Are Interface Theories Equivalent to Contract Theories?

Pierluigi Nuzzo*, Antonio Iannapollo*, Stavros Tripakis†, Alberto Sangiovanni-Vincentelli*
* EECS Department, University of California at Berkeley, Berkeley, CA 94720
† Department of Information and Computer Science, Aalto University, Finland
Email: {nuzzo, antonio, stavros, alberto}@eecs.berkeley.edu

Abstract—Contract-based design is emerging as a unifying compositional paradigm for the specification, design and verification of large-scale complex systems. Different contract frameworks are currently available, but we lack a clear understanding of the relations between them. In this paper, we investigate the relation between interface theories (specifically, relational interfaces) and assume-guarantee (A/G) contracts. We introduce a natural transformation of interfaces to A/G contracts represented by linear temporal logic. Then, we analyze differences and correspondences between key operators and relations in the two theories (i.e., composition, refinement and conjunction), by studying their preservation properties under the proposed transformation. We show that the transformation preserves refinement, but does not generally preserve serial composition and conjunction. Then, we present an assumption-projection operator to make it possible to preserve serial composition and compatibility checking. Finally, we provide illustrative examples that shed light on the effectiveness of both frameworks for requirement formalization, early detection of integration errors, and use of abstraction-refinement.

I. INTRODUCTION

Designing large and complex embedded and cyber-physical systems (such as “smart” buildings, “smart” transportation, energy, security, and health-care systems) cannot be done in a monolithic manner. Instead, designers naturally use compositional methods, which allow to assemble a large and complex system from smaller and simpler components (e.g., pre-defined library blocks or subsystems). Methodologies such as component-based design [1] and contract-based design [2] (CBD) are emerging as unifying formal compositional paradigms. They support requirement engineering by providing rigorous formalisms to capture the correct transition between different abstraction levels in system design. Moreover, they offer mechanisms for early detection of integration errors, e.g., by checking compatibility between the components locally, before performing global system verification.

Yet, different formal theories of components and contracts have been proposed in the literature, and there is currently no clear understanding of the relations between them. This paper aims to fill this gap.

We focus in particular on the relation between the so-called interface theories [1], such as interface automata [3] and relational interfaces [4], on the one hand, and the assume-guarantee (A/G) contract framework [5], [6], on the other hand. Examining the relation between these two frameworks is interesting because, while having the same overall objectives, they are supported by quite different mathematical theories. For instance, in an A/G contract the assumptions made on the environment and the guarantees provided by the system are modeled as separate sets of behaviors, whereas in interface theories the two are “merged” into a single model, called an interface.

In addition, interfaces generally rely on the distinction between inputs and outputs. The fact that an interface may not be input-complete (i.e., accept any input at any time) is essential and leads to game-theoretic definitions of composition and refinement. On the other hand, A/G contracts capture assumptions and guarantees as sets of behaviors over a common set of variables, in general with no distinction between inputs and outputs (e.g., for composition).

These differences result in different definitions of key elements of the theories, such as composition and refinement. This paper aims to shed light on the subtle differences between the two frameworks. To be concrete, we start from the theory of synchronous relational interfaces [4]. We choose stateless relational interfaces rather than other, more general interface theories, such as interface automata, as the former are simpler and can offer more intuitive support to our investigation. We provide an operator which transforms a relational interface into an A/G contract, in the natural way. In particular, a relational interface represented as a formula φ on inputs and outputs is mapped into a set of behaviors representing the safety property that φ holds at every (synchronous) step. This can be concretely represented by the LTL formula □φ.

We then highlight differences and correspondences between key operators and relations in the two theories by studying their preservation properties under the above transformation. We show that, perhaps surprisingly, the basic operation of serial composition of interfaces is not preserved. Specifically, composing two interfaces I1 and I2, and then transforming the result to an A/G contract, is not equivalent to first transforming each of I1 and I2 to an A/G contract, and then composing the contracts. The reason for this is that the interface compatibility check is “built into” the interface composition operator, so that if the interfaces are incompatible, the result of the composition is False. On the other hand, A/G contracts have no way of checking compatibility a-priori during composition. Although compatibility can be checked a-posteriori on the composite contract using the notion of -receptiveness [5], the latter provides a yes/no answer and does not infer new environment assumptions, as in the case of interface composition.

To remedy this, we introduce an assumption-projection operator for A/G contracts. The latter eliminates (“hides”) a given set of variables (only) from the assumption, using universal (i.e., game-theoretic) rather than the usual existential quantification. We show that with this hiding operator the transformation preserves the semantics of interface composition. Unfortunately, LTL formulas are not generally closed under variable elimination (projection). It is, therefore, neces-
sary to resort to a strictly more expressive extension of LTL, such as Quantified Linear Temporal Logic (QLTL) [7], [8], to implement this hiding operator. The satisfiability problem for QLTL has been shown to be decidable, but with non-elementary complexity [7].

We also show that our transformation preserves refinement, that is, interface refinement between interfaces \( I_1 \) and \( I_2 \) is equivalent to A/G contract refinement between the corresponding A/G contracts. However, another interesting operator, that of conjunction (also called shared refinement [4]) is not preserved. The reason is another crucial difference between the two frameworks. While A/G contracts reason about global behaviors of components, possibly spanning infinite sequences of reactions, relational interfaces can also capture punctual relations between the inputs and outputs of a component, at the granularity of a single reaction index. Therefore, computation of conjunction as the greatest lower bound (GLB) with respect to the refinement order, generates a smaller set of allowed environments and a larger set of guaranteed behaviors for A/G contracts, which translates into a tighter, less conservative, bound. As a result, the contract associated with the conjunction of interfaces \( I_1 \) and \( I_2 \) refines, but is generally different than, the conjunction of the contracts associated with \( I_1 \) and \( I_2 \).

The rest of the paper is organized as follows. After a brief overview of related works in Section II, we provide a few motivating examples in Section III. Then, we briefly summarize relational interfaces and A/G contracts in Section IV. In Section V, we present the main results of the paper together with several illustrative examples. Finally, in Section VI, we draw some conclusions.

II. RELATED WORK

Despite the proliferation of work on compositional theories in general, and interface and contract theories in particular, there is little work that attempts at drawing links between the existing frameworks. Benveniste et al. [6] propose a general “meta-theory” of contracts, expressed in terms of sets of implementations and environments, and from which both interface theories and A/G contracts can be instantiated. Following a similar approach, Bauer et al. [9] attempt at providing an abstract formalization of the notion of contracts by relating “specification theories” to “contract theories”. In this paper, instead of recurring to a common, more abstract, meta-theory, we aim to directly map interfaces to A/G contracts and, as a result, reveal some of the subtle differences in the two frameworks.

In particular, we start by mapping synchronous relational interfaces to A/G contracts. Relational interfaces have been proposed as an interface theory for synchronous systems that can capture functional relations between the inputs and the outputs of a component [4]. Input/output relations are expressed as first-order logic formulas over the input and output variables. The developed theory supports two types of composition, serial connection and feedback, as well as refinement, compatibility and conjunction (denoted as shared refinement). On the other hand, the A/G contract theory has been designed as a generic mathematical framework encompassing different specification formalisms, and supporting a rich composition algebra, to reason about hierarchies of components and multiple viewpoints [5], [6]. Several aspects of A/G contracts have been the focus of many publications over the last decade, including their application in different domains, such as automotive [6], consumer electronics [10] and, more recently, synthesis and verification of control protocols for cyber-physical systems [11], [12].

Another theory of A/G contracts has also been proposed, which supports rich component interactions by replacing the notion of parallel composition with the one of circular reasoning [13]. However, compatibility and conjunction are not addressed in this framework. Finally, Doyen et al. [14] propose an interface model similar to relational interfaces, except that assumptions on input variables and guarantees on output variables are separated in two different formulas. This type of “assume-guarantee interfaces” are a strict subclass of relational interfaces, since the latter can model relations between input and output variables, which cannot be captured in the former.

