On Relational Interfaces*

Stavros Tripakis
UC Berkeley

Ben Lickly
UC Berkeley

Thomas A. Henzinger
EPFL and IST Austria

Edward A. Lee
UC Berkeley

ABSTRACT

In this paper we extend the work of Alfaro, Henzinger et al. on interface theories for component-based design. Existing interface theories often fail to capture functional relations between the inputs and outputs of an interface. For example, a simple synchronous interface that takes as input a number $n \geq 0$ and returns, at the same time, as output $n + 1$, cannot be expressed in existing theories. In this paper we provide a theory of relational interfaces, where such input-output relations can be captured. Our theory supports synchronous interfaces, both stateless and stateful. It includes explicit notions of environments and pluggability, and satisfies fundamental properties such as preservation of refinement by composition, and characterization of pluggability by refinement. We achieve these properties by making reasonable restrictions on feedback loops in interface compositions.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Modules and interfaces*; D.2.12 [**Software Engineering**]: Interoperability—*Interface definition languages*

General Terms

Design, Languages, Theory, Verification

Keywords

Compositionality, Interfaces, Composition, Refinement

*This work is supported by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS: PRET) and #0720841 (CSR-CPS)), the U.S. Army Research Office (ARO #W911NF-07-2-0019), the U.S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force Research Lab (AFRL), the State of California Micro Program, and the following companies: Agilent, Bosch, Lockheed-Martin, National Instruments, Thales and Toyota. This work is also supported by the COMBEST and ArtistDesign projects of the European Union, and the Swiss National Science Foundation. Authors' emails: {stavros,blickly,eal}@eecs.berkeley.edu, tah@epfl.ch. Corresponding author's address: Stavros Tripakis, 545Q, DOP Center, Cory Hall, EECS Department, University of California, Berkeley, CA 94720-1772, USA.

EMSOFT'09, October 12–16, 2009, Grenoble, France.

1. INTRODUCTION

Component-based design has emerged as a significant challenge in building complex systems, such as embedded, cyber-physical systems, in an efficient and reliable manner. The size and complexity of such systems prohibit designing an entire system from scratch, or building it as a single unit. Instead, the system must be designed as a set of components, some built from scratch, some taken off the shelf, some inherited by legacy. *Interfaces* play a key role in component-based design, as they provide the means to reason about components. An interface can be seen as an abstraction of a component: on one hand, such an abstraction captures information that is essential in order to use the component in a given context; on the other hand, the abstraction hides unnecessary information, making reasoning simpler and more efficient.

Significant progress has been made in the past several years toward the development of a comprehensive theory of interfaces for component-based design. Such a theory has been pioneered and developed in a series of papers by Alfaro, Henzinger et al. (see [6, 5, 4, 7] for a sample). What has been elusive, however, is a theory of *relational interfaces*, that is, interfaces that specify relations between inputs and outputs. Consider, for example, a synchronous component that is supposed to take as input a number $n \geq 0$ and return as output $n + 1$, in the same synchronous round. The interface for such a component can be described as a binary relation between the input and the output: the relation containing all pairs $(n, n + 1)$, such that $n \geq 0$. Such a relation can be seen as a *contract* between the component and its environment: the contract specifies the *legal* inputs that the environment is allowed to provide to the component (in this case $n \geq 0$); and for every legal input, what are the legal outputs that the component may produce when fed with that input.

Existing interface theories, in particular those proposed in the aforementioned works, only partially capture relational interfaces. [5] defines *interface automata*, which can be used to capture input-output relations in an asynchronous concurrency model, and in a mostly operational manner. In a more denotational setting, [6] defines *relational nets*, which

are networks of processes that non-deterministically relate input values to output values. [6] does not provide an interface theory for the complete class of relational nets. Instead it provides interface theories for subclasses, in particular: *rectangular nets* which have no input-output dependencies; *total nets* which can have input-output dependencies but make no restrictions on the inputs (in this paper we call such interfaces *input-complete*); and *total and rectangular nets* which combine both restrictions above.

The interfaces provided in [6] for rectangular nets are called *assume/guarantee (A/G) interfaces*. [6] studies *stateless* A/G interfaces, while [7] studies also *stateful* A/G interfaces, in a synchronous setting similar to the one considered in this paper. A/G interfaces separate the assumptions on the inputs from the guarantees on the outputs, and as such cannot capture input-output relations. [7] also discusses *extended interfaces* which are essentially the same as the relational interfaces that we study in this paper. However, difficulties with synchronous feedback loops (see discussion below) lead [7] to conclude that extended interfaces are not appropriate.

[4] considers synchronous *Moore interfaces*, defined by a formula ϕ_i that specifies the legal values of the input variables at the *next* state, given the current state, and a formula ϕ_o that specifies the legal values of the output variables at the *next* state, given the current state. This formulation does not allow to describe relations between inputs and outputs at the *same* state, as our relational theory allows.

Both [6] and [7] can handle very general compositions of interfaces, that can be obtained via two operators, namely, *parallel composition* and *connection* (this is similar to the denotational framework of [8]). This allows, in particular, arbitrary *feedback loops* to be created.[1] Synchronous feedback loops are a major source of problems when studying relational interfaces. To illustrate some of the problems that arise, consider the following example, borrowed from [7]. Suppose I_{true} is an interface on input x and output y, with trivial contract true, making no assumptions on the inputs and no guarantees on the outputs. Suppose $I_{y\neq x}$ is another interface on x and y, with contract $y \neq x$, meaning that it guarantees the value of the output will be different from the value of the input. According to standard definitions of refinement, $I_{y\neq x}$ refines I_{true}: this is because $I_{y\neq x}$ is "more deterministic" than I_{true} (the output guarantees of $I_{y\neq x}$ are stronger). Now, consider the feedback connection $x = y$. This could be considered an allowed connection for I_{true}, since it does not contradict its contract. But the same connection contradicts the contract of $I_{y\neq x}$. As a result, even though $I_{y\neq x}$ refines I_{true}, the feedback composition of $I_{y\neq x}$ does not refine the feedback composition of I_{true}. This means that one of the fundamental properties that an interface theory should provide, namely, that composition preserves refinement, would not hold in this case.

In this paper we propose a theory of relational interfaces. Our theory relies on a notion of refinement which is *input-contravariant* like the relations proposed in [2, 6, 7], but not strictly *output-covariant*. We avoid problems created by feedback loops by restricting the cases in which feedback loops are allowed. In particular, we allow an output of an interface I to be connected to one of its inputs x only if I is *Moore with respect to* x, meaning that the contract of I does not depend on x.

Arguably, this is a reasonable restriction in practice. After all, arbitrary feedback loops in synchronous models generally create *causality cycles* that result in ambiguous semantics. In many languages and tools these problems are avoided by making restrictions similar to (in fact, stricter than) ours. For example, tools such as Simulink from The MathWorks [2] or SCADE from Esterel Technologies [3] often require a *unit-delay* block to be placed in every feedback loop.[4] This restriction does not appear to result in a significant loss of expressiveness in practice. Similar restrictions are used in synchronous formalisms such as Lustre [3] or *reactive modules* [1].

Using the above restriction, we are able to derive a comprehensive theory of relational interfaces that supports *substitutability* (if I' refines I then I' can replace I in any context), and preservation of refinement by composition (if components I'_i refine components I_i then the composition of I'_i refines the composition of I_i). Other properties of our theory and contributions of this paper include the following:

We explicitly introduce the notions of *environments* and *pluggability* of interfaces to environments. We also introduce two notions of *equivalence* between interfaces: equivalence of their contracts, and equivalence with respect to environments (two interfaces are equivalent iff they can be plugged to the same set of environments). We show that these two equivalences coincide. We also prove that our notion of refinement characterizes pluggability in the following fundamental way: interface I' refines I iff I' can replace I in every context (i.e., environment). (Note that this is a stronger property than stepwise refinement: it is an *iff* instead of an *only if*.) We show by example that alternative definitions of refinement do not have this property.

We seamlessly handle stateless and stateful interfaces. Stateful interfaces associate a potentially different contract at every state. We model a state very simply and generally, as a history of input/output values. This allows us to handle finite as well as infinite-state interfaces. Stateless interfaces can be viewed as a special case of stateful interfaces where the contract is the same at all states.

