Interface Theories with Component Reuse*

Laurent Doyen
EPFL, Switzerland
laurent.doyen@epfl.ch

Thomas A. Henzinger
EPFL, Switzerland
tah@epfl.ch

Barbara Jobstmann
EPFL, Switzerland
barbara.jobstmann@epfl.ch

Tatjana Petrov
EPFL, Switzerland
tatjana.petrov@epfl.ch

ABSTRACT

Interface theories have been proposed to support incremental design and independent implementability. Incremental design means that the compatibility checking of interfaces can proceed for partial system descriptions, without knowing the interfaces of all components. Independent implementability means that compatible interfaces can be refined separately, maintaining compatibility. We show that these interface theories provide no formal support for component reuse, meaning that the same component cannot be used to implement several different interfaces in a design. We add a new operation to interface theories in order to support such reuse. For example, different interfaces for the same component may refer to different aspects such as functionality, timing, and power consumption. We give both stateless and stateful examples for interface theories with component reuse. To illustrate component reuse in interface-based design, we show how the stateful theory provides a natural framework for specifying and refining PCI bus clients.

Categories and Subject Descriptors

D.2.12 [**Software Engineering**]: Interoperability—*Interface definition languages*

General Terms

Design, Theory

Keywords

Interfaces, Composition, Refinement

1. INTRODUCTION

Interface theories [5] are intended to provide a formal framework for component-based design. An *interface* constrains the interaction of a component with its environment, i.e., with the other components in a system. An interface (φ, ψ) captures both an *assumption* φ that the component makes about the environment, and a *guarantee* ψ that the component provides to the environment. Two interfaces $I_1 = (\varphi_1, \psi_1)$ and $I_2 = (\varphi_2, \psi_2)$ are *compatible* if there is a context in which I_1 and I_2 satisfy each other's assumptions, and the weakest condition on the environment to have I_1 and I_2 fit together — roughly speaking, $(\psi_1 \wedge \psi_2) \rightarrow (\varphi_1 \wedge \varphi_2)$ — is the assumption of the composition $I_1 \| I_2$ (the guarantee of the composition is $\psi_1 \wedge \psi_2$).

Interfaces support stepwise refinement. An interface $I' = (\varphi', \psi')$ refines an interface $I = (\varphi, \psi)$ if in every context, the interface I can be replaced by the more detailed version I'. Formally, this means that if $I' \preceq I$ (denoting that I' refines I), then $I' \| J \preceq I \| J$ for all interfaces J that are also compatible with I. For this to happen, the refined interface I' cannot make any stronger assumption about the environment than I (i.e., $\varphi \rightarrow \varphi'$), and I' cannot provide any weaker guarantee to the environment than I (i.e., $\psi' \rightarrow \psi$). This contravariant refinement can be found in subtyping relations for function types [10]. Indeed, function types are interfaces whose assumptions and guarantees constrain the data values of inputs and outputs. More expressive interface theories have been developed for settings where the assumptions and guarantees constrain the protocol aspect of the interaction of a component with the environment (so-called *interface automata* [4, 8]), the timing of inputs and outputs [6], the power consumption of components [3], the trade-off between throughput and resource needs [11, 12, 7], etc. A main strength of all these interface theories is that they not only provide algorithms for checking interface compatibility and refinement, but that the compatibility check computes the weakest requirement on the inputs, timing, power, or computational resources provided by the environment which makes two or more interfaces fit together. In this way, interface theories provide important information about a partial design to the designer.

A second main strength of interface theories is that refinement supports the *independent implementability* of interfaces. To see this, consider Figure 1. Suppose the top-level interface is decomposed into two interacting component interfaces, I_1 and I_2. The interface I_1 is refined into three component interfaces, I_{11}, I_{12}, and I_{13}. Independently, possibly by a different design team or supplier, the interface I_2 is refined into two component interfaces, I_{21} and I_{22}. The compatibility of I_1 and I_2 ensures that also the refined versions $I_{11} \| I_{12} \| I_{13}$ and $I_{21} \| I_{22}$ are compatible. Similarly, if I_{11} is further refined into $I_{111} \| I_{112}$, then all components in the

*This research was supported by the Swiss National Science Foundation and by the European COMBEST project.

EMSOFT'08, October 19–24, 2008, Atlanta, Georgia, USA.

entire design $I_{111}\|I_{112}\|I_{12}\|I_{13}\|I_{21}\|I_{22}$ are guaranteed to fit together. Design, however, rarely proceeds in such a strict, tree-like, top-down manner. Often design involves the use of already available components. Also, for space or cost reasons, different logical parts of a design may have to share a common implementation. In Figure 1, suppose the interfaces I_{22} and I_{112} (which are at different levels in different parts of the design tree) are similar enough that they should be implemented by the same component. Interface theories do not provide for this possibility. In this paper, we extend interface theories to support such *shared implementability* of interfaces. In other words, within our extended theories, components can be reused in different parts of a design (or in different designs); designs are not restricted to be trees of components, but they can be DAGs.

