Feedback in Synchronous Relational Interfaces[⋆]

Stavros Tripakis[1,2] and Chris Shaver[1]

[1] University of California, Berkeley
[2] Aalto University

Abstract. Synchronous relational interfaces is an interface theory which allows to specify the I/O interface of a component with input requirements and relational input-output guarantees. The theory allows to check interface compatibility during composition and to compute a composite interface from the atomic ones. It provides a refinement operator which allows to check whether a component can safely replace another one. This paper discusses the options and challenges in defining feedback composition in the context of this theory.

1 Introduction

Compositionality is not simply a desirable property in system design, but a "must" for building large and complex systems from smaller and simpler components. In his long career, Joseph Sifakis has pursued numerous research topics around compositionality, including, but not limited to [13,15,11,3,14,4,1,5].

The work presented in this paper is also on the general subject of compositionality. Our work approaches the subject following the framework of so-called *interface theories* [8,7]. In interface theories, components are captured by abstract models generically called interfaces. Such a theory also provides one or more interface composition operators, each allowing to obtain an interface for a composite component (i.e., a network of connected subcomponents) from the interfaces of the subcomponents. Finally, an interface theory provides a *refinement* relation between interfaces, which typically comes with two key theorems:

– Preservation of properties of interest by refinement: if interface I satisfies a given property ϕ (say, a safety property expressed in temporal logic), and interface I' refines I, then I' also satisfies ϕ.
– Preservation of refinement by composition: if I_1' refines I_1, I_2' refines I_2, and $\odot$ is a composition operator, then $I_1' \odot I_2'$ refines $I_1 \odot I_2$.

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

Together the above theorems enable an incremental design methodology, which, for instance, allows to reduce the problem of checking substitutability (when can a component with interface I' replace a component with interface I?) to that of checking whether I' refines I. Indeed, if I is used in a certain composition $I \odot C$, and I' refines I, then I' can replace I to obtain the new composition $I' \odot C$. By preservation of refinement by composition, $I' \odot C$ refines $I \odot C$. By preservation of properties by refinement, if $I \odot C$ satisfies a given ϕ, then so does $I' \odot C$. This means that, provided the existing system $I \odot C$ is known to be correct (i.e., to satisfy ϕ), the new system $I' \odot C$ need not be re-verified from scratch. We only need to check whether I' refines I to ensure substitutability.

A number of interface theories have been proposed over the years, starting from the original *interface automata* (IA) theory proposed in [7]. In this paper we consider the theory of *synchronous relational interfaces* proposed in [16]. Whereas interface automata use an asynchronous model of concurrency based on interleaving and input-output label synchronization, synchronous relational interfaces use synchronous composition similar to finite state machines of type Moore or Mealy.

In addition, compared to interface automata, synchronous relational interfaces offer a more compact and symbolic specification formalism. For instance, consider a component which receives as input an integer, adds one to it, and outputs it. This component could be modeled using the synchronous relational interface $(\{x\}, \{y\}, y = x + 1)$. Here, x and y are the input and output variables, respectively, and $y = x + 1$ is a formula capturing the input-output relation (or *contract*). The same component could be captured as an interface automaton, but this would most likely require an infinite number of states, transitions, and labels, to capture the infinite domain of possible input/output values.[3]

The theory of synchronous relational interfaces provides three composition operators: composition in series (connecting an output of one component to an input of another), in parallel (placing the two components next to each other without any connections), and in feedback (connecting one of the outputs of a component to one of its inputs). These are standard composition schemes found in synchronous systems such as, for instance, digital circuits. This work is concerned specifically with feedback composition. The work of [16] allows only a restricted form of feedback composition. This paper recalls the reasons why this is so, discusses why it would be desirable to extend the theory to allow a less restrictive version of feedback, and examines the challenges in doing so.

2 Background: synchronous relational interfaces

For the purposes of this paper, it suffices to restrict ourselves to the simplest form of synchronous relational interfaces, namely, *stateless* interfaces, where the

[3] We can imagine an interface automaton with transitions of the form $\xrightarrow{i_0?}\xrightarrow{o_1!}$, $\xrightarrow{i_1?}\xrightarrow{o_2!}$, etc., where $i_n?$ is the input action corresponding to reading input $x = n$, and $o_k!$ is the output action corresponding to writing output $y = k$.

input/output contract is the same during the dynamic behavior of the component, i.e., at every synchronous cycle. *Stateful* interfaces are also considered in [16], where the contract may change from one cycle to the next. We only consider stateless interfaces in the sequel.

A (stateless synchronous relational) interface is a triple

$$I = (X, Y, \phi)$$

where X is a finite set of input variables, Y is a finite set of output variables, $X \cap Y = \emptyset$ (input and output variables are disjoint), and ϕ (the *contract*) is a relation between values of input and output variables, typically represented as a logical formula on the set of variables $X \cup Y$. We assume a universe U of possible values for all variables. $V(X)$ denotes the set of *assignments* (or *valuations*) over a set of variables X, that is, the set of all functions of the form $a : X \to U$. Semantically, a formula ϕ on $X \cup Y$ denotes the set of assignments over $X \cup Y$ which satisfy ϕ.

Assuming $Y = \{y_1, y_2, ..., y_n\}$, we define

$$\mathbf{in}(\phi) \quad := \quad \exists Y : \phi \quad := \quad \exists y_1 : \exists y_2 : \cdots \exists y_n : \phi.$$

That is, $\mathbf{in}(\phi)$ is syntactically a formula only on input variables X, characterizing the set of *legal* input assignments. For example, if ϕ is $x \neq 0 \wedge y \geq x$, and x is an input and y an output, then $\mathbf{in}(\phi) \equiv x \neq 0$, meaning that $x = 0$ is illegal.

ϕ, and in turn I, are called *input-complete* when $\mathbf{in}(\phi) \equiv \mathbf{true}$, i.e., when all input assignments are legal for I.

ϕ, and in turn I, are called *deterministic* when for every legal input assignment, i.e., for every function $a_X : X \to U$ satisfying $\mathbf{in}(\phi)$, there is a unique output assignment $a_Y : Y \to U$, such that the pair (a_X, a_Y) satisfies ϕ.