In the stateful case, we distinguish between *well-formed* and *well-formable* interfaces (the two notions coincide for stateless interfaces). Well-formed interfaces are such that their contract can always be satisfied at every reachable state. Well-formable interfaces are not necessarily well-formed, but can be made well-formed by appropriately restricting their inputs. Refinement preserves well-formability always, but it only preserves well-formedness under certain conditions. As in [6], *controller-synthesis* type of procedures can be used to transform finite-state well-formable interfaces into well-formed ones. We should note that well-formability has been implicitly used in previous works [6, 5, 7] as a condition for interface *compatibility*, in the sense that these works require the composition to be well-formable, in order for it to be defined. Here, we take a different approach, and define composition irrespectively of its well-formability.

[1] A feedback loop arises when the dependencies induced by a set of connected interfaces contain a cycle, as in the example shown in Figure 2.

[2] See `http://www.mathworks.com/products/simulink/`.

[3] See `http://www.esterel-technologies.com/products/scade-suite/`.

[4] Simulink provides the user with the option to ignore so-called *algebraic loops* but this results in ambiguous (non-deterministic) semantics.

This allows us to state more general results. In particular, preservation of refinement by connection (Theorem 12) holds independently of whether the connection yields a well-formable interface or not.

We propose a *hiding* operator that allows removal of a subset of the output variables in a contract by projecting them out. This is useful when composing interfaces, where often many variables end up being equal. Hiding is always possible for stateless interfaces, and corresponds to existentially quantifying outputs in the contract. The situation is more subtle in the stateful case, where we need to ensure that the "hidden" variables do not influence the evolution of the contract.

Our theory supports *shared refinement* of two interfaces I and I', which is important for component reuse, as argued in [7]. A shared refinement operator $I \sqcap I'$ is proposed in the discussion section of [7] for extended (i.e., relational) interfaces, and it is conjectured that this operator represents the greatest lower bound with respect to refinement. We show that this holds only if a *shared refinability* condition is imposed. This condition states that for every inputs that is legal in both I and I', the corresponding sets of outputs of I and I' must have a non-empty intersection.

2. PRELIMINARIES, NOTATION

In this paper we use first-order logic (FOL) as a language to describe contracts.[5] For an introduction to FOL, see, for instance, [10]. We use true and false for logical constants true and false, $\neg, \wedge, \vee, \rightarrow, \equiv$ for logical negation, conjunction, disjunction, implication, and equivalence, and $\exists$ and $\forall$ for existential and universal quantification, respectively. We will use $:=$ when defining concepts or introducing new notation, for instance, $x_0 := \max\{1,2,3\}$ defines x_0 to be the maximum of the set $\{1,2,3\}$.

Let V be a finite set of variables. A *property over* V is a FOL formula ϕ such that any free variable of ϕ is in V. The set of all properties over V is denoted $\mathcal{F}(V)$. Let ϕ be a property over V and V' be a finite subset of V, $V' = \{v_1, v_2, ..., v_n\}$. Then, $\exists V' : \phi$ is shorthand for $\exists v_1 : \exists v_2 : ... : \exists v_n : \phi$. Similarly, $\forall V' : \phi$ is shorthand for $\forall v_1 : \forall v_2 : ... : \forall v_n : \phi$.

We will implicitly assume that all variables are *typed*, meaning that every variable is associated with a certain *domain*. An *assignment* over a set of variables V is a (total) function mapping every variable in V to a certain value in the domain of that variable. The set of all assignments over V is denoted $\mathcal{A}(V)$. If a is an assignment over V_1 and b is an assignment over V_2, and V_1, V_2 are disjoint, we use (a,b) to denote the combined assignment over $V_1 \cup V_2$. A formula ϕ is *satisfiable* iff there exists an assignment a over the free variables of ϕ such that a satisfies ϕ, denoted $a \models \phi$. A formula ϕ is *valid* iff it is satisfied by every assignment.

If S is a set, S^* denotes the set of all finite sequences made up of elements in S. S^* includes the empty sequence, denoted ε. If $s, s' \in S^*$, then $s \cdot s'$ is the concatenation of s and s'. $|s|$ denotes the *length* of $s \in S^*$, with $|\varepsilon| = 0$ and $|s \cdot a| = |s| + 1$, for $a \in S$. If $s = a_1 a_2 \cdots a_n$, then the i-th element of the sequence, a_i, is denoted s_i, for $i = 1, ..., n$.

[5] As mentioned in the introduction, contracts are essentially relations between inputs and outputs. Our theory holds for such relations, independently from how the relations are specified. FOL formulae is one possible language, but other languages could be used as well.

3. RELATIONAL INTERFACES

Definition 1 (Relational interface). *A* relational interface *(or simply* interface*) is a tuple $I = (X, Y, \xi)$ where X and Y are two finite and disjoint sets of* input *and* output variables*, respectively, and ξ is a total function*

$$\xi : \mathcal{A}(X \cup Y)^* \rightarrow \mathcal{F}(X \cup Y)$$

Recall that $\mathcal{A}(V)$ is the set of all assignments over set of variables V. Therefore, $\mathcal{A}(X \cup Y)^*$ is the set of all finite sequences of assignments over $X \cup Y$. Note that we allow X or Y to be empty: if X is empty then I is a *source* interface; if Y is empty then I is a *sink*. An element of $\mathcal{A}(X \cup Y)^*$ is called a *state*. The *initial state* is the empty sequence ε. Recall that $\mathcal{F}(X \cup Y)$ is the set of all properties over $X \cup Y$. Therefore, ξ associates with every state s a formula $\xi(s)$ over $X \cup Y$. This formula acts as the contract between I and its environment *at that state*. The contract changes dynamically, as the state of I changes.

Definition 2 (Assumptions, guarantees). *Given a contract $\phi \in \mathcal{F}(X \cup Y)$, the* input assumption *of ϕ is the formula $\mathsf{in}(\phi) := \exists Y : \phi$. The* output guarantee *of ϕ is the formula $\mathsf{out}(\phi) := \exists X : \phi$. Note that $\mathsf{in}(\phi)$ is a property over X and $\mathsf{out}(\phi)$ is a property over Y. Also note that $\phi \rightarrow \mathsf{in}(\phi)$ and $\phi \rightarrow \mathsf{out}(\phi)$ are valid formulae for any ϕ.*

Intuitively, an interface operates in a *synchronous* manner, in an infinite sequence of synchronous *rounds*. Suppose that at the beginning of a given round the state of I is s. The environment presents I with an assignment a_X over the input variables X, such that a_X satisfies the input assumptions $\mathsf{in}(\xi(s))$. I then chooses an assignment a_Y over the output variables Y, such that together the two assignments satisfy $\xi(s)$. The combined assignments yield an assignment over $X \cup Y$, $a := (a_X, a_Y)$. At the end of the round, the new state of I is $s \cdot a$. See also Definition 8 that follows.

Definition 3 (Stateless interface). *An interface $I = (X, Y, \xi)$ is* stateless *if for all $s, s' \in \mathcal{A}(X \cup Y)^*$, $\xi(s) = \xi(s')$.*

That is, a *stateless* interface is one where the contract is independent of the state. For a stateless interface, we can treat ξ as a property, instead of as a function that maps states to properties. For clarify, we will write $I = (X, Y, \phi)$ for a stateless interface, where ϕ is a property over $X \cup Y$. We also write $\mathsf{in}(I), \mathsf{out}(I)$ instead of $\mathsf{in}(\phi), \mathsf{out}(\phi)$.

Example 1. *Consider a component which is supposed to take as input a positive number n and return n or $n+1$ as output. We can capture such a component in different ways. One way is by the following stateless interface:*

$$I_1 := (\{x\}, \{y\}, x > 0 \wedge (y = x \vee y = x + 1)\}).$$

Here, x is the input variable and y is the output variable. The contract of I_1 explicitly forbids zero or negative values for x. Indeed, we have $\mathsf{in}(I_1) \equiv x > 0$.

Another possible stateless interface for this component is:

$$I_2 := (\{x\}, \{y\}, x > 0 \rightarrow (y = x \vee y = x + 1)\}).$$

The contract of I_2 is different from that of I_1: it allows $x \leq 0$, but makes no guarantees about the output y in that case. I_2 is an input-complete *interface, in the sense that it accepts all inputs. Indeed, we have $\mathsf{in}(I_2) \equiv \mathsf{true}$. Input-complete interfaces are discussed in detail in Section 9.*

In general, the state space of an interface is infinite. In some cases, however, only a finite set of states is needed to specify ξ. For example ξ may be specified by a finite-state automaton, as in [7]. Every state of the automaton is labeled with a contract. Every transition of the automaton is labeled with a *guard*, i.e., a condition on the input and output variables. Outgoing transitions from a state must have disjoint guards (for determinism) and the union of such guards must be true (for absence of deadlocks). An interface that can be specified as a finite-state automaton is called a *finite-state interface*.

