From Relational Interfaces to Assume-Guarantee Contracts

Pierluigi Nuzzo
Antonio Iannopollo
Stavros Tripakis
Alberto L. Sangiovanni-Vincentelli

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-21
http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-21.html

March 18, 2014

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
18 MAR 2014		**00-00-2014 to 00-00-2014**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
From Relational Interfaces to Assume-Guarantee Contracts	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Berkeley,Electrical Engineering and Computer Sciences,Berkeley,CA,94720**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Contract-based design is emerging as a unifying composi- tional paradigm for the speci cation, design and veri ca- tion of large-scale complex systems. Yet, di erent contract frameworks are currently available, without a clear under- standing of the relations between them. In this paper, we investigate the relation between interface theories (specif- ically, relational interfaces) and assume-guarantee (A/G) contracts, revealing some of the subtleties involved. We show that the natural transformation of interfaces to A/G contracts represented by LTL formulas preserves re nement but does not generally preserve serial composition, and we present an assumption-projection operator to remedy the latter issue. We also discuss the properties of our trans- formation with respect to conjunction. Finally, we provide illustrative examples that shed light on the e ectiveness of both frameworks for requirement formalization, early detec- tion of integration errors, and principled use of abstraction- re nement.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**9**	

Acknowledgement

This work was partially supported by IBM and United Technologies Corporation (UTC) via the iCyPhy consortium, by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, by the Academy of Finland, and by the National Science Foundation, via projects "ExCAPE: Expeditions in Computer Augmented Program Engineering" and "COSMOI: Compositional System Modeling with Interfaces."

From Relational Interfaces to Assume-Guarantee Contracts

Pierluigi Nuzzo[§], Antonio Iannopollo[§],
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. Yet, different contract frameworks are currently available, without 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, revealing some of the subtleties involved. We show that the natural transformation of interfaces to A/G contracts represented by LTL formulas preserves refinement, but does not generally preserve serial composition, and we present an assumption-projection operator to remedy the latter issue. We also discuss the properties of our transformation with respect to conjunction. Finally, we provide illustrative examples that shed light on the effectiveness of both frameworks for requirement formalization, early detection of integration errors, and principled use of abstraction-refinement.

1. 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 proposed in [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 $\Box\phi$.

We then study the preservation properties of the above transformation. We show that, perhaps surprisingly, the basic operation of *serial composition* of interfaces is not preserved. Specifically, composing two interfaces I_1 and I_2, and then transforming the result to an A/G contract, is not equivalent to first transforming each of I_1 and I_2 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 a-priori notion of compatibility during composition. Although compatibility can be checked a-posteriori on the composite contract using the notion of *c-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 unclear how to implement this hiding operator at the A/G contract level.

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* (called *shared refinement* in [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.

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. The authors in [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, the work in [7] attempts 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.

Another theory of A/G contracts is proposed in [8] to support rich component interactions by replacing the notion of parallel composition with the one of circular reasoning. However, compatibility and conjunction are not addressed in this framework. On the other hand, in [9], an interface model similar to relational interfaces is proposed, 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.

The rest of the paper is organized as follows. We briefly summarize relational interfaces and A/G contracts in Section 2. In Section 3, we present the main results of the paper together with several illustrative examples. Finally, in Section 4, we draw some conclusions.

2. BACKGROUND

We recall the salient parts of the relational interface and A/G contracts frameworks.

2.1 Synchronous Relational Interfaces

For simplicity, we restrict ourselves to *stateless* interfaces. A (relational) interface is a tuple $I = (X, Y, \phi)$ where X and Y are finite sets of input and output variables, respectively, and ϕ is a logical formula on the variables in $X \cup Y$. The sets of input and output variables must be disjoint: $X \cap Y = \emptyset$. To relate to A/G contracts, we assume that all variables in $X \cup Y$ range over the same set of values $\mathcal{U}$. A *valuation* over V is a function $v : V \to \mathcal{U}$ where $\mathcal{U}$ is the set of possible values for the variables. A valuation v over V satisfies a formula ϕ over the same set of variables V, written $v \models \phi$, if replacing free variables in ϕ by their value as specified by v yields a formula that evaluates to *True*. A formula ϕ defines the following set of behaviors:

$$[\![\phi]\!] := \{v_0 v_1 v_2 \cdots \mid \forall i : v_i \models \phi\}.$$

Note that $[\![\phi]\!]$ is a safety property.