Parallel composition. Parallel composition of relational interfaces can be defined by taking the conjunction of their corresponding contracts. Let $I_i = (X_i, Y_i, \phi_i)$, for $i = 1, 2$, where all sets X_1, X_2, Y_1, Y_2 are pair-wise disjoint. Then

$$I_1 || I_2 := (X_1 \cup X_2, Y_1 \cup Y_2, \phi_1 \wedge \phi_2)$$

Note that $\mathbf{in}(\phi_1 \wedge \phi_2) \equiv \exists Y_1, Y_2 : \phi_1 \wedge \phi_2 \equiv \exists Y_1 : (\phi_1 \wedge \exists Y_2 : \phi_2) \equiv (\exists Y_2 : \phi_2) \wedge (\exists Y_1 : \phi_1) \equiv \mathbf{in}(\phi_1) \wedge \mathbf{in}(\phi_2)$.

Serial composition. For reasons thoroughly explained in [16], and not repeated here, serial composition of relational interfaces is defined using the principle of "demonic" non-determinism. In the simple case, where $I_1 = (\{x\}, \{y\}, \phi_1)$ and $I_2 = (\{y\}, \{z\}, \phi_2)$, the serial composition of I_1 and I_2, denoted $I_1 \rightsquigarrow I_2$, and consisting of connecting the output y of I_1 to the input y of I_2, is defined as follows:

$$I_1 \rightsquigarrow I_2 := \Big(\{x\}, \{y, z\}, \phi_1 \wedge \phi_2 \wedge \big(\forall y : \phi_1 \to \mathbf{in}(\phi_2)\big) \Big)$$

Note that if I_2 is input-complete, then $\big(\forall y : \phi_1 \rightarrow \mathbf{in}(\phi_2)\big) \equiv \mathbf{true}$ and the contract of $I_1 \rightsquigarrow I_2$ becomes $\phi_1 \wedge \phi_2$. The same is true when I_1 is deterministic.

When the contract of $I_1 \rightsquigarrow I_2$ is equivalent to **false** (i.e., unsatisfiable), we say that $I_1 \rightsquigarrow I_2$ is *invalid* and that I_1 and I_2 are *incompatible*. Otherwise, we say that $I_1 \rightsquigarrow I_2$ is *valid* and that I_1 and I_2 are *compatible*.

Example 1. Let $I_1 = (\{x\}, \{y\}, x \leq y)$ and $I_2 = (\{y\}, \{z\}, y \neq 0)$. Then

$$I_1 \rightsquigarrow I_2 = (\{x\}, \{y, z\}, x \leq y \wedge y \neq 0 \wedge x > 0)$$

where it is worth noting the additional input assumption $x > 0$ obtained thanks to the term $\forall y : x \leq y \rightarrow y \neq 0$. $\square$

Refinement. Refinement between relational interfaces is defined as follows. Let $I_i = (X, Y, \phi_i)$, for $i = 1, 2$. Then, I_2 refines I_1, written $I_2 \sqsubseteq I_1$, iff

$$\mathbf{in}(\phi_1) \rightarrow \mathbf{in}(\phi_2) \qquad \text{and} \qquad \big(\mathbf{in}(\phi_1) \wedge \phi_2\big) \rightarrow \phi_1$$

are both valid formulas, i.e., equivalent to **true**.

It is shown in [16] that refinement preserves compatibility, that is, if $I_1 \rightsquigarrow I_2$ is valid, and $I_1' \sqsubseteq I_1$ and $I_2' \sqsubseteq I_2$, then $I_1' \rightsquigarrow I_2'$ is also valid.

Feedback. Suppose we want to connect an output $y \in Y$ of a relational interface $I = (X, Y, \phi)$ to one of its inputs $x \in X$. In the framework of [16], this is allowed only when I is so-called *Moore with respect to x*. For stateless interfaces, Moore with respect to x means that ϕ does not refer to x. In that case, connecting y to x results in a new interface where x is an output equal to y:

$$\mathit{feedback}_{y \rightsquigarrow x}(I) \quad := \quad (X - \{x\}, Y \cup \{x\}, \phi \wedge x = y).$$

The problem: general feedback does not preserve refinement. The reason why feedback is restricted to Moore interfaces is illustrated in the following example, borrowed from [9] and also discussed as Example 9.11 in [16].

Example 2. Let $I = (\{x, z\}, \{y\}, \mathbf{true})$ and $I' = (\{x, z\}, \{y\}, x \neq y)$. Then $I' \sqsubseteq I$. Suppose that we want to feed the output of I back to its first input, that is, we want to connect y to x. The straightforward way to define the resulting feedback composition is by adding the constraint $x = y$ to the contract of I. This constraint represents the fact that, once x and y are connected (imagine a wire connection between the two) their values become equal. Adding this constraint, that is, taking the conjunction of $x = y$ with the contract of I, which is **true**, we obtain the new interface $I_f = (\{z\}, \{x, y\}, x = y)$. In I_f, x is now an output, since it has been connected to y. Moreover, the new contract is $x = y$.

Let us try to do the same with I', that is, connect its output y to its input x. Doing so, we obtain the new interface $I_f' = (\{z\}, \{x, y\}, x \neq y \wedge x = y)$. Since the formula $x \neq y \wedge x = y$ is unsatisfiable, I_f' is equivalent to the interface $(\{z\}, \{x, y\}, \mathbf{false})$.

The problem now is that $I'_f \sqsubseteq I_f$ does **not** hold. This shows that the straightforward way of defining feedback results in refinement not being preserved by feedback (I' refines I, but I'_f does not refine I_f). $\square$

3 Generalizing feedback

In this section, we first discuss why a more general form of feedback would be desirable, and then the challenges that need to be overcome in order to achieve this more general form of feedback.

3.1 Why generalize the definition of feedback?

One might say that Example 2 is too artificial to be of value, and that forbidding feedback for non-Moore interfaces makes sense. Consider, however, the following example:

Example 3. Take the parallel composition of two interfaces with contracts $y_i = x_i$, where x_i is an input and y_i is an output, for $i = 1, 2$. The resulting interface has contract $y_1 = x_1 \wedge y_2 = x_2$. This product interface is not Moore in neither x_1 nor x_2. Thus, we cannot form the feedback composition by connecting, say, y_2 to x_1. One might expect, however, that this feedback connection is the same as connecting the two original interfaces in series. $\square$

What Example 3 illustrates is that the restriction to Moore interfaces results in serial composition not being equivalent to parallel composition followed by feedback. Can we relax the restrictions so as to obtain a definition of feedback which allows to express serial composition as parallel composition followed by feedback? We examine the challenges in achieving this goal next.