III. MOTIVATING EXAMPLES

A/G contracts specify components in terms of sets of behaviors which assign a history of “values” to their variables or ports. Behaviors are generic and abstract; for instance, they could be continuous functions that result from solving differential equations, or sequences of values or events recognized by an automata model [2]. Such a behavioral framework is expressive and versatile enough to encompass all kinds of models encountered in system design, from hardware and software models to representations of physical phenomena. The particular structure of the behaviors is defined by specific instances of the contract model. This will only affect the way operators in the contract algebra are implemented, since the basic definitions will not vary.

A framework centered around behaviors over variables, without a-priori distinction between inputs and outputs, is certainly suitable to model the majority of physical (e.g. mechanical, electrical, hydraulic or thermal) components, which are generally governed by laws that merely impose relations (rather than functions) among system variables, and where interconnections mean that variables are shared (rather than assigned) among subsystems. However, there are some important situations, in which a signal flow approach is more appropriate, e.g. in signal processing, feedback control based on sensor outputs and actuator inputs, and in systems composed of unilateral devices [15]. In these cases, relations between system variables are better viewed in terms of inputs and outputs, and interconnections in terms of output-to-input assignments. Inputs are used to capture the influence of the environment on the system, while outputs are used to capture the influence of the system on the environment. When developing a concrete instance of a specification theory, it is therefore beneficial to support both the behavioral and signal-flow approaches. This is further illustrated by the following motivating examples.

Example 1 (Detecting Incompatibility). We would like to analyze a system model built by interconnecting two blocks in a modeling environment such as SIMULINK, as shown in Fig. 1. L is a legacy block, seen as a black box, on the behaviors of which we have no information. Div produces a real output \( z \) which is the inverse of its real input \( y \). The goal for the overall system would be to provide the inverse of any “legal” output of L.
To specify the behavior of Div, we can use a simple (static) contract $C_D$, informally expressed as follows, in terms of assumptions on the environment and guarantees of the component:

$$\text{variables: } y, z \in \mathbb{R}$$
$$\text{assumptions: } y \neq 0$$
$$\text{guarantees: } z = 1/y$$

where behaviors are specified using constraints on real numbers. On the other hand, a contract $C_{L_1}$ for L would allow any real value both as an input and as an output.

In such a situation, we would like to conclude that the two specifications (contracts) for L and Div are “incompatible”, since there is no way to guarantee that the output of L is always a “legal” input for Div, i.e. is different than zero. While such a concept of incompatibility could be directly detected while computing the composition of the specifications within an interface framework, this is not necessarily the case in the generic A/G contract framework. In fact, using the formulas reported in Section IV-B, it is possible to compute the composition of $C_{L_1}$ and $C_D$ as follows:

$$\text{variables: } x, y, z \in \mathbb{R}$$
$$\text{assumptions: } y \neq 0$$
$$\text{guarantees: } (z = 1/y) \lor (y = 0).$$

The contract in (2) seems to suggest that any environment capable of enforcing $y \neq 0$ would be legal for $C_{L_1} \otimes C_D$. Yet, $y$ has now become an “internal” variable for $C_{L_1} \otimes C_D$, and cannot be modified by any environment of the composite contract.

Example 2 (Inferring Environment Assumptions). We consider again the system in Fig. 1. However, we assume now that a more detailed specification is available for the behaviors of L, stating that L accepts any real input $x$ and provides an output $y > x$. Such a specification can be expressed using the contract:

$$\text{variables: } x, y \in \mathbb{R}$$
$$\text{assumptions: } (x, y) \in \mathbb{R}^2$$
$$\text{guarantees: } y > x.$$
where
\[ \phi = \phi_1 \land \phi_2 \land \forall Y_1 : (\phi_1 \rightarrow in(\phi_2)). \]

I_1 and I_2 are said to be compatible interfaces if \( \phi \) is satisfiable, i.e., if \( \phi \) is not equivalent to False. Interface compatibility can then be checked while computing the serial composition.

**Refinement:** Given two interfaces \( I_1 = (X_1, Y_1, \phi_1) \) and \( I_2 = (X_2, Y_2, \phi_2) \), we say that \( I_1 \) refines \( I_2 \), written \( I_1 \sqsubseteq I_2 \), iff \( X_1 \subseteq X_2, Y_1 \subseteq Y_2 \) and the following formula is valid (i.e., true under all valuations):
\[ in(\phi_2) \rightarrow (in(\phi_1) \land (\phi_1 \rightarrow \phi_2)). \]

**Shared refinement:** Two interfaces \( I_1 = (X, Y, \phi_1) \) and \( I_2 = (X, Y, \phi_2) \) are said to be shared-refinable if the following formula is true:
\[ \forall X : \left( (in(\phi_1) \land in(\phi_2)) \rightarrow (\exists Y : (\phi_1 \land \phi_2)) \right). \]

This amounts to state that for every input that is legal in both \( I_1 \) and \( I_2 \), the corresponding sets of outputs of \( I_1 \) and \( I_2 \) must have a non-empty intersection. If \( I_1 \) and \( I_2 \) are shared-refinable, their shared refinement, denoted \( I_1 \cap I_2 \), is defined to be the interface \( I_1 \cap I_2 := (X, Y, \phi_1) \), where \( \phi_1 = (in(\phi_1) \lor in(\phi_2)) \land (in(\phi_1) \land \phi_1) \land (\phi_2). \)

It can be shown that \( I_1 \cap I_2 \), when it exists, acts as the GLB for the refinement relation, i.e. (i) \( I_1 \cap I_2 \) is guaranteed to refine both \( I_1 \) and \( I_2 \), and (ii) for any interface \( I' \) such that \( I' \sqsubseteq I_1 \) and \( I' \sqsubseteq I_2 \), we have \( I' \sqsubseteq (I_1 \cap I_2) \). The existence of such a GLB is an important feature for hierarchical component-based design supporting multiple viewpoints [5], [14], [16].

**B. Assume/Guarantee Contracts**

Following the formulation of Benveniste et al. [5], [6], an assume-guarantee (A/G) contract is a pair \((A, G)\) where \( A \) and \( G \) are sets of behaviors. \( A \) represents the assumptions that a system makes on its environment, and \( G \) represents the guarantees provided by the system under the environment assumptions. As mentioned earlier, the A/G contract framework is abstract in the sense that it does not predefine the type of behaviors. Behaviors can be of different kinds (e.g., discrete or continuous, finite or infinite in length) and they can be concretely represented using different formalisms, e.g., automata, temporal logic, differential equations. For the purposes of this paper, we consider a specific type of behaviors, in order to establish our results. We therefore equip a contract with a finite set of variables \( V \). A behavior over \( V \) is an infinite sequence of valuations over \( V \), \( \rho = v_0v_1v_2 \cdots \). In the sequel, an A/G contract will be a triple \((V, A, G)\) where \( A \) and \( G \) are sets of behaviors over \( V \).

Often contracts are assumed to be in saturated (canonical) form, meaning that they satisfy \( A \subseteq G \), where \( A \) is the complement of \( A \). In the sequel we assume that contracts are given in saturated form. This is not a restrictive assumption as we can always transform a contract \((V, A, G)\) into its saturated form \((V, A, G')\) where \( G' = G \cup \overline{A} \).

**Satisfaction:** A contract is to be realized by an implementation, modeled as a set of behaviors \( M \) over the same set of variables. A set of behaviors \( M \) over \( V \) satisfies a contract \( C = (V, A, G) \), written \( M \models C \), when it satisfies its guarantee subject to the assumption; formally, \( M \cap A \subseteq G \). Similarly, a contract admits a set of legal environments, each modeled as a set of behaviors \( E \) over the same set of variables. A set of behaviors \( E \) over \( V \) satisfies a contract \( C = (V, A, G) \) as an environment, written \( E \models E C \), when it satisfies its assumptions; formally, \( E \subseteq A \).