Given interface $I = (X, Y, \phi)$, the *input assumption* defined by ϕ is the formula $in(\phi) := \exists Y : \phi$, where $\exists Y : \phi$ is $\exists y_1 : \exists y_2 : \cdots \exists y_n : \phi$ when $Y = \{y_1, y_2, ..., y_n\}$. $in(\phi)$ characterizes the legal inputs. An input is considered illegal if there is no output which can satisfy ϕ for that input. Note that $in(\phi)$ is a formula on X only, as variables in Y have been eliminated by existential quantification. For example, if $X = \{x\}$, $Y = \{y\}$, and ϕ is $x \geq 0 \wedge y = x$, then $in(\phi)$ is $x \geq 0$. If ϕ is $x \geq 0 \to y = x$, then $in(\phi)$ is *True*.

Composition: Serial composition of two interfaces $I_1 = (X_1, Y_1, \phi_1)$ and $I_2 = (X_2, Y_2, \phi_2)$ can be defined provided all sets X_1, Y_1, X_2, Y_2 are pairwise disjoint, except possibly the pair Y_1, X_2. Let $V_c = Y_1 \cap X_2$. The interpretation is that variables in V_c are outputs of I_1 which are connected to inputs of I_2. Note that we allow V_c to be empty, in which case serial composition reduces to parallel composition (where no connections between the two interfaces exist). Then, the composite interface $I_1 \rightsquigarrow I_2$ is defined to be the interface

$$I_1 \rightsquigarrow I_2 := (X_1 \cup X_2 \setminus Y_1, Y_1 \cup Y_2, \phi)$$

where

$$\phi = \phi_1 \wedge \phi_2 \wedge \forall Y_1 : \big(\phi_1 \to in(\phi_2)\big).$$

I_1 and I_2 are said to be *compatible interfaces* if ϕ is satisfiable, i.e., if ϕ is not equivalent to *False*.

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 $X_1 \subseteq X_2$, $Y_1 \supseteq Y_2$ and the following formula is valid (i.e., true under all valuations):

$$in(\phi_2) \to \big(in(\phi_1) \wedge (\phi_2 \to \phi_1)\big).$$

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:

$$\forall X : \Big(\big(in(\phi_1) \wedge in(\phi_2)\big) \to \big(\exists Y : (\phi_1 \wedge \phi_2)\big)\Big)$$

If I_1 and I_2 are shared-refinable, their shared refinement, denoted $I_1 \sqcap I_2$, is defined to be the interface $I_1 \sqcap I_2 := (X, Y, \phi_\sqcap)$, where

$$\phi_\sqcap := \big(in(\phi_1) \vee in(\phi_2)\big) \wedge \big(in(\phi_1) \to \phi_1\big) \wedge \big(in(\phi_2) \to \phi_2\big)$$

It can be seen that $I_1 \sqcap I_2$, when it exists, is guaranteed to refine both I_1 and I_2, which, as argued in [5,9], is important for component reuse (see also [10]).

2.2 Assume/Guarantee Contracts

Following [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. 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, $\rho = v_0 v_1 v_2 \cdots$. 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 $\overline{A} \subseteq G$, where $\overline{A}$ 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 assumption; formally, $E \subseteq A$.

Composition: 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]:

$$A = (A_1 \cap A_2) \cup \overline{(G_1 \cap G_2)} \quad (1)$$

$$G = G_1 \cap G_2. \quad (2)$$

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

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 $C_1 \preceq C_2$, 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. Clearly, if $M \models C'$ and $C' \preceq C$, then $M \models C$. On the other hand, if $E \models_E C$, then $E \models_E C'$. In other words, contract C' refines another contract C, if C' admits less implementations than C, but more legal environments than C. This is a standard concept inspired by the notion of behavioral subtyping [7].

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 $C_1 \wedge C_2 = (V, A_1 \cup A_2, G_1 \cap G_2)$. Conjunction of A/G contracts is similar to shared refinement in interfaces. Note, however, that shared refinement of interfaces is not always defined, whereas conjunction of A/G contracts is always defined.

2.3 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) [11]. An LTL A/G contract is then a triple $(V, \varphi_a, \varphi_g)$, where φ_a and φ_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 $(V, \Box x \geq 0, \Box y \geq 0)$. An LTL formula represents a set of behaviors. For example, the formula $\Box x \geq 0$ represents the set of all behaviors where x is never negative.

Most operations on contracts can be implemented as operations on LTL formulas in a straightforward way. For instance, saturation of $(V, \varphi_a, \varphi_g)$ can be achieved by setting $\varphi_g := \varphi_a \rightarrow \varphi_g$; checking that $(V, \varphi_a, \varphi_g)$ refines $(V, \varphi'_a, \varphi'_g)$ amounts to checking that $\varphi'_a \rightarrow \varphi_a$ and $\varphi_g \rightarrow \varphi'_g$ are both valid.