Formally, we say that two interfaces $I_1 = (\varphi_1, \psi_1)$ and $I_2 = (\varphi_2, \psi_2)$ are *shared refinable* if there is an interface that refines both I_1 and I_2. If I_1 and I_2 are shared refinable, then we compute the *shared refinement* $I_1 \sqcap I_2$ as the most general refinement of I_1 and I_2, i.e., the greatest lower bound in the refinement lattice on interfaces: the assumption of $I_1 \sqcap I_2$ is $\varphi_1 \vee \varphi_2$, and the guarantee of $I_1 \sqcap I_2$ is $\psi_1 \wedge \psi_2$. A component can be used to implement both I_1 and I_2 iff it refines $I_1 \sqcap I_2$. Notice that such a component must be prepared to accept inputs that satisfy any of the two assumptions φ_1 and φ_2, and it must provide outputs that satisfy both guarantees ψ_1 and ψ_2. Interestingly, the shared refinement $I_1 \sqcap I_2$ can also be used to implement two or more different views (or aspects) of a single component. For example, the interface I_1 may provide functional constraints (i.e., assumptions and guarantees) and the interface I_2 may provide power constraints on the same component. For the component to satisfy both the constraints specified by I_1 and the constraints specified by I_2, it must refine $I_1 \sqcap I_2$. Note that this is different from refining the composition $I_1\|I_2$: while the composition $I_1\|I_2$ has the assumption $\varphi_1 \wedge \varphi_2$, the shared refinement has the assumption $\varphi_1 \vee \varphi_2$.

In Section 2, we formally develop a simple, stateless interface theory with shared refinement along the lines outlined above. The formalism becomes far more interesting and powerful once we introduce a temporal aspect in the form of state. Then, assumptions and guarantees, which may change from state to state, can be specified by an automaton. To check if two such automata are compatible, we need to solve a game where the environment, which provides inputs to the two interacting automata, must have a strategy that avoids incompatibilities [4, 8]. The most general such strategy defines the composite interface automaton. Refinement between interface automata is an alternating simulation relation [1], which ensures that every winning environment strategy of the more abstract automaton is inherited by the more detailed automaton. In Section 3, we add shared refinement to such a stateful theory of interfaces. For technical simplicity, we choose to present shared refinement not for the original asynchronous theory of interface automata [4], but for a synchronous version, which was used in [2] to specify interfaces for PCI bus components.

As is to be expected from the stateless case, shared refinement is different from parallel composition also in the automaton case. The standard parallel composition of two automata with n_1 and n_2 states is the product automaton with $n_1 n_2$ states. The parallel composition of two interface automata is a pruned product automaton, from which states

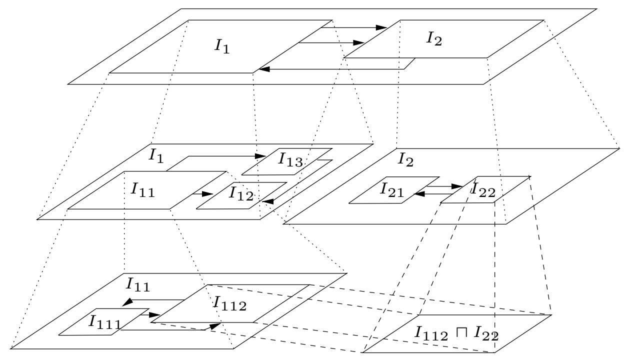

Figure 1: Top-down design with shared refinement.

without a winning environment strategy are removed. By contrast, the shared refinement of two interface automata with n_1 and n_2 states is an extended product automaton with up to $n_1 n_2 + n_1 + n_2$ states. Roughly speaking, as long as the assumptions of both automata are satisfied, the shared refinement provides the guarantees of both automata. But as soon as the assumption of one automaton is violated, only the guarantee of the other automaton is maintained. We believe that this automata-theoretic construction is interesting in its own right, for example, to combine two different views of a system. We illustrate its use in Section 4, where we provide a refined version of the PCI bus example from [2]. In particular, we provide a functional interface F_1 and a power interface P for a PCI bus client, as well as a functional interface F_2 for the client of a different bus. Then we show that $F_1 \sqcap P$ and $F_2 \sqcap P$ are shared refinable. In other words, the client specifications for both busses can be implemented by the same component, and the requirement on such a component is that it refines the shared refinement $F_1 \sqcap F_2 \sqcap P$.