The A/G interfaces considered in [5, 7] are a special case of the relational interfaces that we consider in this paper. A stateless A/G interface is a tuple (X, Y, ϕ_X, ϕ_Y), where ϕ_X is a property on X representing the input assumptions and ϕ_Y is a property on Y representing the output guarantees. This interface can simply be represented as the relational interface $(X, Y, \phi_X \wedge \phi_Y)$.

Definition 1 allows for the contract $\xi(s)$ at a certain state s to be an unsatisfiable property. On the other hand, not all such states may generally be *reachable*, because not all inputs or outputs are legal. We only care about states with unsatisfiable contracts when these states are reachable. Let us define reachable states formally.

A *run* of I is a state $s = a_1 \cdots a_k$, with $k \geq 0$ (if $k = 0$ then $s = \varepsilon$), such that $\forall i \in \{1, ..., k\} : a_i \models \xi(a_1 \cdots a_{i-1})$. A state is *reachable* iff it is a run. The set of reachable states of I is denoted $\mathcal{R}(I)$. By definition, $\varepsilon \in \mathcal{R}(I)$, for any I.

Definition 4 (Well-formed interface). *An interface $I = (X, Y, \xi)$ is* well-formed *iff for all $s \in \mathcal{R}(I)$, $\xi(s)$ is satisfiable.*

Example 2. *Let $I := (\{x\}, \{y\}, \xi)$ where x, y are implicitly considered to be Booleans, and $\xi(\varepsilon) :=$ true, $\xi((x, \_) \cdot s) :=$ false, $\xi((\neg x, \_) \cdot s) :=$ true, for all s. $(x, \_)$ denotes any assignment where x is* true *and $(\neg x, \_)$ denotes any assignment where x is* false*. I is not well-formed, because it has reachable states with contract* false *(all states starting with x being* true*). I can be transformed into a well-formed interface by restricting $\xi(\varepsilon)$ so that all reachable states with unsatisfiable contracts are avoided. In particular, setting $\xi(\varepsilon) := \neg x$, achieves this goal.*

Example 2 shows that some interfaces, even though they are not well-formed, can be turned into well-formed interfaces by appropriately restricting their inputs. This motivates the following definition:

Definition 5 (Well-formable interface). *An interface $I = (X, Y, \xi)$ is* well-formable *if there exists a well-formed interface $I' = (X, Y, \xi')$ such that: for all $s \in \mathcal{R}(I')$, $\xi'(s) \equiv \xi(s) \wedge \phi_s$, where ϕ_s is some property over X.*

Clearly, every well-formed interface is well-formable, but the opposite is not true as Example 2 shows. For stateless interfaces, the two notions coincide, however.

Theorem 1. *A stateless interface I is well-formed iff it is well-formable.*

Due to lack of space, proofs are omitted. We do include, however, the lemmata that are used in the proofs. The proofs can be found in the technical report version of this paper [11]. (Note that, unfortunately, the technical report version contains a typo in the statement of Theorem 11.)

For a finite-state interface, there exists a procedure to check whether it is well-formable, and if this is the case, transform it into a well-formed interface. Such a procedure essentially attempts to find a winning strategy in a *game*, as pointed out in [5]. Roughly speaking, the procedure consists in recursively marking states as *illegal*, until no more states can be marked. Initially, all states s such that $\xi(s)$ is unsatisfiable are marked as illegal. Then, repeatedly, a state s is marked illegal if there exists no *legal input assignment* at s. A legal input assignment at s is an assignment a to input variables, such that for any assignment b to output variables, if $(a, b) \models \xi(s)$ then the successor state $s \cdot (a, b)$ is legal. If at the end of this operation the initial state is marked illegal, then the interface is not well-formable, otherwise it is. During the procedure, the contract $\xi(s)$ of a legal state s can be restricted to allow only legal input assignments.

Definition 6 (Equivalence). *Interfaces $I = (X, Y, \xi)$ and $I' = (X', Y', \xi')$ are* equivalent*, denoted $I \equiv I'$, if $X = X'$, $Y = Y'$, and for all $s \in \mathcal{R}(I) \cap \mathcal{R}(I')$, the formula $\xi(s) \equiv \xi'(s)$ is valid.*

Lemma 1. *If $I \equiv I'$ then $\mathcal{R}(I) = \mathcal{R}(I')$.*

4. ENVIRONMENTS, PLUGGABILITY

Definition 7 (Environment). *An* environment *is a tuple $E = (X, Y, h_X, h_Y)$, where X and Y are as in Definition 1, and h_X, h_Y are total functions*

$$h_X : \mathcal{A}(X \cup Y)^* \to \mathcal{F}(X), \qquad h_Y : \mathcal{A}(X \cup Y)^* \to \mathcal{F}(Y)$$

h_X represents the guarantees on the inputs X that the environment *provides* at a given state. h_Y represents the guarantees that the environment *expects* on the outputs Y. See Definition 8 that follows.

States are defined for environments in the same way as for interfaces. A *stateless environment* is one where $h_X(s)$ and $h_Y(s)$ are constant for all states s. We define the set of reachable states of an environment E, denoted $\mathcal{R}(E)$, in a similar way as for interfaces: $\varepsilon \in \mathcal{R}(E)$, and $s \cdot a \in \mathcal{R}(E)$ iff $s \in \mathcal{R}(E)$ and $a = (a_X, a_Y)$ is an assignment such that $a_X \models h_X(s)$ and $a_Y \models h_Y(s)$. An environment E is said to be *live* if for all $s \in \mathcal{R}(E)$, both $h_X(s)$ and $h_Y(s)$ are satisfiable.

Definition 8 (Pluggability). *Interface $I = (X', Y', \xi)$ is* pluggable *to environment $E = (X, Y, h_X, h_Y)$, denoted $I \models E$, iff $X' = X$, $Y' = Y$, and for all $s \in \mathcal{R}(I_E)$, the following formula is valid:*

$$h_X(s) \to \big(\mathsf{in}(\xi(s)) \wedge (\xi(s) \to h_Y(s))\big) \tag{1}$$

where I_E is the interface defined as follows:

$$I_E := (X, Y, \xi_E) \tag{2}$$

$$\xi_E(s) := \xi(s) \wedge h_X(s), \text{ for any } s \in \mathcal{A}(X \cup Y)^* \tag{3}$$

Pluggability can be intuitively seen as a game between the interface and the environment [6]. Suppose s is the current state of I and E (initially, $s = \varepsilon$). If $h_X(s)$ is unsatisfiable, then E decides to stop the game. Otherwise, E chooses some input assignment a_X satisfying $h_X(s)$. If a_X violates

$\mathsf{in}(\xi(s))$, then Condition (1) is violated, and I is not pluggable to E: the "blame" here is on E, which provides too weak guarantees on the inputs. Otherwise, I chooses an output assignment a_Y such that the input-output assignment $a := (a_X, a_Y)$ satisfies $\xi(s)$. If a_Y violates $h_Y(s)$), then Condition (1) is violated, which again means I is not pluggable to E: in this case the "blame" is on I, which provides too weak guarantees on the outputs. Otherwise, a round is complete, and the new state (for both I and E) is $s{\cdot}a$. The game continues in the same manner.

EXAMPLE 3. *Consider stateless interfaces I_1 and I_2 from Example 1 and stateless environment $E_1 := (\{x\}, \{y\}, x > 0, y > 0)$. It can be seen that both I_1 and I_2 are pluggable to E_1. On the other hand, none of I_1, I_2 are pluggable to $E_2 := (\{x\}, \{y\}, x \geq 0, y > 0)$: the constraint $x \geq 0$ is not strong enough to meet the input assumption $x > 0$. Notice that I_2 does not explicitly impose this input assumption, however, it implicitly does, because it makes no guarantees on the outputs when $x > 0$ is violated. Finally, consider $E_3 := (\{x\}, \{y\}, \mathsf{true}, \mathsf{true})$. I_2 is pluggable to E_3: E_3 provides no guarantees on the inputs, but expects no guarantees on the outputs either. I_1 is not pluggable to E_3 because $\mathsf{true} \not\rightarrow x > 0$.*

The interface I_E defined by (2) and (3) is intended to capture the reachable states of the "closed-loop" composition of E and I. These reachable states are a subset of the reachable states of I, because the environment E may in general provide only a restricted set of inputs, among all possible legal inputs for I. The contract function ξ_E of I_E captures exactly that. The following lemma states that the reachable states of I_E are indeed a subset of those of I.