3. FROM SYNCHRONOUS RELATIONAL INTERFACES TO A/G CONTRACTS

Definition 3.1 (Contract Associated with an Interface)**.** *An interface $I = (X, Y, \phi)$ can be transformed into a contract*

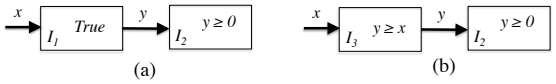

Figure 1: Pictorial representation of the relational interfaces in Example 1 (a) and Example 2 (b).

$C = \mathcal{F}(I) = (V, A, G)$ *where*

$$V := X \cup Y, \quad A := \Box in(\phi), \quad G := \Box in(\phi) \rightarrow \Box\phi.^1$$

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

Even though $in(\phi)$ is a formula over only the set of input variables X, when we define A we choose to interpret $in(\phi)$ 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 $\Box\phi$ denotes the set of behaviors $[\![\phi]\!]$. By definition, contract $\mathcal{F}(I)$ is in saturated form. In what follows, we analyze the behavior of the proposed transformation with respect to serial composition, refinement and conjunction.

3.1 Serial Composition and Compatibility

We would expect that $\mathcal{F}$ preserves serial composition, i.e., for the interfaces I_1 and I_2, $\mathcal{F}(I_1 \rightsquigarrow I_2) = \mathcal{F}(I_1) \otimes \mathcal{F}(I_2)$ holds. However, this is not true in general, as shown by the following example.

Example 1. *Consider the interfaces $I_1 = (\{x\}, \{y\}, True)$ and $I_2 = (\{y\}, \emptyset, y \geq 0)$, shown in Fig. 1(a). We have $\mathcal{F}(I_1) = (\{x, y\}, True, True)$ and $\mathcal{F}(I_2) = (\{x, y\}, \Box(y \geq 0), True)$. Moreover, since $I_1 \rightsquigarrow I_2 = (\{x\}, \{y\}, False)$, we have $\mathcal{F}(I_1 \rightsquigarrow I_2) = (\{x, y\}, False, True)$. On the other hand, we also obtain $\mathcal{F}(I_1) \otimes \mathcal{F}(I_2) = (\{x, y\}, \Box(y \geq 0), True)$, which is clearly not equal to $\mathcal{F}(I_1 \rightsquigarrow I_2)$.*

The difference highlighted by Example 1 can be intuitively explained by the incompatibility of I_1 and I_2. This is correctly expressed by $\phi_{I_1 \rightsquigarrow I_2}$ being *False* and reflected into the assumptions of $\mathcal{F}(I_1 \rightsquigarrow I_2)$, which are also *False*, meaning that the contract $\mathcal{F}(I_1 \rightsquigarrow I_2)$ is also incompatible, i.e. any component satisfying $\mathcal{F}(I_1 \rightsquigarrow I_2)$ cannot be hosted by any environment. On the other hand, such incompatibility is not immediately detected using $\mathcal{F}(I_1) \otimes \mathcal{F}(I_2)$, 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 $\mathcal{F}(I_1) \otimes \mathcal{F}(I_2)$ 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 2 shows that this is not the case either.

Example 2. *Consider the interfaces $I_3 = (\{x\}, \{y\}, y \geq x)$ and $I_2 = (\{y\}, \emptyset, y \geq 0)$, shown in Fig. 1(b). We have $\mathcal{F}(I_3) = (\{x, y\}, True, \Box(y \geq x))$, $\mathcal{F}(I_2) = (\{x, y\}, \Box(y \geq 0), True)$, $I_3 \rightsquigarrow I_2 = (\{x\}, \{y\}, x \geq 0 \wedge y \geq x)$, and*

$$\mathcal{F}(I_3 \rightsquigarrow I_2) = (\{x, y\}, \Box(x \geq 0), \Box(x \geq 0) \rightarrow \Box(y \geq x)).$$

On the other hand, we also obtain

$$\mathcal{F}(I_3) \otimes \mathcal{F}(I_2) = (\{x, y\}, \Box(y \geq x) \rightarrow \Box(y \geq 0), \Box(y \geq x)),$$

which is clearly not equal to $\mathcal{F}(I_3 \rightsquigarrow I_2)$. 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 $\mathcal{F}(I_3) \otimes \mathcal{F}(I_2)$ but does not satisfy the ones of $\mathcal{F}(I_3 \rightsquigarrow I_2)$.