2. STATELESS INTERFACES

A stateless interface for a component in a design describes the environments in which the component can be embedded [4, 8]. It has input and output variables, and two predicates restricting their values: the first predicate specifies the set of input values that the component should accept, the second predicate specifies the set of output values that the component may produce.

Definition 1 (stateless interface). *A* stateless interface *is a tuple $M = \langle X^I, X^O, \varphi, \psi \rangle$, where*

- *X^I and X^O are two disjoint sets of input and output variables, respectively,*
- *φ is a predicate over X^I called* input assumption*, and*
- *ψ is a predicate over X^O called* output guarantee.

We require that an interface accepts at least one input value and produces at least one output value.

Definition 2 (well-formedness). *A stateless interface $M = \langle X^I, X^O, \varphi, \psi \rangle$ is* well-formed *if*

(1) *φ is satisfiable and*

(2) *ψ is satisfiable.*

Refinement.

The refinement relation for interfaces is such that whenever an interface N refines an interface M, then M can be replaced by N in every design that provides inputs satisfying the assumption of M and expects outputs satisfying the guarantee of M. Hence, the refining interface N has to accept at least the same inputs as M, and may produce a subset of the possible outputs of M.

DEFINITION 3. *Given two well-formed stateless interfaces $M = \langle X_M^I, X_M^O, \varphi_M, \psi_M\rangle$ and $N = \langle X_N^I, X_N^O, \varphi_N, \psi_N\rangle$, we say that N* refines *M, written $N \preceq M$, if*

(1) $(X_M^I \cup X_N^I) \cap (X_M^O \cup X_N^O) = \emptyset$,

(2) $\varphi_M \to \varphi_N$, *and*

(3) $\psi_N \to \psi_M$.

The classical theory of interfaces [4, 8] includes two operations to compose stateless interfaces:

- The *connection* allows to connect output variables to input variables of an interface.
- The *parallel composition* describes how to obtain the interface of a component that combines two or more sub-components running in parallel.

We recall the definition and properties of these two operations, and we introduce the new *shared refinement*, a binary operation that gives the most general interface refining two given interfaces.

Connection.

A connection consists of a set of pairs of variables defining which variables are connected when the connection is applied to an interface. The first component of each pair in a connection is an output variable, and the second an input variable to which the output is connected.

DEFINITION 4 (CONNECTION). *A* connection *θ is a set of pairs (x, y) of variables, consisting of a* source *variable x and a* target *variable y, such that for all pairs $(x, y), (x', y') \in \theta$, if $x \neq x'$, then $y \neq y'$.*

We denote the set of source variables of θ by $\mathcal{S}_\theta$ and the set of target variables by $\mathcal{T}_\theta$. The predicate ρ_θ denotes $\bigwedge_{(x,y)\in\theta}(x = y)$.

A connection is compatible with an interface M, if (1) its source variables are all output variables of M, (2) its target variables are all input variables of M, and (3) when the source variables are connected to the target variables (i.e., ρ_θ holds), there exist values of the remaining input variables of M such that the assumption of M is satisfied for all values of the output variables of M that satisfy the guarantee of M.

DEFINITION 5 (COMPATIBILITY FOR CONNECTION). *A stateless interface $M = \langle X^I, X^O, \varphi, \psi\rangle$ is* compatible *with a connection θ if the following conditions hold:*

(1) $\mathcal{S}_\theta \subseteq X^O$,

(2) $\mathcal{T}_\theta \subseteq X^I$, *and*

(3) *the predicate $\hat{\varphi} \equiv \forall X^O \cdot \forall \mathcal{T}_\theta \cdot ((\psi \wedge \rho_\theta) \to \varphi)$ is satisfiable.*

If M is compatible with θ, then the result of applying θ to M is the stateless interface $M\theta = \langle \hat{X}^I, \hat{X}^O, \hat{\varphi}, \hat{\psi}\rangle$, where

- $\hat{X}^I = X^I \setminus \mathcal{T}_\theta$,
- $\hat{X}^O = X^O \cup \mathcal{T}_\theta$, *and*
- $\hat{\psi} \equiv (\psi \wedge \rho_\theta)$.

THEOREM 1. *Let M be a well-formed stateless interface and θ be a connection. If M is compatible with θ, then $M\theta$ is a well-formed stateless interface.*

The connection supports independent implementability. Given a design $M\theta$ and an interface N that refines M, we can replace M by N in the design $M\theta$, independently of θ, because $N\theta$ refines $M\theta$.

THEOREM 2. *Let M and N be two well-formed stateless interfaces and θ be a connection. If $N \preceq M$ and M is compatible with θ, then N is compatible with θ and $N\theta \preceq M\theta$.*

Parallel composition.

Parallel composition allows to combine interfaces that are compatible with each other. Two interfaces are compatible for parallel composition if (1) the sets of output variables are disjoint, (2) the input variables of each interface are disjoint from the output variables of the other interface, and (3) the conjunction of the guarantees is satisfiable.