LEMMA 2. *Let I be an interface, E be an environment, and I_E be defined as in Definition 8. Then, $\mathcal{R}(I_E) \subseteq \mathcal{R}(I)$.*

LEMMA 3. *Let I, I' be interfaces, E be an environment, and I_E, I'_E be defined as in Definition 8. If $I \equiv I'$ then $\mathcal{R}(I_E) = \mathcal{R}(I'_E)$.*

THEOREM 2. *If an interface I is well-formable then there exists a live environment E such that $I \models E$.*

Since well-formed implies well-formable, a corollary of Theorem 2 is that every well-formed interface can be plugged to some live environment. Note that there exist non-well-formed interfaces that *can* nevertheless be plugged to live environments: these environments restrict the inputs, so states with unsatisfiable contracts are never reached. There exist also non-well-formable interfaces that can be plugged to non-live environments: these environments "stop" after some point, i.e., are such that $h_X(s) \equiv \mathsf{false}$ for some state s. Finally, there exist non-well-formed stateless interfaces which can be plugged into the trivial non-live environment that stops immediately (i.e., is such that $h_X(\varepsilon) \equiv \mathsf{false}$).

DEFINITION 9 (EQUIVALENCE W.R.T. ENVIRONMENTS). *Two interfaces I and I' are* equivalent w.r.t. environments, *denoted $I \equiv_e I'$, if for any environment E, I is pluggable to E iff I' is pluggable to E.*

THEOREM 3. *For any interfaces I, I', $I \equiv I'$ iff $I \equiv_e I'$.*

5. COMPOSITION

We define two types of composition. First, we can compose two interfaces I_1 and I_2 by *connecting* some of the output variables of I_1 to some of the input variables of I_2. One output can be connected to many inputs, but an input can be connected to at most one output. The connections define a new stateless interface. Thus, the composition process can be repeated to yield arbitrary (acyclic) interface diagrams. Composition by connection is associative (Theorem 5), so the order in which interfaces are composed does not matter.

Two interfaces $I = (X, Y, \xi)$ and $I' = (X', Y', \xi')$ are called *disjoint* if they have disjoint sets of input and output variables: $(X \cup Y) \cap (X' \cup Y') = \emptyset$.

DEFINITION 10 (COMPOSITION BY CONNECTION). *Let $I_i = (X_i, Y_i, \xi_i)$, for $i = 1, 2$, be two disjoint interfaces. A* connection θ between I_1, I_2, *is a finite set of pairs of variables, $\theta = \{(y_i, x_i) \mid i = 1, ..., m\}$, such that: (1) $\forall (y, x) \in \theta : y \in Y_1 \wedge x \in X_2$, and (2) $\forall (y, x), (y', x') \in \theta : x = x' \rightarrow y = y'$. The* external input and output variables *are the sets of variables $X_{\theta(I_1,I_2)}$ and $Y_{\theta(I_1,I_2)}$, respectively, defined as follows (where $X_\theta := \{x \mid \exists (y, x) \in \theta\}$):*

$$\begin{aligned} X_{\theta(I_1,I_2)} &:= (X_1 \cup X_2) \setminus X_\theta \\ Y_{\theta(I_1,I_2)} &:= Y_1 \cup Y_2 \cup X_\theta \end{aligned}$$

The connection θ defines the composite interface $\theta(I_1, I_2) := (X_{\theta(I_1,I_2)}, Y_{\theta(I_1,I_2)}, \xi)$, *where, for every $s \in \mathcal{A}(X_{\theta(I_1,I_2)} \cup Y_{\theta(I_1,I_2)})^*$:*

$$\begin{aligned} \xi(s) &:= \xi_1(s_1) \wedge \xi_2(s_2) \wedge \rho_\theta \wedge \forall Y_{\theta(I_1,I_2)} : \Phi \\ \Phi &:= (\xi_1(s_1) \wedge \rho_\theta) \rightarrow \mathsf{in}(\xi_2(s_2)) \\ \rho_\theta &:= \bigwedge_{(y,x)\in\theta} y = x \end{aligned} \tag{4}$$

and, for $i = 1, 2$, s_i is defined to be the projection of s to variables in $X_i \cup Y_i$.

Definition 10 may seem unnecessarily complex at first sight. In particular, the reader may doubt the necessity of the term $\forall Y_{\theta(I_1,I_2)} : \Phi$, in the definition of $\xi(s)$. Informally speaking, this term states that, no matter which outputs I_1 chooses to produce for a given input, all such outputs are legal inputs for I_2 (when connected). This condition is essential for the validity of our interface theory. Omitting this condition would result in a fundamental property of the theory, namely, preservation of refinement by composition (Theorem 12) not being true, as will be explained in Example 12.

Notice that, by definition of θ, $X_\theta \subseteq X_2$. This implies that $X_1 \subseteq X_{\theta(I_1,I_2)}$, that is, every input variable of I_1 is also an input variable of $\theta(I_1, I_2)$.

A connection θ is allowed to be empty. In that case, $\rho_\theta \equiv \mathsf{true}$, and the composition can be viewed as the *parallel composition* of two interfaces. If θ is empty, we write $I_1 \| I_2$ instead of $\theta(I_1, I_2)$. As may be expected, the contract of the parallel composition at a given global state is the conjunction of the original contracts at the corresponding local states, which implies that parallel composition is commutative:

LEMMA 4. *Consider two disjoint interfaces, $I_i = (X_i, Y_i, \xi_i)$, $i = 1, 2$. Then $I_1 \| I_2 = (X_1 \cup X_2, Y_1 \cup Y_2, \xi)$, where ξ is such that for all $s \in \mathcal{A}(X_1 \cup X_2 \cup Y_1 \cup Y_2)^*$,*

$\xi(s) \equiv \xi_1(s_1) \wedge \xi_2(s_2)$, *where, for* $i = 1, 2$, s_i *is the projection of* s *to* $X_i \cup Y_i$.

THEOREM 4 (PARALLEL COMPOSITION IS COMMUTATIVE). *For two disjoint interfaces* I_1 *and* I_2, $I_1 \| I_2 \equiv I_2 \| I_1$.

THEOREM 5 (CONNECTION IS ASSOCIATIVE). *Let* I_1, I_2, I_3 *be interfaces. Let* θ_{12} *be a connection between* I_1, I_2, θ_{13} *a connection between* I_1, I_3, *and* θ_{23} *a connection between* I_2, I_3. *Then:*

$$(\theta_{12} \cup \theta_{13})\,(I_1, \theta_{23}(I_2, I_3)) \equiv (\theta_{13} \cup \theta_{23})\,(\theta_{12}(I_1, I_2), I_3)$$

EXAMPLE 4. *Consider the diagram of stateless interfaces shown in Figure 1, where:*

$$\begin{array}{rcl} I_{id} & := & (\{x_1\}, \{y_1\}, y_1 = x_1) \\ I_{+1} & := & (\{x_2\}, \{y_2\}, y_2 = x_2 + 1) \\ I_{eq} & := & (\{z_1, z_2\}, \{\}, z_1 = z_2) \end{array}$$

This diagram can be viewed as the equivalent compositions

$$\theta_2\big(I_{+1}, \theta_1(I_{id}, I_{eq})\big) \equiv (\theta_1 \cup \theta_2)\big((I_{id} \| I_{+1}), I_{eq}\big)$$

where $\theta_1 := \{(y_1, z_1)\}$ *and* $\theta_2 := \{(y_2, z_2)\}$. *We proceed to compute the contract of the interface defined by the diagram. It is easier to consider the composition* $(\theta_1 \cup \theta_2)((I_{id} \| I_{+1}), I_{eq})$. *Define* $\theta_3 := \theta_1 \cup \theta_2$. *From Lemma 4 we get:*

$$I_{id} \| I_{+1} = (\{x_1, x_2\}, \{y_1, y_2\}, y_1 = x_1 \wedge y_2 = x_2 + 1)$$

Then, for $\theta_3((I_{id} \| I_{+1}), I_{eq})$, *Formula (4) gives:*

$$\Phi := (y_1 = x_1 \wedge y_2 = x_2 + 1 \wedge y_1 = z_1 \wedge y_2 = z_2) \rightarrow z_1 = z_2$$

By quantifier elimination, we get

$$\forall y_1, y_2, z_1, z_2 : \Phi \quad \equiv \quad x_1 = x_2 + 1$$

therefore

$$\begin{array}{rcl} \theta_3((I_{id} \| I_{+1}), I_{eq}) & = & (\{x_1, x_2\}, \{y_1, y_2, z_1, z_2\}, \\ & & y_1 = x_1 \wedge y_2 = x_2 + 1 \wedge z_1 = z_2 \\ & & \wedge\, y_1 = z_1 \wedge y_2 = z_2 \wedge x_1 = x_2 + 1) \end{array}$$

Notice that $\mathsf{in}(\theta_3((I_{id} \| I_{+1}), I_{eq})) \equiv x_1 = x_2 + 1$. *That is, because of the connection* θ, *new assumptions have been generated for the external inputs* x_1, x_2.