Again, we see that the assumptions 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 $\mathcal{F}(I_3) \otimes \mathcal{F}(I_2)$ is indeed compatible, since its assumptions are y-receptive. However, we are also interested in inferring the largest set of environments, with respect to set inclusion, that is allowed by the composite contract, 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).

Definition 3.2. *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*

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

We use the fact that the universal quantifier is commutative and associative to lift it to sets of variables in Definition 3.2, so that $\forall W : A := (\forall w_1 : \forall w_2 : \ldots : \forall w_n : A)$ when $W = \{w_1, w_2, \ldots, w_n\}$. We are now ready to state the following theorem, which relates serial composition of interfaces with serial composition of contracts.

Theorem 3.3. *Given two relational interfaces I_1 and I_2 with sets of output variables Y_1 and Y_2, respectively, we have*

$$\mathcal{F}(I_1 \rightsquigarrow I_2) = \mathrm{AP}_{Y_1 \cup Y_2}(\mathcal{F}(I_1) \otimes \mathcal{F}(I_2)). \tag{3}$$

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

Before proving Theorem 3.3, we introduce the following lemma, which will be used in the proof.

Lemma 3.4. *Given the interfaces $I_1 = (X_1, Y_1, \phi_1)$ and $I_2 = (X_2, Y_2, \phi_2)$, let $\psi = \Box(\forall Y_1 : \phi_1 \rightarrow in(\phi_2))$, and $\psi' = (\forall Y_1 : \Box\phi_1 \rightarrow \Box in(\phi_2))$. Then, if $\Box(in(\phi_1))$ is True, we have $\psi \leftrightarrow \psi'$.*

Proof (Lemma 3.4). Suppose first that ψ is *True*, and suppose that on all sequences $y_{1,n}$ of valuations over Y_1, $\Box\phi_1$ 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 ψ, we also have that for all n, for all the valuations over (X_1, X_2, Y_1), $(x_{1,n}, x_{2,n}, y_{1,n}) \models in(\phi_2)$. This implies that $\Box in(\phi_2)$ is also valid for all sequences of valuations over Y_1, and ψ' is *True*. Therefore, we conclude that $\psi \rightarrow \psi'$.

To prove that $\psi' \rightarrow \psi$, we now assume that ψ is *False*, and prove that ψ' must also be *False*. In fact, if ψ 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 $(x_{1,i}, x_{2,i}, y^*) \not\models in(\phi_2)$. Consider such a sequence $(x_{1,k}, x_{2,k})$. Then, since $\Box in(\phi_1)$ holds by hypothesis, we know that, for all k, it is possible to find $\hat{y}_{1,k}$ such that $(x_{1,k}, \hat{y}_{1,k}) \models \phi_1$. 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 $\forall k \neq i$, $y_{1,k} = \hat{y}_{1,k}$, and for $k = i$, $y_{1,i} = y^*$. By construction, $s_k \models \Box\phi_1$ but $s_k \not\models \Box in(\phi_2)$, i.e. s_k falsifies ψ'. We can therefore conclude $\neg\psi \rightarrow \neg\psi'$, which is what we wanted to prove. □

[1] $\Box$ takes precedence over $\rightarrow$, so $\Box in(\phi) \rightarrow \Box\phi$ means $\big(\Box in(\phi)\big) \rightarrow \Box\phi$.

We can now prove Theorem 3.3.

PROOF (THEOREM 3.3). Both the left and right-hand side contracts C_L and C_R in (3) are in saturated form by definition of $\mathcal{F}$ 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 (1) and (2) and the definition of $\mathcal{F}$ we obtain:

$$G_\otimes = (\Box in(\phi_1) \rightarrow \Box\phi_1) \wedge (\Box in(\phi_2) \rightarrow \Box\phi_2) \tag{4}$$

$$\begin{aligned} A_\otimes &= (\Box in(\phi_1) \wedge \Box in(\phi_2)) \vee \neg G_\otimes \\ &= \Box(in(\phi_1) \wedge in(\phi_2)) \vee (\Box in(\phi_1) \wedge \neg\Box\phi_1) \\ &\quad \vee (\Box in(\phi_2) \wedge \neg\Box\phi_2) \end{aligned} \tag{5}$$