3.2 Challenge: monolithic order

Example 3 suggests that the definition of parallel composition is too *monolithic*, in the sense that it loses dependency information between inputs and outputs. This seems to be a fundamental problem, as illustrated with an even simpler example:

Example 4. Consider interfaces $I_1 = (\{\}, \{y\}, \mathbf{true})$ and $I_2 = (\{x, z\}, \{\}, \mathbf{true})$. I_1 has only an output y and I_2 has two inputs x, z. Clearly, the serial composition $I_1 \leadsto_{y \leadsto x} I_2$ formed by connecting y to x [4] is valid, since I_2 is input-complete.

Now let's try to form the same composition by first taking the parallel composition of I_1 and I_2, followed by feedback. The parallel composition of I_1 and I_2 is $I_1 || I_2 = (\{x, z\}, \{y\}, \mathbf{true})$. This is exactly interface I which we saw in Example 2. If we forbid connecting I in feedback, as suggested above, then I_1 connected in series with I_2 would **not** be equivalent with $I_1 || I_2$ connected in feedback, since the latter connection would be forbidden. $\square$

[4] this composition is defined after renaming x to y in I_2

The problem here seems to be the following. When we form the composition in series of I_1 and I_2, we interpret it as a game where I_1 plays first, choosing the output y, and I_2 plays second, accepting y as input x. Therefore, y is chosen first, and then assigned to x. (The point where z is chosen is irrelevant here.)

On the other hand, in a "monolithic" interface such as I, the interpretation of the game is different. First, the environment chooses the input x, and only afterwards does I reply with the output y. By forming the parallel composition of I_1 and I_2, we forced the order $x \to y$. Adding the feedback creates the opposite order $y \to x$, that is, a cycle. This is not the case with composition in series, which only has $y \to x$.

One might try to fix this by enriching the definition of interface to contain also dependency information between input and output variables. In our example, this means that there would be two versions of the interface $(\{x, z\}, \{y\}, \mathbf{true})$. One version where the output y depends on x (this would be I), and another version where y does not depend on x (this would be $I_1 || I_2$).

But when we attempt to add such I/O dependency information, we run into new problems. This is explained next:

3.3 Interfaces with I/O dependency information

General partial orders on I/O variables. Let us first try an approach where an interface is extended with a general partial order D on input and output variables. That is, an interface then becomes a quadruple $I = (X, Y, \phi, D)$ where X, Y, ϕ are the inputs, outputs and contract as previously, and D is a partial order on $X \cup Y$. The idea is that D represents dependencies between the variables, and also the order in which they are evaluated, as well as the possible ways for playing the game between the component and its environment. For example, if x is input and y is output, then dependency $x \to y$ means that, first the environment chooses x and then the component chooses y. $y \to x$ means that first the component chooses y and then the environment chooses x. If x, y are unrelated then they can be evaluated in any order.

This seems to solve the problems identified in §3.2, as it allows to distinguish I (which has the dependency $x \to y$) from $I_1 || I_2$ (which has no dependency).

But consider another example:

Example 5. Let $A = (\{x\}, \{y\}, x = 0 \to y = 0, \{x \to y\})$ and $B = (\{z, u\}, \{\}, z = u, \{\})$. Suppose we wish to connect A and B in series, by connecting y to z. Is the connection valid? It should be, because the environment has two possible strategies for setting the free inputs x, u:

- either set $x = u = 0$, in which case A is forced to set $y = 0$, thus $z = 0$, thus $u = z$ and the input assumptions of B are satisfied;
- or set x to an arbitrary value, wait to observe output y of A, then set $u = y$, so that again $u = z$ is satisfied.

It seems that these two strategies cannot be represented with just a single contract ϕ and a single dependency relation D. Suppose they could. Then D would be $x \to y \to z$: notice that u is independent, since it could be given either at the same time as x, or after observing y.

Now, what would ϕ be? If u is given at the same time as x, then $x = 1, u = 0$ is not a possible assignment. On the other hand, if x is first set to 1, and then y is set to 0, which means that also $z = 0$, then u must be set to 0. So, with the second strategy, $x = 1, u = 0$ is a valid assignment, whereas with the first strategy, it is not. $\Box$

This example appears to suggest that we need *sets* of pairs (ϕ, D), instead of just one pair, to represent the sets of possible strategies that may result during composition, even if the original interfaces had only a single strategy each. This option of using sets of pairs (ϕ, D) appears too complex, and we do not pursue it further here.

Restricted DAGs: Moore outputs, inputs, non-Moore outputs. To simplify in order to avoid problems such as the one above, we may decide to restrict the I/O dependencies to a simpler form: $I = (X, Y, \phi, d)$ where $d \subseteq Y \times X$. d gives for each output the set of inputs it depends on. Those outputs that depend on no inputs are called Moore outputs. The game is played in 3 rounds: first the component chooses Moore outputs; then the environment chooses all inputs; then the component chooses non-Moore outputs.

This solves the problem of Example 5 because the second strategy, where the environment initially sets only x and then waits to observe y before setting u would be forbidden: both x, y should be set at the same time, since there is only one round to set all free inputs.

The problem with this approach is that feedback is non-commutative, as the following example illustrates.

Example 6. Let $I = (\{x_1, x_2, x_3\}, \{y_1, y_2\}, x_2 \neq y_2 \vee x_1 = y_1, \{(y_1, x_3), (y_2, x_3)\})$. In this example, both y_1, y_2 are non-Moore: x_3 is a "dummy" input that serves no other purpose except for providing dependencies for y_1, y_2 so that they are not Moore. The contract can also be read as $x_2 = y_2 \to x_1 = y_1$. Then, connecting y_1 to x_1 results in interface $\mathit{feedback}_{y_1 \rightsquigarrow x_1}(I)$ with contract $x_1 = y_1$. Following this, we can connect y_2 to x_2 to obtain the interface $\mathit{feedback}_{y_2 \rightsquigarrow x_2}(\mathit{feedback}_{y_1 \rightsquigarrow x_1}(I))$ with contract $x_1 = y_1 \wedge x_2 = y_2$. One would expect that if we do the same connections in the opposite order, i.e., first $y_2 \rightsquigarrow x_2$ and then $y_1 \rightsquigarrow x_1$, we should get the same result. But the contract of $\mathit{feedback}_{y_2 \rightsquigarrow x_2}(I)$ is **false**. Indeed, the following game is played here: first the environment chooses x_1, x_3 (there are no Moore outputs so their round is skipped); then the component chooses y_1, y_2; finally x_2 is set to y_2. The environment loses this game, since no matter what it picks for x_1, the component can always pick $y_1 \neq x_1$ and violate the contract of I, because of the feedback $x_2 = y_2$. $\Box$

Extracting dependencies from contracts. Note that there is an additional, mostly orthogonal complexity to the approach of extending interfaces with variable dependency information, and this has to do with where this information comes from. The simple approach is to expect the user to provide such information for atomic interfaces, and then compute it automatically for composite interfaces. This does not avoid the problems illustrated by the examples above.