**Composition:** Parallel composition of contracts can be used to construct composite contracts out of simpler ones. Let \( C_1 = (V, A_1, G_1) \) and \( C_2 = (V, A_2, G_2) \) be contracts (in saturated form) over the same set of variables \( V \). The composite contract \( C_1 \otimes C_2 \) is defined as the triple \((V, A, G)\) where [6]:
\[
A = (A_1 \cap A_2) \cup (G_1 \cap G_2) \quad (4)
\]
\[
G = G_1 \cap G_2. \quad (5)
\]

Note that contract composition preserves saturated form, that is, if \( C_1 \) and \( C_2 \) are in saturated form, then so is \( C_1 \otimes C_2 \). Moreover, \( \otimes \) is associative and commutative and generalizes to an arbitrary number of contracts. We therefore can write \( C_1 \otimes C_2 \otimes \cdots \otimes C_n \).

In order for composition to be defined, contracts need to be over the same set of variables \( V \). If this is not the case, then, before composing the contracts, we must first extend their behaviors to a common set of variables using an inverse projection type of transformation. We call this process alphabet equalization. Formally, let \( C = (V, A, G) \) be a contract and let \( V' \supseteq V \) be the set of variables on which we want to extend \( C \). The extension of \( C \) on \( V' \) is the new contract \( C' = (V', A', G') \) where \( A' \) and \( G' \) are sets of behaviors over \( V' \), defined by inverse projection of \( A \) and \( G \), respectively. In the sequel, we freely compose contracts \( C_1 = (V_1, A_1, G_1) \) and \( C_2 = (V_2, A_2, G_2) \) over arbitrary sets of variables \( V_1, V_2 \), by implicitly first taking their extensions to \( V = V_1 \cup V_2 \).

**Compatibility:** A saturated contract \( C = (V, A, G) \) is called compatible if there exists a legal (non-empty) environment \( E \) for \( C \), i.e., if and only if \( A \neq \emptyset \). This definition can then be lifted to pairs of contracts, so that two contracts \( C_1 \) and \( C_2 \) are compatible iff \( C_1 \otimes C_2 \) is compatible.

Some works present versions of the A/G contract theory which distinguish between input (uncontrolled) and output (controlled) variables [2], [5]. The definition of contract composition is not changed in that case, but a new notion of contract compatibility can be defined. Let \( c \subseteq V \) be the subset of controlled variables of \( C \). Then \( C \) is compatible iff \( A \) is \( c \)-receptive, i.e., iff for all behaviors \( \rho' \) restricted to variables in \( c \), there exists a behavior \( \rho \in A \), such that \( \rho' \) and \( \rho \) coincide over \( c \). Intuitively, an environment has no control on the variables set by an implementation, and therefore \( A \) accepts any history offered to the subset \( c \) of its variables.

**Consistency:** A saturated contract \( C = (V, A, G) \) is called consistent if there exists a non-empty implementation \( M \) for \( C \), i.e., if and only if \( G \neq \emptyset \). As with compatibility, consistency can also be lifted to pairs of contracts, so that \( C_1 \) and \( C_2 \) are consistent iff \( C_1 \otimes C_2 \) is consistent. Similarly, A/G contract versions that distinguish between controlled and uncontrolled variables define consistency based on the notion of receptiveness. In these versions, if \( u \subseteq V \) is the subset
of uncontrolled variables of \( C \), then \( C \) is consistent if \( G \) is \( u \)-receptive.

**Refinement:** We say that contract \( C_1 = (V, A_1, G_1) \) refines contract \( C_2 = (V, A_2, G_2) \) (with \( C_1 \) and \( C_2 \) both in saturated form), written \( C_1 \leq C_2 \), if and only if \( A_1 \supseteq A_2 \) and \( G_1 \subseteq G_2 \). Refinement amounts to relaxing assumptions and reinforcing guarantees, therefore strengthening the contract. The refinement relation defines a preorder on contracts. Clearly, if \( M \models C_1 \) and \( C_1 \leq C_2 \), then \( M \models C_2 \). On the other hand, if \( E \models C_2 \), then \( E \models C_1 \). In other words, contract \( C_1 \) refines another contract \( C_2 \), if \( C_1 \) admits less implementations than \( C_2 \), but more legal environments than \( C_2 \). This is a standard concept inspired by the notion of behavioral subtyping [9].

**Conjunction:** The conjunction of two contracts \( C_1 = (V, A_1, G_1) \) and \( C_2 = (V, A_2, G_2) \) is defined to be the contract \( C_1 \land C_2 = (V, A_{1 \cup A_2}, G_1 \cap G_2) \). Conjunction of A/G contracts has the same properties of shared refinement in relational interfaces. Note, however, that shared refinement of interfaces is not always defined, whereas conjunction of A/G contracts is always defined as the GLB for the refinement preorder, which is guaranteed to exist.

**C. LTL A/G Contracts**

To work with A/G contracts, we may concretely express the sets of behaviors \( A \) and \( G \) as formulas in *linear temporal logic* (LTL) [17], a widespread formalism to reason about reactive systems and perform analysis and synthesis of embedded control software [18], [19]. Contracts expressed as temporal logic formulas have been used in the literature to instantiate a concrete contract framework and proof system for compositional system verification [20], and to demonstrate a scalable refinement checking algorithm [12]. In what follows, we briefly recall how the main A/G contract operators can be mapped onto entailment of temporal logic formulas.

An LTL A/G contract can be seen as a triple \((V, \varphi_a, \varphi_g)\), where \( \varphi_a \) and \( \varphi_g \) are LTL formulas over the set of variables \( V \). For instance, if \( V = \{x, y\} \) and \( x, y \) are both integer variables, a possible LTL A/G contract is \((V, \square x \geq 0, y \geq 0)\). An LTL formula represents a set of behaviors. For example, the formula \( \square x \geq 0 \) represents the set of all behaviors where \( x \) is never negative.

Most operations on contracts can then be implemented as operations on LTL formulas in a straightforward way. Satisfiability of \((V, \varphi_a, \varphi_g)\) can be achieved by setting \( \varphi_g := \varphi_a \Rightarrow \varphi_g \). The parallel composition of contracts \( C_1 = (V, \varphi_{a_1}, \varphi_{g_1}) \) and \( C_2 = (V, \varphi_{a_2}, \varphi_{g_2}) \) can be directly defined in terms of LTL formulas as

\[
C_1 \otimes C_2 = (V, (\varphi_{a_1} \land \varphi_{a_2}) \lor (\lnot (\varphi_{g_1} \land \varphi_{g_2}) \lor \varphi_{g_1} \land \varphi_{g_2})).
\]

Refinement is instead a preorder on contracts, which formalizes a notion of substitutability. We say that contract \( C_1 = (V, \varphi_{a_1}, \varphi_{g_1}) \) refines contract \( C_2 = (V, \varphi_{a_2}, \varphi_{g_2}) \), written \( C_1 \leq C_2 \), if formulas \( \varphi_{a_2} \rightarrow \varphi_{a_1} \) and \( \varphi_{g_1} \rightarrow \varphi_{g_2} \) are both valid, or equivalently, if \( \lnot (\varphi_{a_2} \rightarrow \varphi_{a_1}) \) and \( \lnot (\varphi_{g_1} \rightarrow \varphi_{g_2}) \) are both unsatisfiable. Similarly, compatibility and consistency checking reduce to LTL satisfiability problems. Finally, the conjunction of two contracts \( C_1 = (V, \varphi_{a_1}, \varphi_{g_1}) \) and \( C_2 = (V, \varphi_{a_2}, \varphi_{g_2}) \) can be obtained as

\[
C_1 \land C_2 = (V, \varphi_{a_1} \lor \varphi_{a_2}, \varphi_{g_1} \land \varphi_{g_2}).
\]

---

**Fig. 2.** Pictorial representation of the relational interfaces in Example 3 (a) and Example 4 (b).

**V. MAPPING RELATIONAL INTERFACES INTO ASSUME-GUARANTEE CONTRACTS**

We map interfaces into contracts based on the following definition.