A composite interface is not guaranteed to be well-formed, even if its components are well-formed:

EXAMPLE 5. *Consider the composition* $\theta_3((I_{id} \| I_{+1}), I_{eq})$ *introduced in Example 4, and let* I_y *be the stateless interface defined as follows:*

$$I_y \quad := \quad (\{\}, \{y\}, \mathsf{true})$$

Let $\theta_4 := \{(y, x_1), (y, x_2)\}$. *That is, the output* y *of* I_y *is connected to both external inputs* x_1 *and* x_2 *of* $\theta_3((I_{id} \| I_{+1}), I_{eq})$. *The composite interface* $I_4 := \theta_4(I_y, \theta_3((I_{id} \| I_{+1}), I_{eq})$ *is not well-formed, even though both* I_y *and* $\theta_3((I_{id} \| I_{+1}), I_{eq})$ *are well-formed. This is because, for* I_4, *Formula (4) gives*

$$\Phi \quad := \quad (\mathsf{true} \wedge y = x_1 \wedge y = x_2) \rightarrow x_1 = x_2 + 1$$

therefore,

$$\forall x_1, x_2, y_1, y_2, z_1, z_2 : \Phi \quad \equiv \quad y = y + 1$$

Since the above formula is unsatisfiable, I_4 *is not well-formed.*

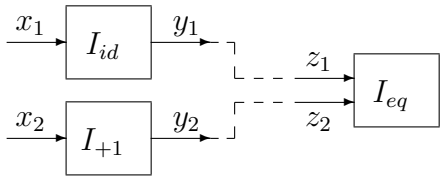

Figure 1: The interface diagram of Example 4.

Notice that all interfaces involved in Example 5 are stateless. Since well-formedness and well-formability coincide for stateless interfaces, Example 5 shows that connection does not preserve well-formability either: I_4 is not well-formable, even though both I_y and $\theta_3((I_{id} \| I_{+1}), I_{eq})$ are well-formed. Also note that, contrary to other works [6, 5, 7], we do not impose a *compatibility* condition on connections. We could easily impose well-formedness or well-formability as a compatibility condition. But we prefer not to do so, because this allows us to state more general results. In particular, Theorem 12 holds independently of whether the connection yields a well-formed interface or not. And together with Theorems 9 and 10, it guarantees that if the refined composite interface is well-formed/formable, then so is the refining one.

Connections can capture *cascade* composition, but not feedback. To capture feedback, we define a second type of composition, called *feedback composition*, where an output variable of an interface I is connected to one of its input variables x. For feedback, I is required to be *Moore with respect to* x. The term "Moore interfaces" has been introduced in [4]. Our definition is similar in spirit, but less restrictive than the one in [4]. Both definitions are inspired by Moore machines, where the outputs are determined by the current state alone and do not depend directly on the input. In our version, an interface is Moore with respect to a given input variable x, meaning that the contract may depend on the current state as well as on input variables other than x. This allows to connect an output to x to form a feedback loop without creating causality cycles.

DEFINITION 11 (MOORE INTERFACES). *An interface* $I = (X, Y, \xi)$ *is* Moore with respect to $x \in X$ *iff for all* $s \in \mathcal{R}(I)$, $\xi(s)$ *is a property over* $(X \cup Y) \setminus \{x\}$. I *is* Moore *when it is Moore with respect to every* $x \in X$.

EXAMPLE 6. *A* unit delay *is a basic building block in many modeling languages (including Simulink and SCADE). Its specification is roughly: "output at time* k *the value of the input at time* $k - 1$*; at time* $k = 0$ *(initial time), output some initial value* v_0*". We can capture this specification as a Moore interface (with respect to its unique input variable)* $I_{ud} := (\{x\}, \{y\}, \xi_{ud})$, *where* ξ_{ud} *is defined as follows:*

$$\begin{array}{rcl} \xi_{ud}(\varepsilon) & := & (y = v_0) \\ \xi_{ud}(s \cdot a) & := & (y = a(x)) \end{array}$$

That is, initially the contract guarantees $y = v_0$. *Then, when the state is some sequence* $s \cdot a$, *the contract guarantees* $y = a(x)$, *where* $a(x)$ *is the last value assigned to input* x.

DEFINITION 12 (COMPOSITION BY FEEDBACK). *Let* $I = (X, Y, \xi)$ *be a Moore interface with respect to some input port* $x \in X$. *A* feedback connection κ on I *is a pair* (y, x)

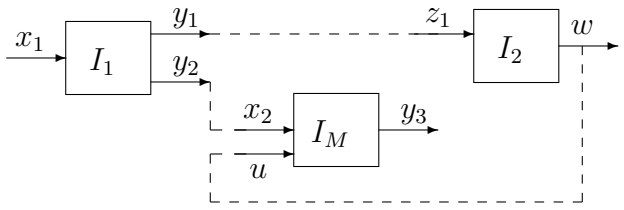

Figure 2: An interface diagram with feedback.

such that $y \in Y$. Define $\rho_\kappa := (x = y)$. The feedback connection κ defines the interface:

$$\kappa(I) := (X \setminus \{x\}, Y \cup \{x\}, \xi_\kappa) \quad (5)$$

$$\xi_\kappa(s) := \xi(s) \wedge \rho_\kappa, \quad \text{for all } s \in \mathcal{A}(X \cup Y)^* \quad (6)$$

THEOREM 6 (FEEDBACK IS COMMUTATIVE). *Let $I = (X, Y, \xi)$ be Moore with respect to both $x_1, x_2 \in X$, where $x_1 \neq x_2$. Let $\kappa_1 = (y_1, x_1)$ and $\kappa_2 = (y_2, x_2)$ be feedback connections. Then $\kappa_1(\kappa_2(I)) \equiv \kappa_2(\kappa_1(I))$.*

EXAMPLE 7. *Consider the diagram of interfaces shown in Figure 2. Suppose that I_M is a Moore interface with respect to u. This diagram can be expressed as the composition*

$$\kappa\Big(\theta\big(I_1, (I_2 \| I_M)\big)\Big)$$

where $\theta := \{(y_1, z_1), (y_2, x_2)\}$ and $\kappa := (w, u)$.

LEMMA 5. *Let I be a Moore interface with respect to some of its input variables, and let κ be a feedback connection on I. Then $\mathcal{R}(\kappa(I)) \subseteq \mathcal{R}(I)$.*

LEMMA 6. *Let $I = (X, Y, \xi)$ be a Moore interface with respect to $x \in X$, and let $\kappa = (y, x)$ be a feedback connection on I. Let $\kappa(I) = (X \setminus \{x\}, Y \cup \{y\}, \xi_\kappa)$. Then for any $s \in \mathcal{R}(\kappa(I))$, the formula $\mathsf{in}(\xi_\kappa(s)) \equiv \mathsf{in}(\xi(s))$ is valid.*

THEOREM 7 (FEEDBACK PRESERVES WELL-FORMEDNESS). *Let I be a Moore interface with respect to some of its input variables, and let κ be a feedback connection on I. If I is well-formed then $\kappa(I)$ is well-formed.*

Feedback does not preserve well-formability:

EXAMPLE 8. *Consider a finite-state interface I_f with two states, s_0 (the initial state) and s_1, one input variable x and one output variable y. I_f remains at state s_0 when $x \neq 0$ and moves from s_0 to s_1 when $x = 0$. Let $\phi_0 := y = 0$ be the contract at state s_0 and let $\phi_1 := \mathsf{false}$ be the contract at state s_1. I_f is not well-formed because ϕ_1 is unsatisfiable while state s_1 is reachable. I_f is well-formable, however: it suffices to restrict ϕ_0 to $\phi_0' := y = 0 \wedge x \neq 0$. Denote the resulting (well-formed) interface by I_f'. Note that I_f is Moore with respect to x, whereas I_f' is not. Let κ be the feedback connection (y, x). Because I_f is Moore, $\kappa(I_f)$ is defined, and is such that its contract at state s_0 is $y = 0 \wedge x = y$, and its contract at state s_1 is $\mathsf{false} \wedge x = y \equiv \mathsf{false}$. $\kappa(I_f)$ is not well-formable: indeed, $y = 0 \wedge x = y$ implies $x = 0$, therefore, state s_1 cannot be avoided.*

6. HIDING

As can be seen in Example 4, composition often creates redundant output variables, in the sense that some of these variables are equal to each other. This happens because input variables that get connected become output variables.