Another approach is to try to extract dependencies automatically from the contract itself. For instance, if the contract is $y = x+1$, where x is the input and y is the output, then we could extract the dependency $x \to y$, i.e., y depends on x. This interpretation assumes that, first, the input x is given, and then the component computes the output y as $x+1$. It is unclear, however, whether this interpretation is correct. An alternative interpretation is the following: first, an output y is chosen non-deterministically; then, the input x must be given, such that $x = y - 1$. Although the first interpretation may appear more natural, there is no reason why the second one should be considered invalid. This is more obvious in an interface with a slightly different contract, say, contract **true**. Here, as mentioned above, we should be able to distinguish the case where first the environment provides the input x and then the component replies with the output y, from the opposite case, where the component provides y first, and then expects x.

The problem of extracting variable dependencies from formulas is itself interesting, although it by itself does not resolve the issues raised by Examples 5 and 6. Let us briefly discuss the problem of extracting variable dependencies from formulas. Consider a formula ϕ on a set of variables V. We may assume no knowledge of "directionality" (input vs. output) for any of these variables, and seek to define a symmetric notion of dependency (see Table 1).

One idea is to define dependency based on the principle of "geometric orthogonality". Consider a ϕ over just two variables, say, x, y. Intuitively, x, y are independent in ϕ, if ϕ is a "rectangle", that is, if ϕ is equivalent to $(\exists x : \phi) \wedge (\exists y : \phi)$. For example, in both $x = y$ and $x \neq y$, x and y are dependent, whereas in $x = 0 \wedge y = 0$, they are independent, and so are they in $0 \leq x \leq 1 \wedge 0 \leq y \leq 1$.

ϕ	dependency
$x = y$	x, y dependent
$x \neq y$	x, y dependent
$x = 0 \wedge y = 0$	x, y independent
$0 \leq x \leq 1 \wedge 0 \leq y \leq 1$	x, y independent
$x = y \wedge y = z \wedge z = 0$	x, y independent
$x = y \wedge y = z$	x, y dependent
$x = y_1 \wedge x_2 = y$	x, y independent
$x < z \wedge z < y$	x, y dependent
$z < x \wedge z < y$	x, y independent ?

Table 1. Dependency examples.

Let us try to generalize this idea to formulas with $n \geq 2$ variables. Let $\phi(\boldsymbol{y}, x_1, x_2)$ be over a set of variables $\boldsymbol{y} \cup \{x_1, x_2\}$. We can attempt the definition:

$$\mathit{indep}(\phi, x_1, x_2) \qquad := \qquad \phi \equiv \big((\exists x_1 : \phi) \wedge (\exists x_2 : \phi)\big)$$

Unfortunately this doesn't seem to work for the formula $x = y \wedge y = z$. In this case, we find that any pair of variables are independent according to the above definition. For instance, $\exists x : x = y \wedge y = z$ is $y = z$, and $\exists y : x = y \wedge y = z$ is $x = z$. This seems in contradiction with the fact that x, y are dependent in $x = y$, which is a weaker constraint than $x = y \wedge y = z$.

To capture the principle of geometric orthogonality, we may use the principle of *factorization*. Namely, if ϕ is over a set of variables X, we should be able to find a partition of X into disjoint sets $X_1, ..., X_n$, and formulas $\phi_1, ..., \phi_n$, where ϕ_i is over X_i, such that ϕ is equivalent to $\bigwedge_{i=1}^{n} \phi_i$. Then, for given variables x and y, they are independent if they do not belong in the same set X_i in the partition. This is an interesting problem, although beyond the scope of this paper.

Non-preservation of dependencies by refinement. One fundamental problem with the conceptual requirements of feedback and refinement with regards to independence is the issue that an interface that has independent variables can have refinements in which the variables are dependent. Concretely, consider the case of the interface with the predicate

$$\phi(\bar{x}, \bar{y}) = \mathbf{true}$$

This can always be written as a conjuction of functions on the individual variables, in particular because each component is similarly true.

$$\phi(\bar{x}, \bar{y}) = \phi_x^1 \wedge \phi_x^2 \wedge \ldots \phi_y^1 \wedge \phi_y^2 \wedge \ldots \quad \text{where } \phi_*^k = \mathbf{true}$$

This would suggest that the interface is equivalent to a parallel composition of a series of ambivalent source and sink actors; they can be composed serially, etc... However, as in the above examples, clearly there are refinements of the original function that can introduce dependencies. If

$$\phi'(\bar{x}, \bar{y}) = x_k \neq y_j$$

refines ϕ, as it is shown to in the above example, the interface with independent variables that seems to be decomposable into parallel parts (that can be composed serially), can be refined into an interface that does not have any admissible definition for feedback.

Intuitively, there is a general sense behind this. Consider that one alteration to an interface to refine it is to take an input for which there are multiple satisfying outputs to choose from, non-determinism, and reduce the number of options it permits, all the way down to a unique option. If an input and output variable are independent, the choice of input has no effect on the choice of output. Consequently, if there are multiple possible outputs, for any given input, making the output depend on the input in a way that still permits a satisfiable choice legitimately refines the behavior even though it is no longer independent. That is, independence is not preserved by refinement.

3.4 Lifting to powersets

Another idea is to treat contracts not as relations, but as functions. Feedback can be naturally defined on functions using fixpoint theory, so this appears to be a promising approach. Unfortunately, it is not as easy as it may appear to be at first.