DEFINITION 6 (PARALLEL COMPOSITION). *Two stateless interfaces $M = \langle X_M^I, X_M^O, \varphi_M, \psi_M\rangle$ and $N = \langle X_N^I, X_N^O, \varphi_N, \psi_N\rangle$ are* compatible for parallel composition*, written $M \approx N$, if*

(1) $X_M^O \cap X_N^O = \emptyset$,

(2) $X_M^I \cap X_N^O = \emptyset$, $X_N^I \cap X_M^O = \emptyset$, *and*

(3) $\varphi_M \wedge \varphi_N$ *is satisfiable.*

If $M \approx N$ holds, then the parallel composition is the stateless interface $M \| N = \langle \hat{X}^I, \hat{X}^O, \hat{\varphi}, \hat{\psi}\rangle$, where

- $\hat{X}^I = X_N^I \cup X_M^I$,
- $\hat{X}^O = X_N^O \cup X_M^O$,
- $\hat{\varphi} \equiv (\varphi_N \wedge \varphi_M)$, *and*
- $\hat{\psi} \equiv (\psi_N \wedge \psi_M)$.

THEOREM 3. *Let M and N be two well-formed stateless interfaces. If $M \approx N$, then $M \| N$ is a well-formed stateless interface.*

The parallel composition supports independent implementability. In a design $M \| S$, we can replace M by any N that refines M if all variables common to N and S are also variables of M, because then $N \| S$ refines $M \| S$. Intuitively, the new variables of N should not conflict with variables of the design S.

THEOREM 4. *Let M, N, and S be three well-formed stateless interfaces such that $X_N \cap X_S \subseteq X_M$. If $M \approx S$ and $N \preceq M$, then $N \approx S$ and $N \| S \preceq M \| S$.*

Shared refinement.

The shared refinement allows to describe the interface of a component that is meant to work in two or more environments based on separate descriptions of each environment. Note that different environment descriptions may use different variable names. Since it is the choice of the user to decide which variables are going to be shared, we assume that the interface variables are renamed accordingly before the shared refinement is applied.

The shared refinement of two interfaces needs to be able to replace each of the given interfaces. Therefore, it has to refine both interfaces. In order to ensure that the combined interface is well-formed, we introduce the notion of shared refinability. Two interfaces are shared refinable if (1) the input variables do not overlap with the output variables and (2) their output guarantees do not contradict each other.

Definition 7 (shared refinement). *Two stateless interfaces $M = \langle X_M^I, X_M^O, \varphi_M, \psi_M\rangle$ and $N = \langle X_N^I, X_N^O, \varphi_N, \psi_N\rangle$ are* shared refinable, *written $M \sim N$, if*

(1) $(X_M^I \cup X_N^I) \cap (X_M^O \cup X_N^O) = \emptyset$ *and*

(2) $\psi_M \wedge \psi_N$ *is satisfiable.*

If $M \sim N$ holds, then the shared refinement of M and N is the interface $M \sqcap N = \langle \hat{X}^I, \hat{X}^O, \hat{\varphi}, \hat{\psi}\rangle$, where

- $\hat{X}^I = X_M^I \cup X_N^I$,
- $\hat{X}^O = X_M^O \cup X_N^O$,
- $\hat{\varphi} \equiv (\varphi_M \vee \varphi_N)$, *and*
- $\hat{\psi} \equiv (\psi_M \wedge \psi_N)$.

Note that the assumption of the shared refinement of two interfaces is the disjunction of their assumptions, while in the parallel composition the assumptions are conjoined.

Theorem 5. *Let M and N be two well-formed stateless interfaces. If $M \sim N$, then $N \sqcap M$ is a well-formed stateless interface.*

Proof. Let $M = \langle X_M^I, X_M^O, \varphi_M, \psi_M\rangle$ and $N = \langle X_N^I, X_N^O, \varphi_N, \psi_N\rangle$. Due to well-formedness of M and N, we have that φ_M and φ_N are satisfiable, and therefore $\hat{\varphi} \equiv \varphi_M \vee \varphi_N$ is satisfiable. It follows from $M \sim N$ that $\hat{\psi} \equiv (\psi_M \wedge \psi_N)$ is satisfiable. $\square$

Example 1. *Assume that after decomposing a design, we obtain (among others) two components with interface descriptions M and N, respectively. Both interfaces refer to input variable x and output variable y. The interfaces are depicted in Figure 2. The interface M states that the component has to accept all even numbers, and produces numbers that are multiple of 3. The second interface N takes numbers greater than 0 and guarantees multiples of 4. A common implementation has to accept all even numbers and all numbers greater than 0. Furthermore, it has to guarantee that every output is a multiple of 3 and 4, therefore a multiple of 12.*

In the following, we prove that the shared refinement of two interfaces subsume all behaviors of the given interfaces.