**Definition V.1 (Contract Associated with an Interface).** An interface \( I = (X, Y, \phi) \) can be transformed into a contract

\[
C = \mathcal{F}(I) = (V, A, G)
\]

where

\[
V := X \cup Y, \quad A := \square \text{in}(\phi), \quad G := \square \text{in}(\phi) \rightarrow \square \phi.
\]

We call \( C \) the contract associated with \( I \) under the transformation \( \mathcal{F} \).

Even though \( \text{in}(\phi) \) is a formula over only the set of input variables \( X \), when we define \( A \) we choose to interpret \( \text{in}(\phi) \) over the entire set of variables \( V = X \cup Y \). In fact, both \( A \) and \( G \) in a contract are defined as behaviors over the same set of variables. Moreover, we conveniently express the sets of behaviors in \( A \) and \( G \) as LTL formulas, where \( \square \phi \) denotes the set of behaviors \([\phi] \). A relational interface represented as a formula \( \phi \) on inputs and outputs is then mapped into sets of behaviors representing the safety property that \( \phi \) holds at every (synchronous) step, under the assumption that \( \text{in}(\phi) \) holds at every step.

The contract \( \mathcal{F}(I) \) in Definition V.1 preserves the semantics of the associated interface \( I \). Any legal environment of \( \mathcal{F}(I) \) provides, at every step, a legal input for \( I \) (while accepting any output from \( I \)), and vice versa. On the other hand, any system satisfying \( \phi \) at every step is an implementation for \( \mathcal{F}(I) \), since it satisfies \( \square \phi \) in any context in which \( \square \text{in}(\phi) \) holds. Finally, by definition, contract \( \mathcal{F}(I) \) is in saturated form. In what follows, we analyze the properties of serial composition, refinement and conjunction in both interfaces and contracts with respect to the proposed transformation.

**A. Serial Composition and Compatibility**

In the sequel, by abuse of notation, we use the operator \( \otimes \), defined in Section IV-B, to also denote the serial composition of contracts. Serial composition can be obtained from the parallel composition of contracts, in which every pair of shared variables are given the same name to denote the presence of an interconnection. However, while parallel composition is commutative, serial composition is generally non-commutative.

To establish a correspondence between interfaces and contracts, we would like serial composition and compatibility to be preserved under \( \mathcal{F} \), i.e., for the interfaces \( I_1 \) and \( I_2 \),

\[
\mathcal{F}(I_1 \otimes I_2) = \mathcal{F}(I_1) \otimes \mathcal{F}(I_2)
\]

to hold. However, this is not true in general, as shown by the following example.

**Example 3.** Consider the interfaces \( I_1 = (\{x\}, \{y\}, \text{True}) \) and \( I_2 = (\{y\}, \emptyset, y \geq 0) \), shown in Fig. 2(a), where

\[
\square \text{in}(\phi) \rightarrow \square \phi
\]

1\( \square \) takes precedence over \( \rightarrow \), so \( \square \text{in}(\phi) \rightarrow \square \phi \) means \( (\square \text{in}(\phi)) \rightarrow \square \phi \).
We have $\mathcal{I}(I_1) = \{(x, y), 1, 0, 0, 0, 0, 0, 0, 0\}$ and $\mathcal{I}(I_2) = \{(x, y), 0, 0, 0, 0, 0, 0, 0, 0\}$. Moreover, since $I_1 \sim I_2 = \{(x, y), 1, 0, 0, 0, 0, 0, 0, 0\}$, $I_2 \sim I_1 = \{(x, y), 0, 0, 0, 0, 0, 0, 0, 0\}$. On the other hand, we also obtain $\mathcal{I}(I_1 \otimes I_2) = \{(x, y), \Box(y \geq 0), 0\}$, which is clearly not equal to $\mathcal{I}(I_1 \sim I_2)$.

The difference highlighted by Example 3 can be intuitively explained by the incompatibility of $I_1$ and $I_2$. This is correctly expressed by $\phi_{1 \sim I_2}$ being False and reflected into the assumptions of $\mathcal{I}(I_1 \sim I_2)$, which are also False. The contract $\mathcal{I}(I_1 \sim I_2)$ is also incompatible, i.e., any component satisfying $\mathcal{I}(I_1 \sim I_2)$ cannot be hosted by any environment. However, such incompatibility is not immediately detected using $\mathcal{I}(I_1 \otimes I_2)$, which seems to indicate that any sequence $y_n$ satisfying $y_n \geq 0$ for all $n \in \mathbb{N}$ is admitted. Only after observing that $y$ is a controlled variable, we can finally conclude that $\mathcal{I}(I_1 \sim I_2)$ is incompatible, since its assumptions are not $y$-receptive.

As a second attempt, we may try to prove that serial composition is preserved provided the interfaces are compatible. Example 4 shows that this is not the case either.

**Example 4.** Consider the interfaces $I_3 = \{(x, y), y \geq x\}$ and $I_2 = \{y), \emptyset, y \geq 0\}$, shown in Fig. 2(b). We have $\mathcal{I}(I_3) = \{(x, y), 1, 0, 0, 0, 0, 0, 0, 0\}$, $\mathcal{I}(I_2) = \{(x, y), \Box(y \geq 0), 0\}$, $I_3 \sim I_2 = \{(x, y), x \geq 0 \land y \geq x\}$, and $\mathcal{I}(I_3 \sim I_2) = \{(x, y), \Box(y \geq 0), \Box(x \geq 0) \lor \Box(y \geq 0)\}$.

On the other hand, we also obtain $\mathcal{I}(I_3) \otimes I_2) = \{(x, y), \Box(y \geq x) \rightarrow \Box(y \geq 0), \Box(y \geq x)\}$, which is clearly not equal to $\mathcal{I}(I_3 \sim I_2)$. In fact, the sequence $(x_n, y_n)$ where $x_n = -1$ and $y_n = -3$ for all $n \in \mathbb{N}$ satisfies the assumptions of $\mathcal{I}(I_3) \otimes I_2)$ but does not satisfy the ones of $\mathcal{I}(I_3 \sim I_2)$.

Again, we see that the assumptions of $\mathcal{I}(I_3) \otimes I_2)$ in Example 4 refer to output variables, and do not contain the important new assumption $x \geq 0$ induced by interface composition, and which is crucial to guarantee interface compatibility. Note that we can still conclude that $\mathcal{I}(I_3) \otimes I_2)$ is indeed compatible, since its assumptions are $y$-receptive. However, we are also interested in inferring the set of environments that is allowed by the composite contract, as captured by the new assumption $\Box x \geq 0$. To obtain this, we introduce a new projection operation on contracts, which we call assumption projection (AP).

**B. Assumption Projection**

**Definition V.2 (Assumption Projection).** Given a contract $C = (V, A, G)$, and a subset $W \subseteq V$, the assumption projection of $C$ with respect to $W$ (AP$_W$) returns the new saturated contract

$$\text{AP}_W(C) = (V, \forall W : A, (\forall W : A) \rightarrow G).$$

We use the fact that the universal quantifier is commutative and associative to lift it to sets of variables in Definition V.2, so that $\forall W : A := (\forall w_1 : \forall w_2 : \ldots : \forall w_n : A)$ when $W = \{w_1, w_2, \ldots, w_n\}$. Moreover, when the assumptions are expressed by an LTL formula, universal quantification is meant over sequences of valuations over the variables in $W$ [8]. We are now ready to state the following theorem, which relates serial composition of interfaces with serial composition of contracts.

**Theorem V.3 (Assumption Projection Mapping).** Given two disjoint relational interfaces $I_1$ and $I_2$, with sets of output variables $Y_1$ and $Y_2$, respectively, we have

$$\mathcal{I}(I_1 \sim I_2) = \text{AP}_{Y_1 \cup Y_2}(\mathcal{I}(I_1) \otimes \mathcal{I}(I_2)).$$

Moreover, $I_1$ and $I_2$ are compatible iff $\text{AP}_{Y_1 \cup Y_2}(\mathcal{I}(I_1) \otimes \mathcal{I}(I_2))$ is compatible.