To transform relations to functions, we can lift their domain and codomain to powersets. Specifically, let ϕ be a contract over I/O variable sets X and Y. Then ϕ is semantically a relation $\phi \subseteq V(X) \times V(Y)$, where $V(X)$ denotes the set of valuations over X. ϕ defines a function

$$\tilde{\phi} : 2^{V(X)} \to 2^{V(Y)}$$

where, for $V_x \subseteq V(X)$, $\tilde{\phi}(V_x)$ is defined as follows:

$$\tilde{\phi}(V_x) := \{a_Y \in V(Y) \mid \exists a_X \in V_X : (a_X, a_Y) \in \phi\}.$$

$\tilde{\phi}$ is monotonic with respect to set inclusion, so it appears as if fixpoint theory can be readily applied. However, an element of $V(X)$ is an assignment over the entire set of input variables X, and an output of the function $\tilde{\phi}$ is an assignment over the entire set of output variables Y. As a result, it is unclear how to define feedback directly on $\tilde{\phi}$. For example, we may have $X = \{x_1, x_2\}$ and $Y = \{y\}$. In this case, the output of $\tilde{\phi}$ does not match its input, since there are two input variables, and only one output variable. But even when the number of input and output variables is the same, it is unclear how to define feedback of *individual* variables. For example, we may have $X = \{x_1, x_2\}$ and $Y = \{y_1, y_2\}$, and we may want to connect y_1 to x_2. It is not clear how to express this connection as a fixpoint operation on $\tilde{\phi}$.

We can attempt to define a feedback connection like the one above using projection and product functions, in addition to the $\tilde{\phi}$ function. For instance, a projection function could be used to extract an assignment over just y_1 from an assignment over $\{y_1, y_2\}$. This may solve the typing problems and allow to define a fixpoint that type checks. However, the above functions (including $\tilde{\phi}$, projection, and product) have the property that they return the empty set when given the empty set as input. As a result, the empty set would be a valid fixpoint of any composition of such functions. Moreover, the empty set would be the *least* fixpoint with respect to set inclusion, therefore the preferred solution chosen in typical fixpoint semantics approaches. Unfortunately, the empty set is not the value that one would expect as in particular it does not allow to capture serial composition.

3.5 Separating input assumptions

In the relational interface framework of [16], input assumptions and output guarantees are combined into a single contract ϕ. One idea is to separate them. Following this idea, an interface would be a quadruple

$$I = (X, Y, \phi, \psi)$$

where X is a finite set of input variables, Y is a finite set of output variables (as usual, we assume that $X \cap Y = \emptyset$), ϕ is a relation/predicate on X, and ψ is a relation/predicate on $X \cup Y$. ϕ captures the requirements on inputs only, while ψ is aimed at capturing guarantees on the outputs, with relation to inputs.

Ideally, we would like to have no redundancy between ϕ and ψ, for example, ψ should not impose more restrictions on the inputs than what ϕ imposes, as for example in the case $\phi := x > 0$, $\psi := x > 1 \wedge y > x$. One way to achieve this which appears "canonical" is to attempt to enforce that ψ be *total* (i.e., input-complete), that is, to enforce $\mathbf{in}(\psi) \equiv \mathbf{true}$. As we shall see below, however, this requirement is not always compatible with our other desiderata.

Let us see first how the definitions of refinement and composition could be adapted to this setting.

Refinement. Refinement can be defined as follows. Let $I_i = (X, Y, \phi_i, \psi_i)$, for $i = 1, 2$. Then, I_2 refines I_1, written $I_2 \sqsubseteq I_1$, iff

$$\phi_1 \to \phi_2 \qquad \text{and} \qquad (\phi_1 \wedge \psi_2) \to \psi_1$$

are both valid formulas, i.e., equivalent to **true**.

With contract pairs, the refinement order gives a lattice, with top being the pair of contracts $(\mathbf{false}, \mathbf{true})$ and bottom being the pair $(\mathbf{true}, \mathbf{false})$.

Parallel composition. Parallel composition of interfaces with contract pairs can be defined as usual, by taking the conjunction of their corresponding contracts. Let $I_i = (X_i, Y_i, \phi_i, \psi_i)$, for $i = 1, 2$, where all sets X_1, X_2, Y_1, Y_2 are pair-wise disjoint. Then

$$I_1 || I_2 := (X_1 \cup X_2, Y_1 \cup Y_2, \phi_1 \wedge \phi_2, \psi_1 \wedge \psi_2).$$

Note that $\phi_1 \wedge \phi_2$ is over $X_1 \cup X_2$, and $\psi_1 \wedge \psi_2$ is over $X_1 \cup X_2 \cup Y_1 \cup Y_2$.

Also note that

$$\begin{aligned}\mathbf{in}(\psi_1 \wedge \psi_2) &\equiv \exists Y_1, Y_2 : \psi_1 \wedge \psi_2 &&\equiv \exists Y_1 : (\psi_1 \wedge \exists Y_2 : \psi_2)\\ &\equiv (\exists Y_2 : \psi_2) \wedge (\exists Y_1 : \psi_1) &&\equiv \mathbf{in}(\psi_1) \wedge \mathbf{in}(\psi_2)\end{aligned}$$

Thus, if $\mathbf{in}(\psi_1) \equiv \mathbf{in}(\psi_2) \equiv \mathbf{true}$, we also have $\mathbf{in}(\psi_1 \wedge \psi_2) \equiv \mathbf{true}$, which means that parallel composition preserves our desiderata of no redundancy between ϕ and ψ.

Feedback. Feedback can be defined as follows. Let $I = (X, Y, \phi, \psi)$ and let $x \in X$ and $y \in Y$. Then connecting output y to input x yields the new interface

$$\mathit{feedback}_{y \rightsquigarrow x}(I) \quad := \quad (X - \{x\}, Y \cup \{x\}, \phi', \psi')$$

where

$$\phi' := \exists x : \Big(\phi \wedge \big(\forall Y, x : (\psi \wedge x = y) \to \phi\big)\Big)$$

and

$$\psi' := \psi \wedge x = y.$$

We can immediately see that ψ' is indeed a predicate over $Y \cup \{x\}$.

$\forall Y, x : (\psi \wedge x = y) \to \phi$ is a predicate over $X - \{x\}$. Therefore, ϕ' is equivalent to

$$(\exists x : \phi) \wedge \big(\forall Y, x : (\psi \wedge x = y) \to \phi\big).$$

and, since both conjuncts above are predicates over $X - \{x\}$, we can now see that ϕ' is a predicate over $X - \{x\}$. The idea behind the definition of ϕ' is to capture the notion of demonic non-determinism, as in the definition of serial composition, by strengthening the original input requirements ϕ with the additional term $\forall Y, x : (\psi \wedge x = y) \to \phi$.

Does this definition of feedback allow to reduce serial composition to parallel composition followed by feedback? This indeed appears to work on simple examples.