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

$$\begin{aligned} A_R &= \forall Y_1 \forall Y_2 : A_\otimes \\ &= \forall Y_1 : \Box(in(\phi_1) \wedge in(\phi_2)) \vee (\Box in(\phi_1) \wedge \neg\Box\phi_1) \\ &\qquad \vee (\forall Y_2 : (\Box in(\phi_2) \wedge \neg\Box\phi_2)) \\ &= \forall Y_1 : \Box(in(\phi_1) \wedge in(\phi_2)) \vee (\Box in(\phi_1) \wedge \neg\Box\phi_1) \\ &= \forall Y_1 : \Box in(\phi_1) \wedge (\Box in(\phi_2) \vee \neg\Box\phi_1) \\ &= \Box in(\phi_1) \wedge (\forall Y_1 : \Box\phi_1 \rightarrow \Box in(\phi_2)) \end{aligned} \tag{6}$$

$$\begin{aligned} G_R &= A_R \rightarrow G_\otimes \\ &= \Box in(\phi_1) \wedge (\forall Y_1 : \Box\phi_1 \rightarrow \Box in(\phi_2)) \\ &\quad \rightarrow (\Box\phi_1 \vee \neg\Box in(\phi_1)) \wedge (\Box\phi_2 \vee \neg\Box in(\phi_2)) \end{aligned} \tag{7}$$

Consider now the assumptions of C_L. We obtain:

$$\begin{aligned} A_L = \Box in(\phi) &= \Box\left[\exists Y_1 \exists Y_2 : \phi_1 \wedge \phi_2 \wedge (\forall Y_1 : \phi_1 \rightarrow in(\phi_2))\right] \\ &= \Box\left[(\forall Y_1 : \phi_1 \rightarrow in(\phi_2)) \wedge (\exists Y_1 : \phi_1 \wedge in(\phi_2))\right] \\ &= \Box\left[(\forall Y_1 : \phi_1 \rightarrow in(\phi_2)) \wedge in(\phi_1)\right] \\ &= \Box(\forall Y_1 : \phi_1 \rightarrow in(\phi_2)) \wedge \Box in(\phi_1) \end{aligned} \tag{8}$$

while for G_L we obtain

$$\begin{aligned} G_L = \Box(\forall Y_1 : \phi_1 \rightarrow in(\phi_2)) \wedge \Box in(\phi_1) \\ \rightarrow \Box(\phi_1 \wedge \phi_2 \wedge (\forall Y_1 : \phi_1 \rightarrow in(\phi_2))). \end{aligned} \tag{9}$$

The equivalence of the assumptions A_L and A_R directly descends from Lemma 3.4. 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

$$(\Box in(\phi_1) \rightarrow \Box\phi_1) \wedge (\Box in(\phi_2) \rightarrow \Box\phi_2) \leftrightarrow \Box(\phi_1 \wedge \phi_2). \tag{10}$$

Clearly, if the formula on the left side of the double implication in (10) 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 (9) 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

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

This allows us to conclude that $\Box\phi_2$ is also *True* and finally (10) holds. We have therefore proved (3).

Let now $\phi = \phi_1 \wedge \phi_2 \wedge (\forall Y_1 : \phi_1 \rightarrow in(\phi_2))$ be the formula associated with $I_1 \rightsquigarrow I_2$. I_1 and I_2 are compatible if and only if ϕ is satisfiable. On the other hand, $\mathrm{AP}_{Y_1 \cup Y_2}(\mathcal{F}(I_1) \otimes \mathcal{F}(I_2))$ 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 ϕ is satisfiable if and only if A_R is satisfiable. This can be directly inferred from the fact that $A_R = A_L = \Box in(\phi)$. In fact, $\Box in(\phi)$ is satisfiable if and only if $in(\phi)$ is satisfiable, i.e. if and only if ϕ is satisfiable, which concludes our proof. □

Assumption projection hides the controlled variables of the 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 [12]. For instance, consider the LTL formula ϕ over two Boolean variables s and p:

$$\phi := p \wedge \Box(s \rightarrow p) \wedge \Box(s \rightarrow \bigcirc\neg s) \wedge \Box(\neg s \rightarrow \bigcirc s)$$

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

3.2 Refinement

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

Theorem 3.5. *Given two relational interfaces I_1 and I_2, then $I_1 \sqsubseteq I_2$ if and only if $\mathcal{F}(I_1) \preceq \mathcal{F}(I_2)$.*

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 $(in(\phi_2) \rightarrow in(\phi_1) \wedge (\phi_1 \rightarrow \phi_2))$ is valid or, equivalently, the following two formulas

$$in(\phi_2) \rightarrow in(\phi_1) \tag{11}$$

$$in(\phi_2) \wedge \phi_1 \rightarrow \phi_2 \tag{12}$$

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

$$\mathcal{F}(I_1) = (Y_1 \cup X_2, \Box in(\phi_1), \Box in(\phi_1) \rightarrow \Box\phi_1)$$

$$\mathcal{F}(I_2) = (Y_1 \cup X_2, \Box in(\phi_2), \Box in(\phi_2) \rightarrow \Box\phi_2).$$