To remove redundant (or other) output variables, we propose a *hiding* operator. For a stateless interface $I = (X, Y, \phi)$, the (stateless) interface resulting from hiding a subset of output variables $Y' \subseteq Y$ can simply be defined as:

$$\mathsf{hide}(Y', I) := (X, Y \setminus Y', \exists Y' : \phi)$$

This definition does not directly extend to the general case of stateful interfaces, however. The reason is that the contract of a stateful interface I may depend on the history of an output y. Then, hiding y is problematic because it is unclear how the contracts of different histories should be combined. To avoid this problem, we allow hiding only those outputs which do not influence the evolution of the contract.

Given $s, s' \in \mathcal{A}(X \cup Y)^*$ such that $|s| = |s'|$ (i.e., s, s' have same length), and given $Z \subseteq X \cup Y$, we say that s and s' *agree on* Z, denoted $s =_Z s'$, when for all $i \in \{1, ..., |s|\}$, and all $z \in Z$, $s_i(z) = s_i'(z)$. Given interface $I = (X, Y, \xi)$, we say that ξ *is independent from* Z if for every $s, s' \in \mathcal{A}(X \cup Y)^*$, $s =_{(X \cup Y) \setminus Z} s'$ implies $\xi(s) = \xi(s')$. That is, the evolution of the variables in Z does not affect the evolution of the contract of I.

Notice that ξ being independent from Z does *not* imply that the contracts of I cannot refer to variables in Z. Indeed, all stateless interfaces trivially satisfy the independence condition: their contracts are invariant in time, i.e., they do not depend on the evolution of variables. Clearly, the contract of a stateless interface can refer to any of its variables.

When ξ is independent from Z, variables in Z can be hidden. In particular, ξ can be viewed as a function from $\mathcal{A}((X \cup Y) \setminus Z)^*$ to $\mathcal{F}(X \cup Y)$ instead of a function from $\mathcal{A}(X \cup Y)^*$ to $\mathcal{F}(X \cup Y)$. We use this when we write $\xi(s)$ for $s \in \mathcal{A}((X \cup Y) \setminus Z)^*$ in the definition the follows:

DEFINITION 13 (HIDING). *Let $I = (X, Y, \xi)$ be an interface and let $Y' \subseteq Y$, such that ξ is independent from Y'. Then, $\mathsf{hide}(Y', I)$ is defined to be the interface*

$$\mathsf{hide}(Y', I) := (X, Y \setminus Y', \xi') \quad (7)$$

such that for any $s \in \mathcal{A}(X \cup Y \setminus Y')^*$, $\xi'(s) := \exists Y' : \xi(s)$.

7. REFINEMENT

DEFINITION 14 (REFINEMENT). *Consider two interfaces $I = (X, Y, \xi)$ and $I' = (X', Y', \xi')$. We say that I' refines I, written $I' \sqsubseteq I$, iff $X = X'$, $Y = Y'$, and for any $s \in \mathcal{R}(I) \cap \mathcal{R}(I')$, the following formulae are valid:*

$$\mathsf{in}(\xi(s)) \rightarrow \mathsf{in}(\xi'(s)) \quad (8)$$

$$\big(\mathsf{in}(\xi(s)) \wedge \xi'(s)\big) \rightarrow \xi(s) \quad (9)$$

This definition is similar in spirit to other input-contravariant refinement relations, such as *alternating refinement* [2] or refinement of A/G interfaces [6, 7], which, roughly speaking, state that I' refines I iff I' accepts more inputs and produces less outputs than I. In the case of A/G interfaces, where input assumptions are separated from output guarantees, this can be simply stated as $\mathsf{in} \rightarrow \mathsf{in}'$ and $\mathsf{out}' \rightarrow \mathsf{out}$. Our refinement is not strictly output-covariant,

however: it requires $\xi'(s) \rightarrow \xi(s)$ only for those inputs that are legal in I.

The reader may wonder why Condition (9) could not be replaced with a simpler condition, namely:

$$\xi'(s) \rightarrow \xi(s) \tag{10}$$

Indeed, for input-complete interfaces, Conditions (8) and (9) are equivalent to Condition (10), (see Theorem 21). In general, however, the two definitions are different in a profound way. Our definition characterizes pluggability in the sense of Theorem 11: I' refines I iff I' can replace I in any context. If we used Condition (10) instead of (9), then this characterization would not hold. We demonstrate this by an example.

EXAMPLE 9. *Consider interface I_1 from Example 1 and interface $I_{id} := (\{x\}, \{y\}, x = y)$. It can be checked that $I_{id} \sqsubseteq I_1$. If we used Condition (10) instead of Condition (9), however, then I_{id} would not refine I_1: this is because $x = y \not\rightarrow x > 0$. Yet there is no environment E such that $I_1 \models E$ but $I_{id} \not\models E$: this follows from Theorem 11.*

Perhaps surprisingly, among all interfaces with same sets of input and output variables, the interface with contract false is the "top" element with respect to the $\sqsubseteq$ order, that is, it is refined by every other interface. This is in accordance with Theorem 11. The false interface is pluggable only in the trivial environment that stops immediately. Clearly, any other interface can be plugged into this environment as well.

We proceed to state our main results about refinement.

LEMMA 7. *Let I, I', I'' be interfaces and suppose $I'' \sqsubseteq I'$ and $I' \sqsubseteq I$. Then $\mathcal{R}(I) \cap \mathcal{R}(I'') \subseteq \mathcal{R}(I')$.*

THEOREM 8 (REFLEXIVITY, TRANSITIVITY). *$\sqsubseteq$ is a reflexive and transitive relation on interfaces.*

Refinement preserves well-formedness for stateless interfaces:

THEOREM 9. *Let I, I' be stateless interfaces such that $I' \sqsubseteq I$. If I is well-formed then I' is well-formed.*

Theorem 9 does not generally hold for stateful interfaces: the reason is that, because I' may accept more inputs than I, there may be states that are reachable in I' but not in I, and the contract of I' in these states may be unsatisfiable. When this situation does not occur, refinement preserves well-formedness also in the stateful case. Moreover, refinement always preserves well-formability:

THEOREM 10. *Let I, I' be interfaces such that $I' \sqsubseteq I$. If I is well-formed and $\mathcal{R}(I') \subseteq \mathcal{R}(I)$ then I' is well-formed. Moreover, if I is well-formable then I' is well-formable.*

The following two lemmata, together with Lemma 2, are used in the proof of Theorem 11.

LEMMA 8. *Consider properties ϕ, ϕ' over $X \cup Y$ such that*

$$\Big(\mathsf{in}(\phi) \rightarrow \mathsf{in}(\phi')\Big) \wedge \Big(\big(\mathsf{in}(\phi) \wedge \phi'\big) \rightarrow \phi\Big)$$

is valid. Then for any property ψ over Y, the following formula is also valid:

$$\big(\mathsf{in}(\phi) \wedge (\phi \rightarrow \psi)\big) \rightarrow \big(\mathsf{in}(\phi') \wedge (\phi' \rightarrow \psi)\big)$$

LEMMA 9. *Let I, I' be interfaces and E be a environment. If I is pluggable to E and $I' \sqsubseteq I$ then $\mathcal{R}(I'_E) \subseteq \mathcal{R}(I_E)$.*

Theorem 11 is one of the main properties of our theory. It states that refinement characterizes pluggability.