Example 7. Consider the interfaces defined in Example 1, $I_1 = (\{x\}, \{y\}, x \le y)$, $I_2 = (\{y\}, \{z\}, y \ne 0)$, and their serial composition

$$I_1 \leadsto I_2 = (\{x\}, \{y, z\}, x \le y \wedge y \ne 0 \wedge x > 0).$$

Transforming I_1 and I_2 into interfaces with pairs of contracts, we get $I_1' = (\{x\}, \{y\}, \mathbf{true}, x \le y)$ and $I_2' = (\{y'\}, \{z\}, y' \ne 0, \mathbf{true})$, where we have also renamed y to y' in I_2'. We can now form the parallel composition of I_1' and I_2':

$$I_1' || I_2' = (\{x, y'\}, \{y, z\}, y' \ne 0, x \le y)$$

and then connect y to y' in feedback, to obtain:

$$\mathit{feedback}_{y \leadsto y'}(I_1 || I_2) = (\{x\}, \{y, y', z\}, \phi', x \le y \wedge y = y')$$

where

$$\phi' \quad := \quad (\exists y' : y' \ne 0) \wedge (\forall y, y', z : x \le y \wedge y = y' \to y' \ne 0)$$

which is equivalent to $x > 0$. Therefore, after eliminating y' which is equal to y in $\mathit{feedback}_{y \leadsto y'}(I_1 || I_2)$, the latter simplifies to

$$\mathit{feedback}_{y \leadsto y'}(I_1 || I_2) = (\{x\}, \{y, z\}, x > 0, x \le y)$$

which, as it can be seen, is the same as $I_1 \leadsto I_2$, except that the single contract is replaced by a contract pair. □

The above definition of feedback also appears to resolve the problem of non-preservation of refinement described in Example 2:

Example 8. Let $I = (\{x, z\}, \{y\}, \mathbf{true}, \mathbf{true})$ and $I' = (\{x, z\}, \{y\}, \mathbf{true}, x \ne y)$. Then $I' \sqsubseteq I$. Also, $\mathit{feedback}_{y \leadsto x}(I) = (\{z\}, \{y, x\}, \mathbf{true}, x = y)$ and $\mathit{feedback}_{y \leadsto x}(I') = (\{z\}, \{y, x\}, \mathbf{true}, \mathbf{false})$. As it can be seen, $\mathit{feedback}_{y \leadsto x}(I') \sqsubseteq \mathit{feedback}_{y \leadsto x}(I)$. □

Note that, as Example 8 demonstrates, the canonical form requirement for interfaces with contract pairs, namely that $\mathbf{in}(\psi)$ must be $\mathbf{true}$, is not generally preserved by feedback composition.

Summary. As it can be seen, interfaces with pairs of contracts seem to resolve several issues that exist in interfaces with single contracts. On the other hand, the interpretation of interfaces with contract pairs is unclear, and some new problems are introduced. First, regarding interpretation, it is unclear what the meaning of interfaces such as $(\mathbf{false}, \mathbf{true})$ and $(\mathbf{true}, \mathbf{false})$ is, and what the difference between the two is. The standard interpretation is that in $(\mathbf{false}, \mathbf{true})$ all inputs are illegal, and that in $(\mathbf{true}, \mathbf{false})$ all inputs are legal, but no output is produced. We find these interpretations unsatisfactory. What is the meaning of a "legal" input if no output can be produced? And why distinguish the contract $(\mathbf{false}, \mathbf{true})$ from, say, $(\mathbf{false}, y \geq 0)$? After all, both accept no inputs, therefore, they cannot be used in any useful composition. In addition, interfaces with at least one contract being **false** do not seem to have valid implementations (say, by deterministic state machines).

Perhaps the most important problem with pairs of contracts is the fact that refinement does not preserve the "well-formedness" property that none of the two contracts be **false**, and therefore does not preserve implementability as discussed above. For instance, $(\mathbf{true}, \mathbf{false})$ refines every pair of contracts. This means that we could start from an implementable (well-formed) specification and reach a non-implementable one by successive refinements. This is clearly undesirable.

4 Refinement-Preserving Feedback

In this section a definition of feedback composition, derived from a set of desiderata, will be proposed for general relational interfaces. Given the interface

$$I := (\{\bar{u},\, \bar{v}\},\, \{\bar{x},\, \bar{y}\}, \phi)$$

where the barred variables indicate sequences of individual variables, and specifically, $\bar{x}$ is of the same length as $\bar{u}$, feedback composition will be defined

$$\mathit{feedback}_{\bar{x} \rightsquigarrow \bar{u}}(I) := (\{\bar{v}\},\, \{\bar{x},\, \bar{y}\}, \phi^*)$$

eliminating the set of inputs $\bar{u}$ from the signature of the interface. The definition of the new relation ϕ^* is what will be determined here from the desiderata. For brevity the interface will be expressed in terms of its relation, such as ϕ here for I. Sometimes the variables will be given with the relation with the input and output variables separated by a semicolon, as in $\phi(\bar{u},\, \bar{v};\, \bar{x},\, \bar{y})$. Also, unless otherwise noted, all interfaces will have this same general signature, and the asterisk, as in ϕ^* will indicate the feedback composition $\mathit{feedback}_{\bar{x} \rightsquigarrow \bar{u}}(I)$ (and its constituting relation).

For a relational interface characterized by a total function, the concept of a feedback connection between an output and an input can be defined straightforwardly by a fixed-point relation. Specifically, given a functional interface

$$f(\bar{u},\, \bar{v};\, \bar{x},\, \bar{y})$$

feedback composition can be defined as

$$f^*(\bar{v};\, \bar{x},\, \bar{y}) \leftrightarrow f(\bar{x},\, \bar{v};\, \bar{x},\, \bar{y}) \tag{1}$$

meaning that, for given input $\bar{v}$, $\bar{x}$ is an output of the feedback interface f^* iff $\bar{x}$ is a fixed-point of f.[5] A general definition for feedback over all relations therefore can be constrained to at least reduce to this one in the case of the relation being a graph of a total function; that is both input complete and deterministic. The formula 1 will then be the first desideratum.

A second qualification desirable for a feedback definition is that it preserves refinement relations in the same manner as serial and parallel composition. Given two interfaces ϕ and ψ with the same signature

$$\psi \sqsubseteq \phi \rightarrow \psi^* \sqsubseteq \phi^* \tag{2}$$

In other words, feedback composition is monotonic under the refinement order. This qualification is consistent with the first vacuously, since functional interfaces are minimal in the refinement order. The formula 2 will be the second desideratum.