We first prove that $I_1 \sqsubseteq I_2 \rightarrow \mathcal{F}(I_1) \preceq \mathcal{F}(I_2)$. Let A_i and G_i be, respectively, the assumptions and the guarantees of $\mathcal{F}(I_i)$. We need to show that formulas (11) and (12) imply $A_2 \rightarrow A_1$ and $G_1 \rightarrow G_2$. Assume $A_2 = \Box in(\phi_2)$ is *True*, then, by (11), $A_1 = \Box in(\phi_1)$ is also *True*; therefore, $A_2 \rightarrow A_1$. Assume now that G_1 is *True*, i.e. either $\Box in(\phi_1)$ is *False* or $\Box\phi_1$ is *True*. If $\Box in(\phi_1)$ is *False*, then from $A_2 \rightarrow A_1$, $\Box in(\phi_2)$ is also *False*, which makes G_2 *True*. If $\Box\phi_1$ is *True*, then, by (12), we conclude $\Box in(\phi_2) \rightarrow \Box\phi_2$, hence G_2 is again *True*. We therefore conclude that $G_1 \rightarrow G_2$.

We now prove that if $\mathcal{F}(I_1) \preceq \mathcal{F}(I_2)$, i.e. $A_2 \rightarrow A_1$ and $G_1 \rightarrow G_2$, then (11) and (12) are valid. To do so, we assume instead that $I_1 \not\sqsubseteq I_2$ and show that $\mathcal{F}(I_1) \not\preceq \mathcal{F}(I_2)$. In fact, if (11) is not valid, then we can create a sequence x_n of valuations over X_2 and an index i such that $x_n \models in(\phi_2)$ for all n, and $x_i \not\models in(\phi_1)$. Then, for such a sequence, $\Box in(\phi_2)$ is *True* while $\Box in(\phi_1)$ is *False*, which means that $A_2 \rightarrow A_1$ is not valid. Similarly, assume (12) 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 $(x_n, y_n) \models in(\phi_2)$ and $(x_n, y_n) \models \phi_1$ for all n, while $(x_i, y_i) \not\models \phi_2$. However, this implies that $\Box\phi_1$, hence G_1 is *True* while G_2 is *False*, since $\Box in(\phi_2)$ is *True* without $\Box\phi_2$ being *True*. Therefore, $G_1 \rightarrow G_2$ is also not valid, which allows us to

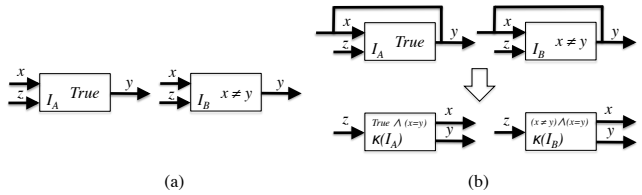

Figure 2: Configurations considered in Example 3.

conclude $(I_1 \not\sqsubseteq I_2) \rightarrow (\mathcal{F}(I_1) \not\preceq \mathcal{F}(I_2))$, 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]. However, this is not always the case for feedback composition. In relational interfaces, feedback preserves refinement only if the interfaces are "Moore" with respect to the input variables involved in the connection, i.e., when the fed-back output only depends on state variables but not on current inputs [4]. In A/G contracts, refinement is instead preserved by feedback composition [5].

An in-depth investigation of the properties of feedback composition is out of the scope of this paper. In what follows, we discuss just one property of interest. First, we provide a definition of feedback for A/G contracts.

Definition 3.6 (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 $C_{id} = (\{x, y\}, True, \Box(x = y))$. Then, κ defines a new contract $\kappa(C) := C \otimes C_{id}$.*

Theorem 3.7 (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*

$$(I_1 \sqsubseteq I_2) \rightarrow (\kappa(\mathcal{F}(I_1)) \preceq \kappa(\mathcal{F}(I_2)))\,, \tag{13}$$

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

Proof. By Theorem 3.5, we know that if $I_1 \sqsubseteq I_2$ then $\mathcal{F}(I_1) \preceq \mathcal{F}(I_2)$. By definition of κ, 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 $(\{x, y\}, True, \Box(x = y))$. Then, by Property 3 (independent implementability) of the parallel composition of contracts in [6], if $\kappa(\mathcal{F}(I_2))$ is compatible, we can conclude that $\kappa(\mathcal{F}(I_1))$ is also compatible and $\kappa(\mathcal{F}(I_1)) \preceq \kappa(\mathcal{F}(I_2))$, as we wanted to show. □

We observe that (13) holds even if I_1 and I_2 are not Moore with respect to x, in which case $\kappa(I_1) \sqsubseteq \kappa(I_2)$ is not guaranteed. As illustrated by the following two examples, $\kappa(I_1) \sqsubseteq \kappa(I_2)$ may not hold either because $\phi_{\kappa(I_1)}$ is *False* (Example 3) or because $(\phi_{\kappa(I_1)} \rightarrow \phi_{\kappa(I_2)})$ is *False* (Example 4).