According to Theorem V.3, the contract associated to the composition of two interfaces is equivalent to the assumption projection contract of the composition of the associated contracts with respect to the output variables, i.e., we can still map the contract associated to the composition of two interfaces $\mathcal{I}(I_1 \sim I_2)$ to the composition of the associated contracts $\mathcal{I}(I_1) \otimes \mathcal{I}(I_2)$ only after applying the AP operator. Before proving Theorem V.3, we introduce the following lemma, which will be used in the proof.

**Lemma V.4.** Given the interfaces $I_1 = (X_1, Y_1, \phi_1)$ and $I_2 = (X_2, Y_2, \phi_2)$, let $\psi = (\forall Y_1 : \phi_1 \rightarrow m(\phi_2))$, and $\psi' = (\forall Y_1 : \phi_1 \rightarrow m(\phi_2))$. Then, if $\Box m(\phi_1)$ is True, we have $\psi \leftrightarrow \psi'$.

**Proof (Lemma V.4):** Suppose first that $\psi$ is True, and suppose that on all sequences $y_{1,n}$ of valuations over $Y_1$, $\phi_1$ holds. Then, for all $n$, for all valuations $(x_{1,n}, x_{2,n}, y_{1,n})$ over $(X_1, X_2, Y_1)$, we have $(x_{1,n}, x_{2,n}, y_{1,n}) \models \phi_1$. Hence, by $\psi$, we also have that for all $n$, for all the valuations over $(X_1, X_2, Y_1)$, $(x_{1,n}, x_{2,n}, y_{1,n}) \models m(\phi_2)$. This implies that $\Box m(\phi_2)$ is also valid for all sequences of valuations over $Y_1$, and $\psi'$ is True. Therefore, we conclude that $\psi \leftrightarrow \psi'$.

To prove that $\psi' \rightarrow \psi$, we now assume that $\psi$ is False, and prove that $\psi'$ must also be False. In fact, if $\psi$ is False, then there exists a sequence $(x_{1,k}, x_{2,k})$ of valuations over $(X_1, X_2)$, an index $i \in \mathbb{N}$ and a valuation $y^*$ over $Y_1$ such that $(x_{1,i}, x_{2,i}, y^*) \models \phi_1$ and $(x_{1,i}, x_{2,i}, y^*) \not\models m(\phi_2)$. Consider such a sequence $(x_{1,k}, x_{2,k})$. Then, since $\Box m(\phi_1)$ holds by hypothesis, we know that, for all $k$, it is possible to find $y_{1,k}$ such that $(x_{1,k}, y_{1,k}) \models \phi_1$. Therefore, starting from $(x_{1,k}, x_{2,k})$, we can construct a new sequence $s_k = (x_{1,k}, x_{2,k}, y_{1,k})$ such that $\forall k \neq i, y_{1,k} = y_{1,i}$, and for $k = i$, $y_{1,i} = y^*$. By construction, $s_k \models \Box \phi_1$ but $s_k \not\models \Box m(\phi_2)$, i.e., $s_k$ falsifies $\psi'$. We can therefore conclude $\lnot \psi \rightarrow \lnot \psi'$, which is what we wanted to prove.

We can now prove Theorem V.3.

**Proof (Theorem V.3):** Both the left and right-hand side contracts $C_L$ and $C_R$ in (6) are in saturated form by definition of $\mathcal{I}$ and of AP. To prove that $C_L$ and $C_R$ are equal we need to prove that they have the same assumption and guarantee sets. We first compute assumptions and guarantees for $C_R$. By applying (4) and (5) and the definition of $\mathcal{I}$ we obtain:

$$G_{\Box} = (\Box m(\phi_1) \rightarrow \Box \phi_1) \land (\Box m(\phi_2) \rightarrow \Box \phi_2)$$
A_\phi = (\Box in(\phi_1) \land \Box in(\phi_2)) \lor \neg G_\phi \\
= \Box (in(\phi_1) \land in(\phi_2)) \lor (\Box in(\phi_1) \land \neg \Box \phi_1) \\
\lor (\Box in(\phi_2) \land \neg \Box \phi_2) \tag{8}

where A_\phi and G_\phi are the assumptions and guarantees of \mathcal{F}(I_1) \otimes \mathcal{F}(I_2). Finally, after assumption projection, we obtain:

A_R = \forall Y_1 \forall Y_2 : A_\phi \\
= \forall Y_1 : (\Box (in(\phi_1) \land in(\phi_2)) \lor (\Box in(\phi_1) \land \neg \Box \phi_1)) \\
\lor (\forall Y_2 : (\Box in(\phi_2) \land \neg \Box \phi_2)) \\
= \forall Y_1 : (\Box (in(\phi_1) \land in(\phi_2)) \lor (\Box in(\phi_1) \land \neg \Box \phi_1)) \\
= \forall Y_1 : (\Box (in(\phi_1) \land (\Box in(\phi_2) \land \neg \Box \phi_1)) \\
= \Box in(\phi_1) \land (\forall Y_1 : \phi_1 \rightarrow \Box in(\phi_2)) \tag{9}

G_R = AR \rightarrow G_\phi \\
= \Box in(\phi_1) \land (\forall Y_1 : \phi_1 \rightarrow \Box in(\phi_2)) \rightarrow (\Box \phi_1 \lor \neg \Box in(\phi_1)) \land (\Box \phi_2 \lor \neg \Box in(\phi_2)) \tag{10}

Consider now the assumptions of C_L. We obtain:

A_L = \square in(\phi) = \square [(\forall Y_1 \exists Y_2 : \phi_1 \lor \phi_2 \land (\forall Y_1 : \phi_1 \rightarrow in(\phi_2)))] \\
= \square [(\forall Y_1 : \phi_1 \rightarrow in(\phi_2)) \land (\forall Y_1 : \phi_1 \land in(\phi_2))] \\
= [\forall Y_1 : \phi_1 \rightarrow in(\phi_2)] \land in(\phi_1) \tag{11}

while for G_L we obtain:

G_L = \square (\forall Y_1 : \phi_1 \rightarrow in(\phi_2)) \land \square in(\phi_1) \\
\rightarrow (\Box \phi_1 \land \Box \phi_2 \land (\forall Y_1 : \phi_1 \rightarrow in(\phi_2))) \tag{12}

The equivalence of the assumptions A_L and A_R directly descends from Lemma V.4. To prove the equivalence of G_L and G_R it is enough to prove that, if A_L or A_R is True, then

(\Box in(\phi_1) \rightarrow \Box \phi_1) \land (\Box in(\phi_2) \rightarrow \Box \phi_2) \leftrightarrow (\Box \phi_1 \land \Box \phi_2). \tag{13}

Clearly, if the formula on the left side of the double implication in (13) is True, the formula on the right side is also trivially True when A_R and A_L are True. Suppose now that the left-hand side of (12) is True. Since A_L and A_R are True then \Box in(\phi_1) is True, which implies \Box \phi_1 is True. On the other hand, by A_L and A_R being again True, we also have

\Box (\forall Y_1 : \phi_1 \rightarrow in(\phi_2)) \land \Box \phi_1 \rightarrow \Box in(\phi_2). 

This allows us to conclude that \Box \phi_2 is also True and finally (13) holds. We have therefore proved (6).

Let now \phi = \phi_1 \land \phi_2 \land (\forall Y_1 : \phi_1 \rightarrow in(\phi_2)) be the formula associated with I_1 \rightarrow I_2, I_1 and I_2 are compatible if and only if \phi is satisfiable. On the other hand, AP_{Y_1 \cup Y_2}(\mathcal{F}(I_1) \otimes \mathcal{F}(I_2)) is compatible if and only if its assumptions A_R are satisfiable. Then, to prove the last statement of the theorem, we need to prove that \phi is satisfiable if and only if A_R is satisfiable. This can be directly inferred from the fact that A_R = AR = \Box in(\phi). In fact, \Box in(\phi) is satisfiable if and only if in(\phi) is satisfiable, i.e. if and only if \phi is satisfiable, which concludes our proof.