THEOREM 11 (REFINEMENT AND PLUGGABILITY). *$I' \sqsubseteq I$ iff for all environments E, $I \models E$ implies $I' \models E$.*

The following lemma is used in the proof of Theorem 12.

LEMMA 10. *Consider two disjoint interfaces I_1 and I_2, and a connection θ between I_1, I_2. Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be the projections of $\mathcal{R}(\theta(I_1, I_2))$ to states over the variables of I_1 and I_2, respectively. Then $\mathcal{R}_1 \subseteq \mathcal{R}(I_1)$ and $\mathcal{R}_2 \subseteq \mathcal{R}(I_2)$.*

Theorems 12 and 13 state another main property of our theory, namely, that refinement is preserved by composition.

THEOREM 12 (CONNECTION PRESERVES REFINEMENT). *Consider two disjoint interfaces I_1 and I_2, and a connection θ between I_1, I_2. Let I'_1, I'_2 be interfaces such that $I'_1 \sqsubseteq I_1$ and $I'_2 \sqsubseteq I_2$. Then, $\theta(I'_1, I'_2) \sqsubseteq \theta(I_1, I_2)$.*

THEOREM 13 (FEEDBACK PRESERVES REFINEMENT). *Let I, I' be interfaces such that $I' \sqsubseteq I$. Suppose both I and I' are Moore interfaces with respect to one of their input variables, x. Let $\kappa = (y, x)$ be a feedback connection. Then $\kappa(I') \sqsubseteq \kappa(I)$.*

Note that the assumption that I' be Moore w.r.t. x in Theorem 13 is essential. Indeed, Mooreness is not generally preserved by refinement, as Example 10 shows.

EXAMPLE 10. *Consider the stateless interfaces $I_{even} := (\{x\}, \{y\}, y \mod 2 = 0)$, where mod denotes the modulo operator, and $I_{\times 2} := (\{x\}, \{y\}, y = 2x)$. I_{even} is Moore. $I_{\times 2}$ is not Moore. Yet $I_{\times 2} \sqsubseteq I_{even}$.*

Thanks to Theorems 9 and 10, a corollary of Theorem 12 is that composition by connection preserves well-formability for general interfaces, and well-formedness for stateless interfaces. Similarly, a corollary of Theorem 13 is that feedback composition preserves well-formability for general Moore interfaces, and well-formedness for stateless Moore interfaces.

8. SHARED REFINEMENT

Shared refinement is introduced in [7] as a mechanism to combine two interfaces I and I' into a single interface $I \sqcap I'$ that refines both I and I': $I \sqcap I'$ is able to accept inputs that are legal in either I or I', and provide outputs that are legal in both I and I'. Because of this, $I \sqcap I'$ can replace both I and I', which, as argued in [7], is important for component reuse.

A shared refinement operator for extended (i.e., relational) interfaces is proposed in the discussion section of [7], and it is conjectured that this operator represents the greatest lower bound with respect to refinement. We show that this holds only if a *shared refinability* condition is imposed. This condition states that for every inputs that is legal in both I and I', the corresponding sets of outputs of I and I' must have a non-empty intersection. Otherwise, it is impossible to provide an output that is legal in both I and I'.

DEFINITION 15 (SHARED REFINEMENT). *Two interfaces* $I = (X, Y, \xi)$ *and* $I' = (X', Y', \xi')$ *are* shared-refinable *if* $X = X'$, $Y = Y'$ *and the following formula is true for all* $s \in \mathcal{R}(I) \cap \mathcal{R}(I')$*:*

$$\forall X : \big(\mathsf{in}(\xi(s)) \wedge \mathsf{in}(\xi'(s))\big) \rightarrow \exists Y : (\xi(s) \wedge \xi'(s)) \qquad (11)$$

In that case, the shared refinement *of* I *and* I'*, denoted* $I \sqcap I'$*, is the interface:*

$$I \sqcap I' \;:=\; (X, Y, \xi_\sqcap)$$

$$\xi_\sqcap(s) \;:=\; \begin{cases} \big(\mathsf{in}(\xi(s)) \vee \mathsf{in}(\xi'(s))\big) \wedge \big(\mathsf{in}(\xi(s)) \rightarrow \xi(s)\big) \wedge \\ \qquad \big(\mathsf{in}(\xi'(s)) \rightarrow \xi'(s)\big), \text{ if } s \in \mathcal{R}(I) \cap \mathcal{R}(I') \\ \xi(s), \text{ if } s \in \mathcal{R}(I) \setminus \mathcal{R}(I') \\ \xi'(s), \text{ if } s \in \mathcal{R}(I') \setminus \mathcal{R}(I) \end{cases}$$

EXAMPLE 11. *Consider interfaces* $I_{00} := (\{x\}, \{y\}, x = 0 \rightarrow y = 0)$ *and* $I_{01} := (\{x\}, \{y\}, x = 0 \rightarrow y = 1)$*.* I_{00} *and* I_{01} *are not shared-refinable because there is no way to satisfy* $y = 0 \wedge y = 1$ *when* $x = 0$*.*

LEMMA 11. *If* I *and* I' *are shared-refinable interfaces then*

$$\mathcal{R}(I) \cap \mathcal{R}(I') \subseteq \mathcal{R}(I \sqcap I') \subseteq \mathcal{R}(I) \cup \mathcal{R}(I')$$

LEMMA 12. *Let* I *and* I' *be shared-refinable interfaces such that* $I = (X, Y, \xi)$*,* $I' = (X, Y, \xi')$ *and* $I \sqcap I' = (X, Y, \xi_\sqcap)$*. Then, for all* $s \in \mathcal{R}(I) \cap \mathcal{R}(I')$

$$\mathsf{in}(\xi_\sqcap(s)) \equiv \mathsf{in}(\xi(s)) \vee \mathsf{in}(\xi'(s))$$

THEOREM 14 (GREATEST LOWER BOUND). *If* I *and* I' *are shared-refinable interfaces then* $(I \sqcap I') \sqsubseteq I$*,* $(I \sqcap I') \sqsubseteq I'$*, and for any interface* I'' *such that* $I'' \sqsubseteq I$ *and* $I'' \sqsubseteq I'$*, we have* $I'' \sqsubseteq (I \sqcap I')$*.*

THEOREM 15. *If* I *and* I' *are shared-refinable interfaces and both are well-formed, then* $I \sqcap I'$ *is well-formed.*

9. THE INPUT-COMPLETE CASE

Input-complete interfaces do not restrict the set of input values, although they may provide no guarantees when the input values are illegal. Although input-complete interfaces are a special case of general interfaces, it is instructive to study them separately for two reasons: first, input-completeness makes things much simpler, thus easier to understand and implement; second, some interesting properties hold for input-complete interfaces but not in general.

DEFINITION 16 (INPUT-COMPLETE INTERFACE). *An interface* $I = (X, Y, \xi)$ *is* input-complete *if for all* $s \in \mathcal{A}(X \cup Y)^*$*,* $\mathsf{in}(\xi(s))$ *is valid.*

THEOREM 16. *Every input-complete interface is well-formed.*

Definition 17 and Theorem 17 that follow show that every interface I can be turned into an input-complete interface $\mathsf{IC}(I)$ that refines I.

DEFINITION 17 (INPUT-COMPLETION). *Consider an interface* $I = (X, Y, \xi)$*. The* input-complete version *of* I*, denoted* $\mathsf{IC}(I)$*, is the interface* $\mathsf{IC}(I) := (X, Y, \xi_{ic})$*, where* $\xi_{ic}(s) := \xi(s) \vee \neg\mathsf{in}(\xi(s))$*, for all* $s \in \mathcal{A}(X \cup Y)^*$*.*

THEOREM 17 (INPUT-COMPLETE REFINES ORIGINAL). *If* I *is an interface then: (1)* $\mathsf{IC}(I)$ *is an input-complete interface, and (2)* $\mathsf{IC}(I) \sqsubseteq I$*.*

Theorems 17 and 11 imply that for any environment E, if $I \models E$ then $\mathsf{IC}(I) \models E$. The converse does not hold in general (see Examples 1 and 3, and observe that I_2 is the input-complete version of I_1).