Example 3. *Consider $I_A = (\{x, z\}, \{y\}, True)$ and $I_B = (\{x, z\}, \{y\}, x \neq y)$ as in Fig. 2 (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 $I_B \sqsubseteq I_A$ since $in(\phi_A) = in(\phi_B) = True$ and $\phi_B \rightarrow \phi_A$. However, given $\kappa(I_A) = (\{z\}, \{y, x\}, x = y)$ and $\kappa(I_B) = (\{z\}, \{y, x\}, False)$, obtained as shown in Fig. 2 (b), is clearly $\kappa(I_B) \not\sqsubseteq \kappa(I_A)$ since $\phi_{\kappa(I_B)}$ is False. Consider now the associated contracts $A = (V, True, True)$ and $B = (V, True, \Box(y \neq x))$ on variables $V = \{x, y, z\}$. We have $B \preceq A$, $\kappa(A) = (V, True, \Box(y = x))$, $\kappa(B) = (V, True, False)$, and $\kappa(B) \preceq \kappa(A)$. Therefore, refinement is preserved by feedback composition, even if $\kappa(B)$ is inconsistent.*

Example 4. *Consider now $I_A = (\{x\}, \{y\}, (x \neq 0) \wedge (xy = 1))$ and $I_B = (\{x\}, \{y\}, (x \neq 0) \rightarrow (xy = 1))$. We have $I_B \sqsubseteq I_A$ since $in(\phi_A) = (x \neq 0)$, $in(\phi_B) = True$ and $(\phi_B \rightarrow \phi_A) = (x \neq 0)$. However, given $\kappa(I_A) = (\emptyset, \{y, x\}, (x^2 = 1) \wedge (x = y))$ and $\kappa(I_B) = (\emptyset, \{y, x\}, (x \neq 0 \rightarrow x^2 = 1) \wedge (x = y))$, we obtain $\kappa(I_B) \not\sqsubseteq \kappa(I_A)$. In fact, $in(\phi_{\kappa(I_B)}) = in(\phi_{\kappa(I_A)}) = True$; however, $\phi_{\kappa(I_B)} \rightarrow \phi_{\kappa(I_A)}$ is False. Consider now the associated contracts $A = (V, \Box(x \neq 0), \Box(x \neq 0) \rightarrow \Box(xy = 1))$ and $B = (V, True, \Box((x \neq 0) \rightarrow (xy = 1))$ on variables $V = \{x, y\}$. Since $\kappa(A) = (V, \Box(x \neq 0) \vee \neg\Box(y = x), (\Box(x \neq 0) \rightarrow \Box(x^2 = 1)) \wedge \Box(y = x))$ is compatible, $\kappa(B)$ is also compatible and $\kappa(B) \preceq \kappa(A)$. In this case, however, $\kappa(B)$ is also consistent.*

3.3 Conjunction

Even if $\mathcal{F}$ preserves refinement, it does not preserve conjunction. First, conjunction (shared refinement) is not always defined for relational interfaces. For A/G contracts, conjunction can always be defined as the GLB of the refinement relation, but it can still generate inconsistent contracts, as illustrated by the following example.

Example 5. *Consider $I_{00} = (\{x\}, \{y\}, x = 0 \rightarrow y = 0)$ and $I_{01} = (\{x\}, \{y\}, x = 0 \rightarrow y = 1)$. As discussed in [4], they are not shared refinable, since it is not possible to guarantee $y = 0$ and $y = 1$ at the same time. However, conjunction can still be defined for their associated contracts $\mathcal{F}(I_{00}) = (\{x, y\}, True, \Box(x = 0 \rightarrow y = 0))$ and $\mathcal{F}(I_{01}) = (\{x, y\}, True, \Box(x = 0 \rightarrow y = 1))$, although it only generates the inconsistent contract $(\{x, y\}, True, False)$.*

When conjunction is well-defined in both frameworks, 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 3.8. *Let $I = (X, Y, \phi)$ and $I' = (X, Y, \phi')$ be two* shared-refinable *relational interfaces. Then we have*

$$\mathcal{F}(I \sqcap I') \preceq \mathcal{F}(I) \wedge \mathcal{F}(I'), \tag{14}$$

with $\mathcal{F}(I \sqcap I') \neq \mathcal{F}(I) \wedge \mathcal{F}(I')$ in general.