The assumption projection mapping allows preserving the semantics of interface composition; we demonstrate its application to Examples 1-4 discussed above. In Example 1, while computing the assumptions of AP(\mathcal{F}(C_L \otimes C_D)), we obtain, as expected, A_{ex1} = (\forall y : y \neq 0) = False, which corresponds to CL and CD being incompatible. A similar result is obtained in Example 3, where A_{ex3} = (\forall y : \Box (y \geq 0)) = False are the assumptions of AP(\mathcal{F}(I_1) \otimes \mathcal{F}(I_2)). By applying assumption projection to CL \otimes CD in Example 2, we instead obtain

A_{ex2} = (\forall y : y \neq 0 \lor x \geq y = \neg (\exists y : (y = 0 \land x < y)) \\
= \neg (\exists y : y = 0) \land x < 0) \\
= x \geq 0,

which corresponds to the desired set of legal environments for the composite specification. Finally, in Example 4, we need to compute A_{ex4} = (\forall y : \Box (y \geq x) \rightarrow \Box (y \geq 0)). To do this, it is convenient to exchange the order between the quantifier and the temporal construct, by using Lemma V.4, with \psi = (\forall y : (y \geq x) \rightarrow (y \geq 0)) and \psi' = A_{ex4} = (\forall y : \Box (y \geq x) \rightarrow \Box (y \geq 0)). Then, since the hypothesis of the lemma is satisfied, we conclude that \psi is equivalent to \psi' and, in particular,

A_{ex4} = (\forall y : (y \geq x) \rightarrow (y \geq 0)) \\
= (\forall \neg (\exists y : (y \geq x) \land (0 > y)) \\
= (\forall \neg (\exists y : x \leq y < 0) = (\forall \neg (x < 0) = \forall \neg x \geq 0),

which, as expected, preserves the equivalence with the contract associated to the composite interface.

C. Implementing Assumption Projection in Temporal Logic

Assumption projection hides the controlled (output) variables of a composite contract from its assumptions, thus enabling preservation of serial composition and compatibility between interfaces and their associated contracts. However, we observe that this operator is not straightforward to implement, since LTL is not closed under projection [21]. For instance, consider the LTL formula \phi over two Boolean variables \textit{s} and \textit{p}:

\phi := p \land \Box (s \rightarrow p) \land \Box (s \rightarrow \neg \Box s) \land \Box (\neg s \rightarrow \Box s).

It can be shown that there is no LTL formula over \textit{p} that characterizes exactly the set of infinite traces obtained by projecting the traces characterized by \phi onto the \textit{p} variable.

It is, however, possible to resort to an extension of (propositional) LTL, namely Quantified (Propositional) Linear Temporal Logic (QKTL/QPTL), which introduces quantification over propositions [7], [8]. Having an expressive power equal to that of \omega-automata, QKTL is strictly more expressive than LTL, and has been used for formulating and verifying refinement relations between reactive systems or programs [22]. However, we may pay for the augmented expressive power with a substantial increase in complexity. In fact, checking compatibility between two (Q)KTL A/G contracts after assumption projection can always be reduced to solving a satisfiability problem for QKTL, which is decidable with non-elementary complexity [7].

Every QKTL formula can be written in normal form as

\langle Q_1 p_1 Q_2 p_2 \ldots Q_k p_k : \varphi \rangle, \tag{14}

where \{Q_1, Q_2, \ldots, Q_k\} is a finite sequence of existential or universal quantifiers, P = \{p_1, p_2, \ldots, p_k\} a finite sequence of propositional variables, and \varphi is a quantifier-free formula. For each alternation of existential and universal quantifiers in (14) the space complexity increases by exactly one exponential. For
instance, a QLTL formula of the form $(\exists P : \varphi)$, where $\varphi$ is an LTL formula of size $n$, is said to be in the set $\Sigma_{\text{QLTL}}$. The satisfiability problem for this set of formulas, which are also denoted as Existentially Quantified LTL (EQLTL) formulas, is PSPACE-complete, like that for LTL. On the other hand, a formula of the form $(\forall P : \varphi)$ is instead said to be in $\Pi_{\text{QLTL}}$. For this set of formulas the satisfiability problem becomes complete for $\text{SPACE}(2^n)$. A sound and complete proof system for QLTL is provided in [23].

D. Refinement

While $\mathcal{F}$ does not generally preserve serial composition, it preserves refinement, i.e. the mapping is monotonous, as the following theorem shows.

**Theorem V.5 (Refinement Preservation).** Given two relational interfaces $I_1$ and $I_2$, then $I_1 \sqsubseteq I_2$ if and only if $\mathcal{F}(I_1) \sqsubseteq \mathcal{F}(I_2)$.

**Proof:** Let $I_1 = (X_1, Y_1, \phi_1)$ and $I_2 = (X_2, Y_2, \phi_2)$. By definition of refinement, we recall that $I_1 \sqsubseteq I_2$ if and only if $(\square \neg \phi_2 \rightarrow \square \neg \phi_1 \land \phi_1 \rightarrow \phi_2)$ is valid or, equivalently, the following two formulas

\[
\begin{align*}
\square \neg \phi_2 & \rightarrow \square \neg \phi_1 \quad (15) \\
\square \phi_2 \land \phi_1 & \rightarrow \phi_2 \quad (16)
\end{align*}
\]

are both valid. Moreover, by definition of $\mathcal{F}$, we have

\[
\begin{align*}
\mathcal{F}(I_1) &= (Y_1 \cup X_2, \square \neg \phi_1, \square \neg \phi_1) \\
\mathcal{F}(I_2) &= (Y_1 \cup X_2, \square \phi_2, \square \phi_2).
\end{align*}
\]

We first prove that $I_1 \sqsubseteq I_2 \rightarrow \mathcal{F}(I_1) \sqsubseteq \mathcal{F}(I_2)$. Let $A_1$ and $G_1$ be, respectively, the assumptions and the guarantees of $\mathcal{F}(I_1)$. We need to show that formulas (15) and (16) imply $A_2 \rightarrow A_1$ and $G_1 \rightarrow G_2$. Assume $A_2$ is True, then, by (15), $A_1$ is also True; therefore, $A_2 \rightarrow A_1$. Assume now that $G_1$ is True, i.e. either $\square \neg \phi_1$ is False or $\neg \phi_1$ is True. If $\neg \phi_1$ is False, then from $A_2 \rightarrow A_1$, $\neg \phi_1$ is also False, which makes $G_2$ True. If $\neg \phi_1$ is True, then, by (16), we conclude $\square \phi_2 \rightarrow \phi_2$, hence $G_2$ is again True. We therefore conclude that $G_1 \rightarrow G_2$.

We now prove that if $\mathcal{F}(I_1) \sqsubseteq \mathcal{F}(I_2)$, i.e. $A_2 \rightarrow A_1$ and $G_1 \rightarrow G_2$, then (15) and (16) are valid. To do so, we assume instead that $I_2 \not\sqsubseteq I_2$ and show that $\mathcal{F}(I_1) \not\sqsubseteq \mathcal{F}(I_2)$. In fact, if (15) is not valid, then we can create a sequence $\sigma_n$ of valuations over $X_2$ and an index $i$ such that $\sigma_n \models \phi_2$ for all $n$, and $x_i \not\models \phi_1$. Then, for such a sequence, $\square \neg \phi_2$ is True while $\square \neg \phi_1$ is False, which means that $A_2$ is not valid. Similarly, assume (16) is not valid; then we can create a sequence of valuations $(x_n, y_n)$ for the variables in $X_2 \cup Y_1$ and an index $i$ such that $(x_n, y_n) \models \phi_2$ and $(x_n, y_n) \models \phi_1$ for all $n$, while $(x_i, y_i) \not\models \phi_2$. However, this implies that $\square \phi_2$, hence $G_1$ is True while $G_2$ is False, since $\square \phi_2$ is True without $\phi_2$ being True. Therefore, $G_1 \rightarrow G_2$ is also not valid, which allows us to conclude $(I_1 \not\sqsubseteq I_2) \rightarrow (\mathcal{F}(I_1) \not\sqsubseteq \mathcal{F}(I_2))$, as we wanted to prove.