Theorem 6 (greatest lower bound). *Let M and N be two well-formed stateless interfaces. If $M \sim N$, then $M \sqcap N \preceq M$ and $M \sqcap N \preceq N$, and for all well-formed stateless interfaces S, if $S \preceq M$ and $S \preceq N$, then $S \preceq M \sqcap N$.*

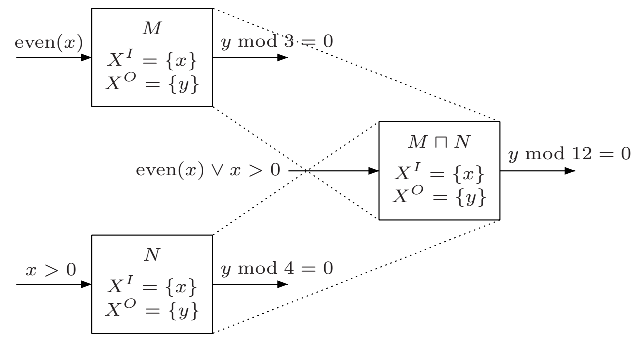

Figure 2: Shared refinement of two simple stateless interfaces.

Proof. Let $M \sqcap N = \langle \hat{X}^I, \hat{X}^O, \hat{\varphi}, \hat{\psi}\rangle$, let $M = \langle X_M^I, X_M^O, \varphi_M, \psi_M\rangle$, and let $N = \langle X_N^I, X_N^O, \varphi_N, \psi_N\rangle$.

First, we show that $M \sqcap N \preceq M$. The proof of $M \sqcap N \preceq N$ is analogous. From Condition (1) of Definition 7, since $M \sim N$, we have that $(\hat{X}^I \cup X_N^I) \cap (\hat{X}^O \cup X_N^O) = \emptyset$. Furthermore, φ_M implies $\hat{\varphi} \equiv \varphi_M \vee \varphi_N$ and $\hat{\psi} \equiv (\psi_M \wedge \psi_N)$ implies ψ_M. Hence $M \sqcap N \preceq M$.

Second, we show that every well-formed interface $S = \langle X_S^I, X_S^O, \varphi_S, \psi_S\rangle$ that refines both M and N is a refinement of $M \sqcap N$. Since $S \preceq M$ and $S \preceq N$, we have that $(X_S^I \cup X_M^I) \cap (X_S^O \cup X_M^O) = \emptyset$ and $(X_S^I \cup X_N^I) \cap (X_S^O \cup X_N^O) = \emptyset$. From Condition 1 of Definition 7, since $M \sim N$, we know that $(X_M^I \cup X_N^I) \cap (X_M^O \cup X_N^O) = \emptyset$ holds, so $(\hat{X}^I \cup X_S^I) \cap (\hat{X}^O \cup X_S^O) = \emptyset$ holds. It follows from $\varphi_M \to \varphi_S$ and $\varphi_N \to \varphi_S$ that $\hat{\varphi} \to \varphi_S$ (because $\hat{\varphi} \equiv \varphi_M \vee \varphi_N$). Furthermore, $\psi_S \to \psi_M$ and $\psi_S \to \psi_N$ implies $\psi_S \to \hat{\psi}$ (because $\hat{\psi} \equiv (\psi_M \wedge \psi_N)$). $\square$

3. STATEFUL INTERFACES

We consider interfaces with an internal *state*, analogous to the Moore interfaces of [2]. In each state of the interface, an assumption predicate constrains the input variables, and a predicate over output variables provides an output guarantee. The state of the interface changes according to a deterministic transition function over input and output variables.

Definition 8 (stateful interface). *A* stateful interface *is a tuple $M = \langle X^I, X^O, Q, \hat{q}, \varphi, \psi, \rho\rangle$, where*

- X^I, X^O *are disjoint sets of input and output variables. We define $X_M = X^I \cup X^O$;*
- Q *is a finite set of locations (or states), and $\hat{q} \in Q$ is the initial location;*
- φ *and* ψ *are two labelings that associate with each location $q \in Q$ an* input assumption *predicate $\varphi(q)$ over X^I, and an* output guarantee *predicate $\psi(q)$ over X^O;*
- ρ *is a labeling that associates with each pair of locations $q, q' \in Q$ a predicate $\rho(q, q')$ over X_M, called the* transition guard.

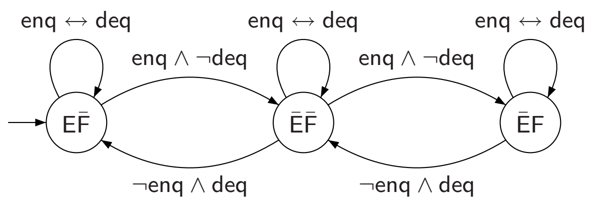

Figure 3: FIFO buffer – functional specification S_1.