Proof. We recall that $I \sqcap I' = (X, Y, \phi_\sqcap)$, where

$$\phi_\sqcap = (in(\phi) \vee in(\phi')) \wedge (in(\phi) \rightarrow \phi) \wedge (in(\phi') \rightarrow \phi'),$$

where $in(\phi_\sqcap) = in(\phi) \vee in(\phi')$ by Lemma 8 in [4]. Therefore, by transforming $I \sqcap I'$, we obtain $\mathcal{F}(I \sqcap I') = (X \cup Y, A_\sqcap, G_\sqcap)$, where

$$A_\sqcap = \Box(in(\phi) \vee in(\phi'))$$

and

$$G_\sqcap = \Box(in(\phi) \vee in(\phi')) \rightarrow (\Box(in(\phi) \rightarrow \phi) \wedge \Box(in(\phi') \rightarrow \phi')).$$

Moreover, by definition of conjunction, we obtain $\mathcal{F}(I) \wedge \mathcal{F}(I') = (X \cup Y, A_\wedge, G_\wedge)$, where

$$A_\wedge = \Box in(\phi) \vee \Box in(\phi')$$

and

$$G_\wedge = (\Box in(\phi) \rightarrow \Box\phi) \wedge (\Box in(\phi') \rightarrow \Box\phi').$$

It is straightforward to see that $A_\wedge \rightarrow A_\sqcap$. On the other hand, we also notice that $A_\sqcap \not\rightarrow A_\wedge$. In fact, any sequence x_n such that $x_1 \models in(\phi)$, $x_1 \not\models in(\phi')$, and $x_n \models in(\phi')$ for all $n > 1$, satisfies $A_\wedge$ but does not satisfy $A_\sqcap$.

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

Therefore, the contract associated with the shared refinement of I and I' is indeed a refinement of the conjunction of the contracts associated with I and I', and equality does not generally hold. □

4. 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 design specification, early detection of incompatibilities, and principled use of abstraction-refinement. 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 relational interfaces, which requires extending the latter with liveness properties. Finally, the implementation of the assumption-projection operator on LTL contracts will also be considered as future work.

Acknowledgements

This work was partially supported by IBM and United Technologies Corporation (UTC) via the iCyPhy consortium, by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, by the Academy of Finland, and by the National Science Foundation, via projects "ExCAPE: Expeditions in Computer Augmented Program Engineering" and "COSMOI: Compositional System Modeling with Interfaces."

5. REFERENCES

[1] L. de Alfaro and T. Henzinger, "Interface theories for component-based design," in *EMSOFT'01*. Springer, LNCS 2211, 2001.

[2] A. Sangiovanni-Vincentelli *et al.*, "Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems," *European Journal of Control*, vol. 18-3, no. 3, pp. 217–238, 2012.

[3] L. de Alfaro and T. Henzinger, "Interface automata," in *FSE*. ACM, 2001.

[4] S. Tripakis, B. Lickly, T. A. Henzinger, and E. A. Lee, "A theory of synchronous relational interfaces," *ACM TOPLAS*, vol. 33, no. 4, 2011.

[5] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis, "Multiple viewpoint contract-based specification and design," in *FMCO*, 2008.

[6] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier *et al.*, "Contracts for System Design," INRIA, Rapport de recherche RR-8147, Nov. 2012.

[7] S. S. Bauer, A. David, R. Hennicker, K. G. Larsen, A. Legay *et al.*, "Moving from specifications to contracts in component-based design," in *FASE*, 2012, pp. 43–58.

[8] S. Quinton, S. Graf, and R. Passerone, "Contract-based reasoning for component systems with complex interactions," Verimag Research Report, Tech. Rep. TR-2010-12, 2010.

[9] L. Doyen, T. A. Henzinger, B. Jobstmann, and T. Petrov, "Interface theories with component reuse," in *EMSOFT*, 2008, pp. 79–88.

[10] T. A. Henzinger and D. Nickovic, "Independent implementability of viewpoints," in *Monterey Workshop*. Springer, 2012, pp. 380–395.

[11] A. Pnueli, "The temporal logic of programs," in *Foundations of Computer Science*, 1977, pp. 46–57.

[12] P. Wolper, "Temporal logic can be more expressive," in *Foundations of Computer Science*, 1981, pp. 340–348.