To enable compositional methods in system design, it is useful to investigate whether refinement is preserved by composition. For both relational interfaces and A/G contracts refinement is preserved by parallel composition and serial composition [4], [5]. For instance, given the A/G contracts $C_1, C_2, C_1', C_2'$, if $C_1' \sqsubseteq C_1$, $C_2' \sqsubseteq C_2$ and $C_1$ is compatible with $C_2$, we are allowed to conclude that $C_1'$ is also compatible with $C_2'$ and $C_1' \otimes C_2' \sqsubseteq C_1 \otimes C_2$. Therefore, compatible contracts can be independently refined, which is key to enable top-down incremental design, by iteratively decomposing a system-level contract $C$ into sub-system contracts $C_i$ for further independent development.

However, refinement is not always preserved by feedback composition in both frameworks. In relational interfaces, there is no composition operator that supports feedback loops for stateless interfaces. For feedback, an interface is required to be Moore with respect to the input variables involved in the connection. In a Moore interface, if an output $y$ is fed-back to an input $x$, then $y$ may only depend on state variables and on the current value of the input variables which are not connected to it, i.e. the ones in $X \setminus \{x\}$ [4]. This definition, inspired by Moore machines, allows forming feedback loops without creating causality cycles. As a result, in relational interfaces, feedback preserves refinement only if the interfaces are Moore with respect to the input variables involved in the connection.

Mapping feedback composition of relational interfaces into A/G contracts would require dealing with stateful interfaces, which is out of the scope of this paper. In what follows, we discuss just one property of interest. Since feedback connections in A/G contracts can be defined based on the parallel composition operator, contract refinement is preserved by feedback composition. In Theorem V.7, we show that this is also the case for the contracts associated to two relational interfaces, even if the original interfaces are not Moore. To do this, we first provide a definition of feedback for A/G contracts.

**Definition V.6 (Feedback Composition of A/G Contracts).** Given a contract $C = (V, A, G)$ and a feedback connection $\kappa = (x, y) \in V^2$ on $C$, let $C_{id}$ be the contract defined as $C_{id} = (\{x, y\}, True, \square (x = y))$. Then, $\kappa$ defines a new contract $\kappa(C) := C \otimes C_{id}$.

**Theorem V.7 (Refinement under Feedback Composition).** Let $I_1 = (X, Y, \phi_1)$ and $I_2 = (X, Y, \phi_2)$ be two relational interfaces and $\kappa = (x, y) \in X \times Y$ a feedback connection on the associated contracts $\mathcal{F}(I_1)$ and $\mathcal{F}(I_2)$, then

\[
(I_1 \sqsubseteq I_2) \rightarrow (\kappa(\mathcal{F}(I_1)) \sqsubseteq \kappa(\mathcal{F}(I_2))),
\]

provided that $\kappa(\mathcal{F}(I_2))$ is compatible.

**Proof:** By Theorem V.5, we know that if $I_1 \sqsubseteq I_2$ then $\mathcal{F}(I_1) \sqsubseteq \mathcal{F}(I_2)$. By definition of $\kappa$, we also have $\kappa(\mathcal{F}(I_1)) = \mathcal{F}(I_1) \otimes C_{id}$ and $\kappa(\mathcal{F}(I_2)) = \mathcal{F}(I_2) \otimes C_{id}$. $C_{id}$ being the contract $(\{x, y\}, True, \square (x = y))$. Then, by the property of independent implementability of the parallel composition of contracts [6], if $\kappa(\mathcal{F}(I_2))$ is compatible, we can conclude that $\kappa(\mathcal{F}(I_1))$ is also compatible and $\kappa(\mathcal{F}(I_1)) \sqsubseteq \kappa(\mathcal{F}(I_2))$, as we wanted to show.

As observed above, (17) holds even if $I_1$ and $I_2$ are not Moore with respect to $x$. A similar definition of feedback as the one stated in Definition V.6 can also be extended to non-Moore interfaces as follows.

**Definition V.8 (Feedback for Non-Moore Interface).** Given an interface $I = (X, Y, \phi)$, a feedback connection $\kappa = (x, y) \in X \times Y$ defines a new interface $\kappa(I) = (X \setminus \{x\}, Y \cup \{x\}, \phi \land (x = y))$. 
However, if $I_1$ and $I_2$ are not Moore, $\kappa(I_1) \subseteq \kappa(I_2)$ is not guaranteed. This may happen either because $\phi_{\kappa(I_2)}$ is False or because $\phi_{\kappa(I_1)}$ is True, but $\phi_{\kappa(I_1)} \not\rightarrow \phi_{\kappa(I_2)}$. We conclude this section by illustrating how the two cases above are mapped into the A/G contract framework, using the following two examples.

Example 5 (Feedback-Induced Inconsistency). Consider $I_A = \{\{x, z\}, \{y\}, True\}$ and $I_B = \{\{x, z\}, \{y\}, x \neq y\}$ as in Fig. 3 (a). $I_A$ does not make any assumptions on the inputs and any guarantee on the outputs, while $I_B$ guarantees that the value of the output is different from the value of the input. We have $I_B \subseteq I_A$ since $in(\phi_A) = in(\phi_B) = True$ and $\phi_B \rightarrow \phi_A$.

However, given $\kappa(I_A) = \{\{x, y\}, True\}$ and $\kappa(I_B) = \{\{x, y\}, False\}$, obtained as shown in Fig. 3 (b), clearly $\kappa(I_B) \not\subseteq \kappa(I_A)$ since $\phi_{\kappa(I_A)}$ is False.

Consider now the associated contracts $A = (V, True, True)$ and $B = (V, True, \square(y \neq x))$ on variables $V = \{x, y, z\}$. We have $B \leq A$, $\kappa(A) = (V, True, \square(y = x))$, $\kappa(B) = (V, True, False)$, and $\kappa(B) \leq \kappa(A)$. Therefore, refinement is preserved by feedback composition. However, the feedback connection has caused an inconsistency internal to $\kappa(B)$; $\phi_{\kappa(I_B)} = False$ reflects $\kappa(B)$ being inconsistent.

Example 6 (Feedback-Induced Inconsistency). Consider the two interfaces $I_A$ and $I_B$ defined as follows:

$I_A = \{\{x\}, \{y\}, (x \neq 0) \land (xy = 1)\}$,

$I_B = \{\{x\}, \{y\}, (x \neq 0) \rightarrow (xy = 1)\}$.

We have $I_B \subseteq I_A$ since $in(\phi_A) = (x \neq 0)$, $in(\phi_B) = True$ and $\phi_B \rightarrow \phi_A = (x \neq 0)$. However, given $\kappa(I_A)$ and $\kappa(I_B)$ computed as follows:

$\kappa(I_A) = (\emptyset, \{x, y\}, (x^2 = 1) \land (x = y))$,

$\kappa(I_B) = (\emptyset, \{x, y\}, (x \neq 0) \rightarrow (x^2 = 1) \land (x = y))$,

we obtain $in(\phi_{\kappa(I_B)}) = in(\phi_{\kappa(I_A)}) = True$, $\phi_{\kappa(I_B)} \not\rightarrow \phi_{\kappa(I_A)}$, and therefore $\kappa(I_B) \not\subseteq \kappa(I_A)$. In fact, $\kappa(I_B)$ admits all sequences of the form $(x_n, x_m)$, with $x_n \in \{-1, 0, 1\}$ for all $n \in \mathbb{N}$, while $x_m = 0$ is not allowed by $\kappa(I_A)$. Consider now the associated contracts $A$ and $B$ defined below on variables $V = \{x, y\}$:

$A = (V, \square(x \neq 0), \square(x \neq 0) \rightarrow \square(xy = 1))$,

$B = (V, True, \square((x \neq 0) \rightarrow (xy = 1)))$.