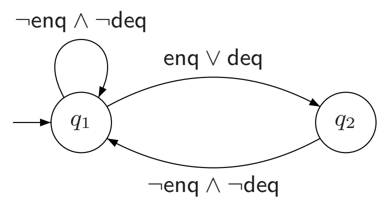

Figure 4: FIFO buffer – functional specification S_2.

Example 2. *We illustrate the stateful model of interfaces with the specification of a FIFO buffer. The buffer has two Boolean input variables* enq *and* deq*, which are set to perform an enqueue or dequeue operation, and two Boolean output variables* E *and* F*, which describe whether the buffer is empty or full. In Figure 3, we present a specification S_1 for this buffer, which we assume to be of size* 2*. States are labeled by their guarantee, and transitions are labeled by their guards. The assumption in a state is the disjunction of the guards of the outgoing transitions. Initially, the buffer is empty* ($\mathsf{E}\bar{\mathsf{F}}$) *and therefore it is not allowed to dequeue. When the buffer is neither empty nor full* ($\bar{\mathsf{E}}\bar{\mathsf{F}}$)*, dequeuing makes it empty, and enqueuing makes it full. If the buffer is full* ($\bar{\mathsf{E}}\mathsf{F}$)*, then enqueuing is not allowed. Simultaneous enqueue and dequeue operations have no effect (but they are allowed).*

Given a set X of variables, we denote by $\mathcal{V}[X]$ the set of all *valuations* v for X, i.e., the functions that assign to each $x \in X$ a value $v(x)$. Given a predicate φ on X, we write $v \models \varphi$ if the valuation v satisfies φ.

An *execution* of M is a sequence $q_0, v_0, q_1, \ldots, q_n, v_n, q_{n+1}$ of states $q_k \in Q$ and valuations $v_k \in \mathcal{V}[X_M]$ such that $q_0 = \hat{q}$, and $v_k \models \varphi_M(q_k) \wedge \psi_M(q_k) \wedge \rho_M(q_k, q_{k+1})$ for all $0 \le k \le n$. We say that the sequence $v_0, \ldots, v_n$ is a *trace* of M, and that the states $q_0, \ldots, q_{n+1}$ are *reachable* in M. We denote by $\mathsf{Traces}(M)$ the set of all traces of M, and by $\mathsf{Reach}(M)$ the set of all states that are reachable in M.

Definition 9 (well-formedness). *A stateful interface $M = \langle X^I, X^O, Q, \hat{q}, \varphi, \psi, \rho\rangle$ is* well-formed *if for all states $q \in \mathsf{Reach}(M)$,*

(1) *both $\varphi(q)$ and $\psi(q)$ are satisfiable,*

(2) *$(\varphi(q) \wedge \psi(q)) \rightarrow \exists q' \cdot \rho(q, q')$ is valid, and*

(3) *$(\rho(q, q') \wedge \rho(q, q'')) \rightarrow (q' = q'')$ is valid for all $q', q'' \in Q$.*

Well-formedness ensures that the interface is non-blocking by condition (1) and (2), and deterministic by condition (3).

Refinement and parallel composition.

The definition of refinement and parallel composition for stateful interfaces follows the lines of [2].

Definition 10 (Refinement). *Given two stateful interfaces $M = \langle X^I_M, X^O_M, Q_M, \hat{q}_M, \varphi_M, \psi_M, \rho_M\rangle$ and $N = \langle X^I_N, X^O_N, Q_N, \hat{q}_N, \varphi_N, \psi_N, \rho_N\rangle$, we say that N* refines *M, written $N \preceq M$, if*

(1) *$(X^I_N \cup X^I_M) \cap (X^O_N \cup X^O_M) = \emptyset$ and*

(2) *there exists a relation $R \subseteq Q_N \times Q_M$ such that $(\hat{q}_N, \hat{q}_M) \in R$, and for all pairs $(r, q) \in R$,*

(2a) *$\varphi_M(q) \rightarrow \varphi_N(r)$ is valid,*

(2b) *$\psi_N(r) \rightarrow \psi_M(q)$ is valid, and*

(2c) *for all $q' \in Q_M$ and $r' \in Q_N$, if $\varphi_M(q)$ and $\psi_N(r)$ and $\rho_M(q, q')$ and $\rho_N(r, r')$ are valid, then $(r', q') \in R$.*

Such a relation R is an alternating simulation relation [1] and we say that R is a *witness* for $N \preceq M$.

For parallel composition, we do require that the two stateful interfaces share no output variable, but we allow variables that are both output variable in one interface and input variable in the other interface. This implicitly enables a *connection* operation for stateful interfaces.

