a framework of assume-guarantee contracts for input/output discrete or continuous time systems. Assumptions vs. guarantees are properties stated on inputs vs. outputs; with this restriction, reactive contracts are considered and an elegant formula is proposed for the parallel composition of contracts. This motivates us to establish further links.

[3] Simulink and Modelica toolsuites propose requirements toolboxes, in which requirements are physical system properties that can be tested on a given system model, thus providing a limited form of contract. This motivates the development of a richer contract framework helping for requirement engineering in Cyber-Physical Systems design.

References

1. M. Abadi and L. Lamport. Composing specifications. *ACM Trans. Program. Lang. Syst.*, 15(1):73–132, Jan. 1993.
2. S. S. Bauer, A. David, R. Hennicker, K. G. Larsen, A. Legay, U. Nyman, and A. Wąsowski. Moving from specifications to contracts in component-based design. In J. de Lara and A. Zisman, editors, *Fundamental Approaches to Software Engineering*, pages 43–58, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
3. S. S. Bauer, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. A modal specification theory for components with data. *Sci. Comput. Program.*, 83:106–128, 2014.
4. A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis. Multiple viewpoint contract-based specification and design. In F. S. de Boer, M. M. Bonsangue, S. Graf, and W.-P. de Roever, editors, *Formal Methods for Components and Objects: 6th International Symposium, FMCO 2007, Amsterdam, The Netherlands, October 24-26, 2007, Revised Lectures*, pages 200–225, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
5. 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*, 12(2-3):124–400, 2018.
6. F. Bujtor and W. Vogler. Error-pruning in interface automata. In *40th International Conference on Current Trends in Theory and Practice of Computer Science*, SOFSEM 2014, pages 162–173, Nový Smokovec, Slovakia, January 26-29, 2014.
7. M. R. Clarkson and F. B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.
8. J. W. Coleman and C. B. Jones. A structural proof of the soundness of rely/guarantee rules. *J. Log. Comput.*, 17(4):807–841, 2007.
9. 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, page 109–120, New York, NY, USA, 2001. Association for Computing Machinery.
10. L. de Alfaro and T. A. Henzinger. Interface theories for component-based design. In T. A. Henzinger and C. M. Kirsch, editors, *EMSOFT*, volume 2211 of *Lecture Notes in Computer Science*, pages 148–165. Springer, 2001.
11. L. Doyen, T. A. Henzinger, B. Jobstmann, and T. Petrov. Interface theories with component reuse. In *Proceedings of the 8th ACM & IEEE International conference on Embedded software, EMSOFT'08*, pages 79–88, Atlanta, GA, 2008.
12. J. A. Goguen and J. Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, April 26-28, 1982*, pages 11–20, Oakland, CA, USA, 1982. IEEE Computer Society.
13. I. J. Hayes and C. B. Jones. A guide to rely/guarantee thinking. In J. P. Bowen, Z. Liu, and Z. Zhang, editors, *Engineering Trustworthy Software Systems - Third International School, SETSS 2017, Chongqing, China, April 17-22, 2017, Tutorial Lectures*, volume 11174 of *Lecture Notes in Computer Science*, pages 1–38. Springer, 2017.
14. I. Incer, L. Mangeruca, T. Villa, and A. L. Sangiovanni-Vincentelli. The quotient in preorder theories. In J.-F. Raskin and D. Bresolin, editors, Proceedings 11th International Symposium on *Games, Automata, Logics, and Formal Verification,* Brussels, Belgium, September 21-22, 2020, volume 326 of *Electronic Proceedings*

in Theoretical Computer Science, pages 216–233, Brussels, Belgium, 2020. Open Publishing Association.