Composition by connection reduces to conjunction of contracts for input-complete interfaces, and preserves input-completeness:

THEOREM 18. *Let* $I_i = (X_i, Y_i, \xi_i)$*,* $i = 1, 2$*, be disjoint input-complete interfaces, and let* θ *be a connection between* I_1, I_2*. Then the contract function* ξ *of the composite interface* $\theta(I_1, I_2)$ *is such that for all* $s \in \mathcal{A}(X_{\theta(I_1,I_2)} \cup Y_{\theta(I_1,I_2)})^*$

$$\xi(s) \;\equiv\; \xi_1(s) \wedge \xi_2(s) \wedge \rho_\theta$$

Moreover, $\theta(I_1, I_2)$ *is input-complete.*

It is important to note that taking $\xi_1(s) \wedge \xi_2(s) \wedge \rho_\theta$ as the contract of a composite interface does not work for general interfaces, even though it works for input-complete interfaces. This is illustrated in the following example.

EXAMPLE 12. *Let*

$$I_{10} \;:=\; \big(\{x\}, \{y\}, x = 0 \wedge (y = 0 \vee y = 1)\big)$$
$$I_{12} \;:=\; (\{z\}, \{w\}, z = 0 \wedge w = 0)$$

Let $\theta := \{(y, z)\}$*. The conjunction of the contracts of* I_{10} *and* I_{12}*, together with the equality* $y = z$ *imposed by the connection* θ*, gives the contract* $x = 0 \wedge (y = 0 \vee y = 1) \wedge z = 0 \wedge w = 0 \wedge y = z$*, which is equivalent to* $x = y = z = w = 0$*, which is clearly satisfiable. Therefore, we could interpret the composite interface* $\theta(I_{10}, I_{12})$ *as the interface*

$$(\{x\}, \{y, z, w\}, x = y = z = w = 0)$$

Now, consider the interface:

$$I_{11} \;:=\; (\{x\}, \{y\}, x = 0 \wedge y = 1)$$

It can be checked that $I_{11} \sqsubseteq I_{10}$*. But if we connect* I_{11} *to* I_{12}*, we find that the conjunction of their contracts (with the connection* $y = z$*) is unsatisfiable. Therefore, if we used conjunction for composition by connection, then the composite interface* $\theta(I_{11}, I_{12})$ *would not refine* $\theta(I_{10}, I_{12})$*, even though* I_{11} *refines* I_{10}*, i.e., Theorem 12 would not hold.*

Input-complete interfaces alone do not help in avoiding problems with arbitrary feedback compositions: indeed, in the example given in the introduction both interfaces I_{true} and $I_{y \neq x}$ are input-complete.[6] This means that in order to add a feedback connection (y, x) in an input-complete interface, we must still ensure that this interface is Moore w.r.t. input x. In that case, feedback preserves input-completeness.

THEOREM 19. *Let* $I = (X, Y, \xi)$ *be an input-complete interface which is also Moore with respect to some* $x \in X$*. Let* $\kappa = (y, x)$ *be a feedback connection on* I*. Then* $\kappa(I)$ *is input-complete.*

THEOREM 20. *Let* $I = (X, Y, \xi)$ *be an input-complete interface and let* $Y' \subseteq Y$*, such that* ξ *is independent from* Y'*. Then,* $\mathsf{hide}(Y', I)$ *is input-complete.*

[6] It is not surprising that input-complete interfaces alone cannot solve the problems with arbitrary feedback compositions, since these are general problems of causality, not particular to interfaces.

THEOREM 21. *Let $I = (X, Y, \xi)$ and $I' = (X', Y', \xi')$ be input-complete interfaces. Then $I' \sqsubseteq I$ iff for all $s \in \mathcal{A}(X \cup Y)^*$, $\xi'(s) \rightarrow \xi(s)$ is valid.*

For input-complete interfaces, the shared-refinability condition, i.e., Condition (11), simplifies to

$$\forall X : \exists Y : \xi(s) \wedge \xi'(s)$$

Clearly, this condition does *not* always hold. Indeed, the interfaces of Example 11 are not shared-refinable, even though they are input-complete. For shared-refinable input-complete interfaces, shared refinement reduces to conjunction of contracts for states that are reachable in both interfaces.

THEOREM 22. *Let $I = (X, Y, \xi)$ and $I' = (X, Y, \xi')$ be input-complete shared-refinable interfaces. Then $I' \sqcap I = (X, Y, \xi_\sqcap)$, where for all $s \in \mathcal{R}(I) \cap \mathcal{R}(I')$, $\xi_\sqcap(s) \equiv \xi(s) \wedge \xi'(s)$.*

As the above presentation shows, input-complete interfaces are much simpler than general interfaces: refinement is implication of contracts, composition is conjunction, and so on. Then, a legitimate question is, why consider non-input-complete interfaces at all? There are mainly two reasons.

First, non-input-complete interfaces can be used to model situations that cannot be modeled by input-complete interfaces. For example, consider modeling a component implementing some procedure that requires certain conditions on its inputs to be satisfied, otherwise it may not terminate. We can capture the specification of this component as an interface, by imposing these conditions in the contract of the interface. But we cannot capture the same specification as an input-complete interface: for what would the output be when the input conditions are violated? We cannot simply add an extra output taking values in $\{T, NT\}$, for "terminates" and "does not terminate", since non-termination is not an observable property.

Second, even in the case where we could use input-complete interfaces to capture a specification, we may decide not to do so, in order to allow for *local compatibility* checks. In particular, when connecting two interfaces I and I', we may want to check that their composition is well-formed *before* proceeding to form an entire interface diagram. Input-complete interfaces are always well-formed and so are their compositions (Theorems 16, 18 and 19), therefore, local compatibility checks provide useful information only in the non-input-complete case.

10. CONCLUSIONS

Synchronous interface theories have been proposed before [6, 4, 7] but fail to capture input-output relations in a general way. The main message of this paper is that a theory of synchronous relational interfaces *can* be developed, provided feedback compositions are restricted appropriately.

In the future, we plan to further extend this theory. One useful extension would be to refine the definition of Moore interfaces to speak about dependencies between specific pairs of input and output variables. This would allow to express, for example, the fact that in the parallel composition of $(\{x_1\}, \{y_1\}, x_1 = y_1)$ and $(\{x_2\}, \{y_2\}, x_2 = y_2)$, y_1 does not depend on x_2 and y_2 does not depend on x_1 (and therefore one of the feedbacks (y_1, x_2) or (y_2, x_1) can be allowed). Such an extension could perhaps be achieved by combining our relational interfaces with the *causality interfaces* of [12], input-output dependency information such as that used in reactive modules [1], or the coarser *profiles* of [9]. We expect this extended theory to have a single composition operator, which generalizes and captures both composition by connection and composition by feedback.

11. REFERENCES

[1] R. Alur and T. Henzinger. Reactive modules. *Formal Methods in System Design*, 15:7–48, 1999.

[2] R. Alur, T. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In *CONCUR'98*, volume 1466 of *LNCS*. Springer, 1998.

[3] P. Caspi, D. Pilaud, N. Halbwachs, and J. Plaice. Lustre: a declarative language for programming synchronous systems. In *14th ACM Symp. POPL*. ACM, 1987.

[4] A. Chakrabarti, L. de Alfaro, T. Henzinger, and F. Mang. Synchronous and bidirectional component interfaces. In *CAV*, LNCS 2404, pages 414–427. Springer, 2002.

[5] L. de Alfaro and T. Henzinger. Interface automata. In *Foundations of Software Engineering (FSE)*. ACM Press, 2001.

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

[7] L. Doyen, T. Henzinger, B. Jobstmann, and T. Petrov. Interface theories with component reuse. In *8th ACM & IEEE International conference on Embedded software, EMSOFT*, pages 79–88, 2008.

[8] E. Lee and A. Sangiovanni-Vincentelli. A unified framework for comparing models of computation. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 17(12):1217–1229, Dec. 1998.

[9] R. Lublinerman and S. Tripakis. Modularity vs. Reusability: Code Generation from Synchronous Block Diagrams. In *Design, Automation, and Test in Europe (DATE'08)*. ACM, Mar. 2008.

[10] G. Tourlakis. *Mathematical Logic*. Wiley, 2008.

[11] S. Tripakis, B. Lickly, T. A. Henzinger, and E. A. Lee. On Relational Interfaces. Technical Report UCB/EECS-2009-60, EECS Department, University of California, Berkeley, May 2009. `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-60.html`.

[12] Y. Zhou and E. Lee. Causality interfaces for actor networks. *ACM Trans. Embed. Comput. Syst.*, 7(3):1–35, 2008.