Definition 11 (parallel composition). *Given two stateful interfaces $M = \langle X^I_M, X^O_M, Q_M, \hat{q}_M, \varphi_M, \psi_M, \rho_M\rangle$ and $N = \langle X^I_N, X^O_N, Q_N, \hat{q}_N, \varphi_N, \psi_N, \rho_N\rangle$, let*

- *$X^O_P = X^O_M \cup X^O_N$,*
- *$X^I_P = (X^I_M \cup X^I_N) \setminus X^O_P$,*
- *$Q_P = Q_M \times Q_N$, and*
- *$\hat{q}_P = (\hat{q}_M, \hat{q}_N)$,*

and for all states $q, q' \in Q_M$ and $r, r' \in Q_N$, let

- *$\psi_P(q, r) \equiv (\psi_M(q) \wedge \psi_N(r))$ and*
- *$\rho_P((q, r), (q', r')) \equiv (\rho_M(q, q') \wedge \rho_N(r, r'))$.*

We say that M and N are compatible for parallel composition*, written $M \approx N$, if*

(1) *$X^O_M \cap X^O_N = \emptyset$ and*

(2) *there exists a labeling $\varphi_\otimes$ that associates with each pair $(q, r) \in Q_M \times Q_N$ a predicate $\varphi_\otimes(q, r)$ on X^I_P such that*

(2a) *$\varphi_\otimes(q, r)$ is satisfiable, and*

(2b) *for all executions $(q_0, r_0), v_0, (q_1, r_1), \ldots, (q_n, r_n)$ of $\langle X^I_P, X^O_P, Q_P, \hat{q}_P, \varphi_\otimes, \psi_P, \rho_P\rangle$, we have that $v_k \models (\varphi_M(q_k) \wedge \varphi_N(r_k))$ holds for all $0 \le k < n$.*

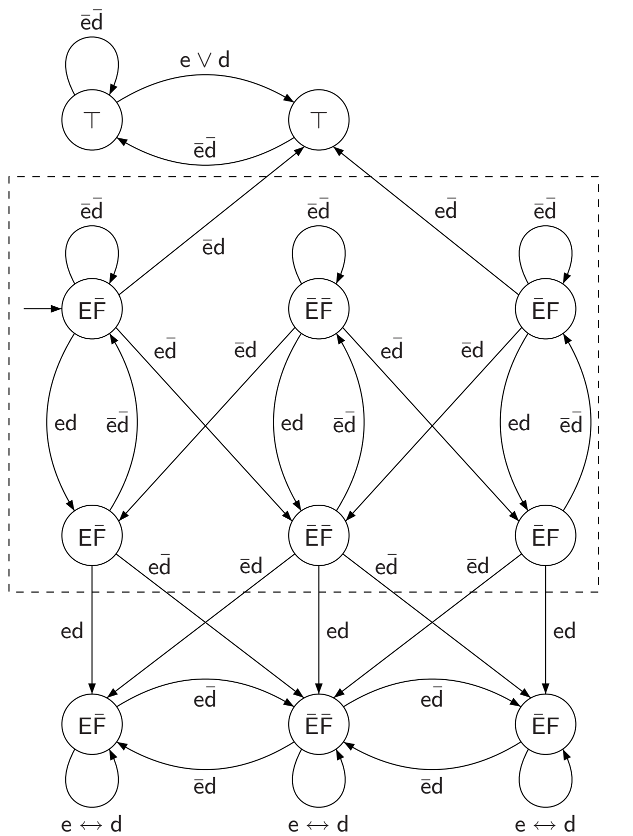

Figure 5: FIFO buffer – shared refinement $S_1 \sqcap S_2$.

When $M \approx N$, the parallel composition $M \| N$ is the well-formed stateful interface $P = \langle X_P^I, X_P^O, Q_P, \hat{q}_P, \varphi_P, \psi_P, \rho_P \rangle$, where φ_P is the weakest labeling[1] *such that conditions* $(2a)$ *and* $(2b)$ *are satisfied.*

Algorithms are presented in [2] to check refinement (given interfaces M and N, decide whether $N \preceq M$) and to check compatibility for parallel composition (given interfaces M and N, decide whether $M \approx N$, and if the answer is affirmative, then construct $M \| N$). Refinement checking is done by constructing the largest alternating simulation relation for $N \preceq M$ using an iterative fixed point computation. Starting with the relation $R_0 = Q_N \times Q_M$, for each $i \geq 0$, the relations R_{i+1} are obtained by removing the pairs (r, q) from R_i that violate one of the conditions $(2a)$, $(2b)$ or $(2c)$ in Definition 10 (for R_i instead of R). Compatibility for parallel composition is checked by solving a safety game. The input assumptions of the interface P in Definition 11 are iteratively strengthened to ensure that no state with unsatisfiable assumption is reachable. Initially, for all pairs of states $(q, r) \in Q_M \times Q_N$, the assumption $\varphi_P(q, r)$ is set to $\forall X_P^O \cdot (\psi_P(q, r) \to (\varphi_M(q) \wedge \varphi_N(r)))$. The algorithm stops when no assumption needs to be strengthened, and the interfaces are declared compatible for parallel composition if the pair of their initial states has a satisfiable assumption. The updated iterface P computed by the algorithm is their parallel composition.