We can compute $\kappa(A)$ and $\kappa(B)$ as

$\kappa(A) = (V, \square(x \neq 0) \lor \square(y = x), \square((x \neq 0) \rightarrow (x^2 = 1) \land \square(y = x)))$,

$\kappa(B) = (V, True, \square((x \neq 0) \rightarrow (x^2 = 1)) \land (y = x))$.

In this case, $\kappa(B)$ is compatible and consistent. Moreover, we have $\kappa(B) \leq \kappa(A)$, i.e. refinement is preserved by feedback.

However, even if $\kappa(A)$ is compatible for the general A/G contract framework, the assumption projection of $\kappa(A)$ with respect to either $x$ or $y$ unveils a potential source of incompatibility induced by feedback. For example, the assumptions $AP_y(\kappa(A))$ would require $\square x \neq 0$. However, such a condition may not be guaranteed per se by the feedback connection.

E. Conjunction

In this section, we will use the term conjunction to also denote shared refinement, which is the counterpart of conjunction for relational interfaces. We show that even if $\mathcal{F}$ preserves refinement, it does not preserve conjunction. First, as mentioned in Section IV, conjunction cannot always be defined for relational interfaces. For A/G contracts, conjunction is instead always defined as the GLB of the refinement relation. However, when the set of shared refinements of two interfaces is empty, the conjunction of their associated contracts can become inconsistent, as illustrated by the following example.

Example 7 (Inconsistent Conjunction). Consider $I_{00} = \{(x, y), x = 0 \rightarrow y = 0\}$ and $I_{01} = \{(x, y), x = 0 \rightarrow y = 1\}$. They are not shared refinable, since it is not possible to guarantee both $y = 0$ and $y = 1$ when $x = 0$ [4]. However, conjunction can still be defined for the associated contracts $\mathcal{F}(I_{00}) = \{(x, y), True, \square(x = 0 \rightarrow y = 0)\}$ and $\mathcal{F}(I_{01}) = \{(x, y), True, \square(x = 0 \rightarrow y = 1)\}$, although it corresponds to the inconsistent contract $\{(x, y), True, False\}$, i.e. the “bottom” element for the refinement preorder.

When conjunction is well-defined in relational interfaces, the contract associated with the conjunction of two interfaces is, in general, a refinement of the conjunction of the contracts associated with the interfaces, as stated by the following theorem.

Theorem V.9 (Interface Conjunction Refines Contract Conjunction). Let $I = (X, Y, \phi)$ and $I' = (X, Y, \phi')$ be two shared-refinable relational interfaces. Then we have

$\mathcal{F}(I \cap I') \leq \mathcal{F}(I) \wedge \mathcal{F}(I')$,   \hspace{1cm} (18)

where, in general, $\mathcal{F}(I \cap I') \neq \mathcal{F}(I) \wedge \mathcal{F}(I')$.

While (18) can be easily proved as a direct consequence of Theorem V.5, our main goal is to gather insight into the reason why the equality does not generally hold, and the GLB is not preserved by $\mathcal{F}$. Therefore, we present an alternative proof, which directly reasons about the assumption and guarantee sets of the contracts in (18).

Proof (Theorem V.9): We recall that $I \cap I' = (X, Y, \phi_\cap)$, where

$\phi_\cap = \{in(\phi) \lor in(\phi')\} \land \{in(\phi) \rightarrow \phi\} \land \{in(\phi') \rightarrow \phi'\}$,

and $in(\phi_\cap) = \{in(\phi) \lor in(\phi')\}$ by Lemma 8 in [4]. Therefore, by transforming $I \cap I'$, we obtain $\mathcal{F}(I \cap I') = (X \cup Y, A_\cap, G_\cap)$, where $A_\cap = \square(in(\phi) \lor in(\phi'))$ and $G_\cap = \square(in(\phi) \lor in(\phi')) \rightarrow \square(in(\phi) \lor in(\phi'))$. Moreover, by definition of conjunction, we obtain $\mathcal{F}(I) \wedge \mathcal{F}(I') = (X \cup Y, A_\cap, G_\cap)$, where $A_\cap = \square(in(\phi) \lor in(\phi'))$ and $G_\cap = \square(in(\phi) \lor in(\phi')) \rightarrow \square(in(\phi) \lor in(\phi'))$. 


It is easy to see that $A_L \rightarrow A_R$. On the other hand, we also notice that $A_L \not\rightarrow A_R$, in general. In fact, any sequence $x_n$ such that $x_1 = \prec (\phi)$, $x_1 \not\prec \prec (\phi')$, and $x_n \equiv \prec (\phi')$ for all $n > 1$, satisfies $A_R$ but does not satisfy $A_L$.

We also notice that $G_\cap \rightarrow G_A$. In fact, $G_A$ is trivially $\text{True}$ if both $\Box \prec (\phi)$ and $\Box \prec (\phi')$ are False. If $\Box \prec (\phi')$ is instead $\text{True}$, then, because $G_\cap$ is True, $\Box \prec (\phi \rightarrow \phi')$ is $\text{True}$, which implies that $\Box \phi$ is also $\text{True}$. Similarly, if $\Box \prec (\phi')$ is $\text{True}$, $\Box \phi'$ will also be $\text{True}$. Therefore, in all cases, both the implications in $G_A$ will be $\text{True}$ under the assumption that $G_\cap$ is $\text{True}$. On the other hand, we also notice that $G_A \not\rightarrow G_\cap$ in general. In fact, any sequence $(x_n, y_n)$ such that $x_1 = \prec (\phi)$, $x_1 \not\prec \prec (\phi')$, $x_n \equiv \prec (\phi')$ for all $n > 1$, and $(x_1, y_1) \not\prec \phi$, would certainly satisfy $G_A$ but not $G_\cap$.

As stated by Theorem V.9, the contract associated with the conjunction of $I$ and $I'$ is not generally equal to the conjunction of the contracts associated with $I$ and $I'$. Moreover, the expressions of the assumptions and guarantees computed in our proof highlight another crucial difference between the two frameworks, which explains this result. While an A/G contract reasons about the entire behavior of a component, possibly spanning infinite sequences of reactions, a relational interface can constrain the inputs and outputs of a component at the granularity of a single reaction index. Therefore, computation of conjunction generates a smaller set of allowed environments and a larger set of guaranteed behaviors for A/G contracts, which translates into a tighter, less conservative, greatest lower bound.

VI. CONCLUSIONS AND FUTURE WORK

This paper has established a link between the theory of relational interfaces and the one of A/G contracts, shedding light on some of their key features for system specification, early detection of incompatibilities, and use of abstraction-refinement.

We have proposed a natural transformation from interfaces to LTL A/G contracts, and we have discussed the subtleties involved in it. The transformation preserves refinement, but does not generally preserve serial composition, and in particular interface compatibility. To address this, we have proposed a new assumption-projection operator on contracts that captures the distinct nature of inputs and outputs during hiding, thus enabling preservation of the semantics of interface composition. Finally, we have shown that computing the conjunction of transformed contracts generally translates into a more abstract specification than transforming the conjunction of the associated interfaces.

Future extensions of this work include studying the properties of the proposed transformation with respect to feedback composition, as well as its generalization to the theory of interface automata. We are also interested in investigating a reverse transformation that maps A/G contracts into functional interfaces, which requires extending the latter with liveness properties. Finally, algorithms for the efficient implementation of the assumption-projection operator on QLTL contracts will also be considered as future work.

REFERENCES