15. C. B. Jones. Specification and design of (parallel) programs. In *IFIP Congress*, pages 321–332, Paris, France, 1983.
16. C. B. Jones. Wanted: a compositional approach to concurrency. In A. McIver and C. Morgan, editors, *Programming Methodology*, pages 5–15, New York, NY, 2003. Springer New York.
17. K. G. Larsen, U. Nyman, and A. Wasowski. Interface Input/Output Automata. In *FM*, pages 82–97, 2006.
18. K. G. Larsen, U. Nyman, and A. Wasowski. Modal I/O Automata for Interface and Product Line Theories. In *Programming Languages and Systems, 16th European Symposium on Programming, ESOP'07*, volume 4421 of *Lecture Notes in Computer Science*, pages 64–79. Springer, 2007.
19. K. G. Larsen, U. Nyman, and A. Wasowski. On Modal Refinement and Consistency. In *Proc. of the 18th International Conference on Concurrency Theory (CONCUR'07)*, pages 105–119. Springer, 2007.
20. G. Lüttgen and W. Vogler. Modal interface automata. *Logical Methods in Computer Science*, 9(3), 2013.
21. I. Mastroeni and M. Pasqua. Verifying bounded subset-closed hyperproperties. In A. Podelski, editor, *Static Analysis*, pages 263–283, Cham, 2018. Springer International Publishing.
22. R. Negulescu. Process spaces. In C. Palamidessi, editor, *CONCUR 2000 — Concurrency Theory*, pages 199–213, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
23. R. Passerone, I. Incer, and A. L. Sangiovanni-Vincentelli. Coherent extension, composition, and merging operators in contract models for system design. *ACM Trans. Embed. Comput. Syst.*, 18(5s), Oct. 2019.
24. T. Phan-Minh. *Contract-Based Design: Theories and Applications*. PhD thesis, California Institute of Technology, 2021.
25. T. Phan-Minh and R. M. Murray. Contracts of reactivity. Technical report, California Institute of Technology, 2019.
26. M. N. Rabe. *A temporal logic approach to information-flow control*. PhD thesis, Universität des Saarlandes, 2016.
27. J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay, and R. Passerone. Modal interfaces: Unifying interface automata and modal specifications. In *Proceedings of the Seventh ACM International Conference on Embedded Software*, EMSOFT '09, page 87–96, New York, NY, USA, 2009. Association for Computing Machinery.
28. A. L. Sangiovanni-Vincentelli, W. Damm, and R. Passerone. Taming Dr. frankenstein: Contract-based design for cyber-physical systems. *Eur. J. Control*, 18(3):217–238, 2012.
29. A. Saoud, A. Girard, and L. Fribourg. On the composition of discrete and continuous-time assume-guarantee contracts for invariance. In *16th European Control Conference, ECC, June 12-15, 2018*, pages 435–440, Limassol, Cyprus, 2018. IEEE.
30. A. Saoud, A. Girard, and L. Fribourg. Assume-guarantee contracts for continuous-time systems. working paper or preprint, Feb. 2021.
31. S. A. Seshia, A. Desai, T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, S. Shivakumar, M. Vazquez-Chanlatte, and X. Yue. Formal specification for deep neural networks. In S. K. Lahiri and C. Wang, editors, *Automated Technology for Verification and Analysis*, pages 20–34, Cham, 2018. Springer International Publishing.

A Supplementary material: Proofs

Proof (Proposition 1).

$$\begin{aligned}
\mathcal{C}''/\mathcal{C} &= \bigvee \{\mathcal{C}' \mid \mathcal{C} \parallel \mathcal{C}' \le \mathcal{C}''\} = \bigvee \left\{ (\mathcal{E}', \mathcal{I}') \;\middle|\; \begin{bmatrix} \mathcal{I} \parallel \mathcal{I}' \le \mathcal{I}'', \\ \mathcal{E}'' \parallel \mathcal{I} \le \mathcal{E}', \text{ and} \\ \mathcal{E}'' \parallel \mathcal{I}' \le \mathcal{E} \end{bmatrix} \right\} \\
&= \left(\left(\bigvee \left\{ (\mathcal{E}', \mathcal{I}') \;\middle|\; \begin{bmatrix} \mathcal{I} \parallel \mathcal{I}' \le \mathcal{I}'', \\ \mathcal{E}'' \parallel \mathcal{I} \le \mathcal{E}', \text{ and} \\ \mathcal{E}'' \parallel \mathcal{I}' \le \mathcal{E} \end{bmatrix} \right\} \right)^{-1} \right)^{-1} \\
&\overset{(6)}{=} \left(\bigwedge \left\{ (\mathcal{I}', \mathcal{E}') \;\middle|\; \begin{bmatrix} \mathcal{I} \parallel \mathcal{I}' \le \mathcal{I}'', \\ \mathcal{E}'' \parallel \mathcal{I} \le \mathcal{E}', \text{ and} \\ \mathcal{E}'' \parallel \mathcal{I}' \le \mathcal{E} \end{bmatrix} \right\} \right)^{-1} \\
&= \left(\bigwedge \left\{ (\mathcal{I}', \mathcal{E}') \;\middle|\; \begin{bmatrix} \mathcal{E}'' \parallel \mathcal{I} \le \mathcal{E}', \\ \mathcal{I}' \parallel \mathcal{I} \le \mathcal{I}'', \text{ and} \\ \mathcal{I}' \parallel \mathcal{E}'' \le \mathcal{E} \end{bmatrix} \right\} \right)^{-1} \\
&= \left((\mathcal{C}'')^{-1} \parallel \mathcal{C} \right)^{-1}.
\end{aligned}$$