A third qualification should then be given to determine the possible values taken on by the feedback edges of a feedback composition. The obvious possibility is to make the set of values some subset of the fixed-points of the relation between the connected output and input variables. In formal terms

$$\phi^*(\bar{v};\, \bar{x},\, \bar{y}) \rightarrow \phi(\bar{x},\, \bar{v};\, \bar{x},\, \bar{y}) \tag{3}$$

Here, the difference between the case for functional relations and general relations is that the feedback values for the former are exactly the fixed-points, whereas the latter could reject certain fixed points. Outside of fixed points, another possible choice for feedback values might be values that have some finite orbit through the relation. However, these points would have to be included in desideratum 1 for total functional relations, hence for consistency with this first qualification, this expanded set will not be considered. The formula 3 will be the third desideratum.

Using these three desiderata, building off of the third 3, a definition for feedback composition can be postulated. This definition will be of the form

$$\phi^*(\bar{v};\, \bar{x},\, \bar{y}) := \phi(\bar{x},\, \bar{v};\, \bar{x},\, \bar{y}) \wedge \textit{additional constraints} \tag{4}$$

where the *additional constraints* remove certain fixed-points. Clearly, based on 1, these constraints must reduce to **true** in the case of total functional relations. These constraints will be deduced in the following by considering particular cases of relations.

[5] Note that 1 says that *all* fixed-points of f must be solutions of the feedback f^*. This is different from the semantics of deterministic synchronous formalisms such as Esterel [2] or synchronous block diagrams [10], where feedback is defined by choosing from the set of fixed-points a unique representative, namely the *least* fixed-point. The least fixed-point solution relies on f being defined on an ordered set structure such as a complete partial order. We make no such assumption here.

4.1 Partiality

Consider first the case of the free inputs $\bar{v}$ in the formula being fixed to a particular value $\mathbf{a}$, and the remaining relation

$$\phi(\bar{u},\, \mathbf{a};\, \bar{x},\, \bar{y})$$

being *partial and deterministic* with respect to the input $\bar{u}$. Then, suppose the following relation is defined:

$$\psi(\bar{u},\, \mathbf{a};\, \bar{x},\, \bar{y}) := \phi(\bar{u},\, \mathbf{a};\, \bar{x},\, \bar{y}) \vee \big[\big(\neg\exists \bar{z}, \bar{w} :\, \phi(\bar{u},\, \mathbf{a};\, \bar{z},\, \bar{w})\big) \wedge \bar{u} = \bar{x} \wedge \bar{y} = \mathbf{b}\big]$$

where $\mathbf{b}$ is some arbitrary chosen constant tuple of the same length as $\bar{y}$. It can be verified that $\psi \sqsubseteq \phi$, intuitively, because ψ simply gives output values for inputs not in the domain of ϕ, thus refining it. Moreover, by the definition of ψ and the assumption that ϕ is deterministic, ψ returns a unique value for all input values. Thus, ψ is a total function. By desideratum 1, it follows that

$$\psi^*(\mathbf{a};\, \bar{x},\, \bar{y}) \leftrightarrow \psi(\bar{x},\, \mathbf{a};\, \bar{x},\, \bar{y}).$$

Using the above definition, along with 3, we obtain

$$\phi^*(\mathbf{a};\, \bar{x},\, \bar{y}) \rightarrow \psi^*(\mathbf{a};\, \bar{x},\, \bar{y}).$$

The only way the above and 2 can both be true is if

$$\phi^*(\mathbf{a};\, \bar{x},\, \bar{y}) := \mathbf{false}.$$

What can be concluded from this is that feedback composition over a relation that is partial with respect to the feedback input must exclude all fixed points. This can be accomplished by conjoining the following constraint to the definition for feedback composition

$$\forall \bar{u} :\, \exists \bar{x}, \bar{y} :\, \phi(\bar{u},\, \bar{v};\, \bar{x},\, \bar{y}).$$

4.2 Nondeterminism

Consider first the case of the free inputs $\bar{v}$ in the formula being fixed to a particular value $\mathbf{a}$, and the remaining relation

$$\phi(\bar{u},\, \mathbf{a};\, \bar{x},\, \bar{y})$$

being *total* or complete with respect to the input $\bar{u}$, but also being nondeterministic. Let

$$\hat{\phi}(\bar{u},\, \bar{v}) := \{(\bar{x},\, \bar{y}) \,|\, \phi(\bar{u},\, \bar{v};\, \bar{x},\, \bar{y})\}$$

denote the set of output values for an interface corresponding to the given input values.

Since ϕ is total, every refinement ψ will be total as well. More, it follows from the definition of refinement that

$$\hat{\psi}(\bar{u},\, \bar{v}) \subseteq \hat{\phi}(\bar{u},\, \bar{v}).$$

Suppose then that a particular relation ψ is defined such that $\hat{\psi}(\bar{u},\, \bar{v})$ is a singleton for every set of input values. Moreover, if $\hat{\phi}(\bar{u},\, \bar{v})$ contains more than one choice for the value of $\bar{x}$, the feedback output, let the choice of singleton value be the one where $\bar{x} \neq \bar{u}$; this is always possible if such a choice exists.

The above construction for ψ gives a refinement of ϕ for the above given reasons. This construction is also total functional, since a unique value was chosen for every input. By desideratum 1, the feedback values for $\bar{x}$ in ψ^* are exactly the fixed point solutions for $\bar{x}$ in ψ. However, if there are no such fixed point solutions for ψ, and consequently no feedback values for ψ^*, by desideratum 2, ϕ^* can have no feedback values for $\bar{x}$; that is, ϕ^* would have to be **false**. On the other hand, by the definition of refinement, for any set of inputs to a relation, if the set of outputs is unique, it must be unique in every refinement for the same inputs, thus the fixed point solutions that are also unique outputs for their corresponding inputs are preserved by refinement. The construction for ψ removes all of the other fixed point solutions from ϕ, arising from nondeterministic inputs.

What can be concluded from this is that the feedback composition, to be consistent with the desiderata, must be false unless there is at least one fixed point solution that is a unique output value for its corresponding input. This constraint can be formulated as the following term

$$\exists \bar{z} :\ \phi(\bar{z},\, \bar{v};\, \bar{z},\, \bar{y}) \wedge (\forall \bar{w} :\ \phi(\bar{w},\, \bar{v};\, \bar{w},\, \bar{y}) \rightarrow \bar{w} = \bar{z}) \tag{5}$$

which can be conjoined with the feedback composition definition. A simpler term to conjoin would be one that constrains the feedback values to only be deterministic ones; ones that are the unique outputs for their corresponding input values. This term would be

$$\forall \bar{z} :\ \phi(\bar{z},\, \bar{v};\, \bar{z},\, \bar{y}) \rightarrow \bar{x} = \bar{z}. \tag{6}$$

From the perspective of the desiderata, this latter definition would make an unnecessary restriction, but nevertheless it would simplify the definition for feedback composition considerably. The decision to use the former or the latter would hinge on the presence of additional desiderata, perhaps regarding the preservation of feedback values by refinement; clearly, it is necessary that at least one feedback value should be preserved for others to exist, and the important question should be whether or not they all should be.

4.3 General Feedback Composition

Combining the above considerations and assembling the corresponding constraints, the following two definitions may be postulated for feedback composition on relational interfaces:

$$\begin{aligned}\phi^*(\bar{v};\,\bar{x},\,\bar{y}) := {} & \phi(\bar{x},\,\bar{v};\,\bar{x},\,\bar{y}) \\ & \wedge [\forall \bar{u} :\, \exists \bar{x}, \bar{y} :\, \phi(\bar{u},\,\bar{v};\,\bar{x},\,\bar{y})] \\ & \wedge [\exists \bar{z} :\, \phi(\bar{z},\,\bar{v};\,\bar{z},\,\bar{y}) \wedge (\forall \bar{w} :\, \phi(\bar{w},\,\bar{v};\,\bar{w},\,\bar{y}) \rightarrow \bar{w} = \bar{z})]\end{aligned} \tag{7}$$

$$\begin{aligned}\phi^*(\bar{v};\,\bar{x},\,\bar{y}) := {} & \phi(\bar{x},\,\bar{v};\,\bar{x},\,\bar{y}) \\ & \wedge [\forall \bar{u} :\, \exists \bar{x}, \bar{y} :\, \phi(\bar{u},\,\bar{v};\,\bar{x},\,\bar{y})] \\ & \wedge [\forall \bar{z} :\, \phi(\bar{z},\,\bar{v};\,\bar{z},\,\bar{y}) \rightarrow \bar{x} = \bar{z}]\end{aligned} \tag{8}$$

The full ramifications of these definitions of feedback warrant much further investigation. The property that the two definitions reduce to only the first term, the fixed-point relation, in the case of total functional relations means that at least, for the subclass of total functional relations (or simply functions), this definition is consistent with the usual notion of feedback in deterministic synchronous models of computation.

As an example, the above definitions both applied to the one input, one output **true** interface result in the one output **false** interface, consistent with the earlier example demonstrating that this must be the case. Suppose, instead, that a relation were defined

$$\phi(x;\,y) := x = 5 \rightarrow y = 5$$

which is similar to **true**, except on the input $x := 5$, which must be mapped deterministically to $y := 5$. Then, under the two definitions of feedback, the corresponding compositions would be

$$\phi^*(\cdot;\,y) := \textbf{true}$$

and

$$\phi^*(\cdot;\,y) := y = 5$$

In both cases, the relation $x \neq y$ is not a refinement. Indeed, every refinement of both must at least include 5 as a feedback output value.

5 Conclusions

The definition of feedback composition in the context of synchronous relational interfaces has been investigated. Challenges were described in Section 3, and a systematic derivation of novel alternatives was proposed in Section 4. Future work includes a more thorough study of these new alternatives in the context of the full theory presented in [16], as well as in the context of recent work on *error-completion* [17]. In addition, it would be interesting to examine how the difficulties in defining feedback in synchronous interfaces are related to the problem of compositionality of relational semantics of non-deterministic dataflow networks and the so-called *Brock-Ackerman anomaly* [6,12].

References

1. S. Bensalem, M. Bozga, T.-H. Nguyen, and J. Sifakis. D-finder: A tool for compositional deadlock detection and verification. In *Computer Aided Verification*, pages 614–619. Springer, 2009.
2. G. Berry. *The foundations of Esterel*, pages 425–454. MIT Press, 2000.
3. S. Bliudze and J. Sifakis. The algebra of connectors: structuring interaction in bip. In *EMSOFT'07*, pages 11–20. ACM, 2007.
4. S. Bliudze and J. Sifakis. A notion of glue expressiveness for component-based systems. In *CONCUR – Concurrency Theory*, pages 508–522. Springer, 2008.
5. B. Bonakdarpour, M. Bozga, M. Jaber, J. Quilbeuf, and J. Sifakis. A framework for automated distributed implementation of component-based models. *Distributed Computing*, 25(5):383–409, 2012.
6. J.D. Brock and W.B. Ackerman. Scenarios: A model of non-determinate computation. In *Formalization of Programming Concepts*. Springer, 1981.
7. L. de Alfaro and T. Henzinger. Interface automata. In *Foundations of Software Engineering (FSE)*. ACM Press, 2001.
8. L. de Alfaro and T. Henzinger. Interface theories for component-based design. In *EMSOFT'01*. Springer, LNCS 2211, 2001.
9. L. Doyen, T. Henzinger, B. Jobstmann, and T. Petrov. Interface theories with component reuse. In *EMSOFT'08*, pages 79–88, 2008.
10. S. Edwards and E. Lee. The semantics and execution of a synchronous block-diagram language. *Science of Computer Programming*, 48:21–42(22), July 2003.
11. G. Gössler and J. Sifakis. Composition for component-based modeling. *Science of Computer Programming*, 55(1):161–183, 2005.
12. B. Jonsson. A fully abstract trace model for dataflow and asynchronous networks. *Distributed Computing*, 7(4):197–212, 1994.
13. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, S. Bensalem, and D. Probst. Property preserving abstractions for the verification of concurrent systems. *Formal methods in system design*, 6(1):11–44, 1995.
14. M. Poulhies, J. Pulou, C. Rippert, and J. Sifakis. A methodology and supporting tools for the development of component-based embedded systems. In *Composition of Embedded Systems. Scientific and Industrial Issues*, pages 75–96. Springer, 2007.
15. J. Sifakis and S. Yovine. Compositional specification of timed systems. In *STACS 96*, pages 345–359. Springer, 1996.
16. S. Tripakis, B. Lickly, T. A. Henzinger, and E. A. Lee. A theory of synchronous relational interfaces. *ACM TOPLAS*, 33(4), July 2011.
17. S. Tripakis, C. Stergiou, M. Broy, and E. A. Lee. Error-Completion in Interface Theories. In *SPIN 2013*, volume 7976 of *LNCS*. Springer, 2013.