Parallel composition supports independent implementability. In a design $M \| S$, we can replace M by any N that refines M if all variables common to N and S are also variables of M, because then $N \| S$ refines $M \| S$.

Theorem 7. [2] *Let $M, N,$ and S be three well-formed stateful interfaces such that $X_N \cap X_S \subseteq X_M$. If $M \approx S$ and $N \preceq M$, then $N \approx S$ and $N \| S \preceq M \| S$.*

[1]Here, a labeling φ is *weaker* than a labeling φ' if $\varphi'(p) \to \varphi(p)$ is valid for all states p that are reachable in P with labeling φ'. Note that if φ is weaker than φ', then every state that is reachable in P with φ' is also reachable in P with φ. Furthermore, the predicates that label the states that are not reachable in P with φ' are irrelevant. Therefore, assuming that $\top$ is the assumption of every unreachable state of an interface, it is easy to show that there always exists a weakest labeling.

Shared refinement.

The shared refinement $M \sqcap N$ is the weakest interface that refines both M and N. Roughly, the interface for $M \sqcap N$ starts with a classical product construction of M and N, where the assumptions allow inputs that satisfy the assumption of either M or N. As long as both assumptions of M and N are satisfied by the inputs, the outputs must satisfy the conjunction of the guarantees of M and N. When the assumption of M (resp. N) is violated, then the interface jumps to a copy of N (resp. M), where the assumptions and guarantees are those of N (resp. M). Note that the interface does not allow inputs that violate the assumptions of both M and N.

Definition 12 (shared refinement). *Given two stateful interfaces $M = \langle X_M^I, X_M^O, Q_M, \hat{q}_M, \varphi_M, \psi_M, \rho_M \rangle$ and $N = \langle X_N^I, X_N^O, Q_N, \hat{q}_N, \varphi_N, \psi_N, \rho_N \rangle$, let P be the interface $\langle X_P^I, X_P^O, Q_P, \hat{q}_P, \varphi_P, \psi_P, \rho_P \rangle$ where*

- $X_P^I = X_M^I \cup X_N^I$,
- $X_P^O = X_M^O \cup X_N^O$,
- $Q_P = (Q_M \times Q_N) \cup Q_M \cup Q_N$,
- $\hat{q}_P = (\hat{q}_M, \hat{q}_N)$,
- φ_P *and* ψ_P *are defined, for all $q \in Q_M$ and $r \in Q_N$, by*

$$
\begin{array}{ll}
\varphi_P(q, r) \equiv (\varphi_M(q) \vee \varphi_N(r)) & \psi_P(q, r) \equiv (\psi_M(q) \wedge \psi_N(r)) \\
\varphi_P(q) \quad \equiv \varphi_M(q) & \psi_P(q) \quad \equiv \psi_M(q) \\
\varphi_P(r) \quad \equiv \varphi_N(r) & \psi_P(r) \quad \equiv \psi_N(r),
\end{array}
$$

- ρ_P *is defined, for all $q, q' \in Q_M$ and $r, r' \in Q_N$, by*

$$
\begin{array}{ll}
\rho_P((q, r), (q', r')) & \equiv (\varphi_M(q) \wedge \varphi_N(r) \wedge \rho_M(q, q') \wedge \rho_N(r, r')) \\
\rho_P((q, r), q') & \equiv (\varphi_M(q) \wedge \neg\varphi_N(r) \wedge \rho_M(q, q')) \\
\rho_P((q, r), r') & \equiv (\neg\varphi_M(q) \wedge \varphi_N(r) \wedge \rho_N(r, r')) \\
\rho_P(q, q') & \equiv \rho_M(q, q') \\
\rho_P(r, r') & \equiv \rho_N(r, r') \\
\rho_P(q, (q', r')) & \equiv \rho_P(r, (q', r')) \equiv \bot.
\end{array}
$$

We say that M and N are shared refinable, *written $M \sim N$, if*

- $X_P^I \cap X_P^O = \emptyset$ *and*
- $\psi_P(p)$ *is satisfiable for all states $p \in \mathsf{Reach}(P)$.*

When $M \sim N$, the shared refinement $M \sqcap N$ *is the well-formed stateful interface P.*

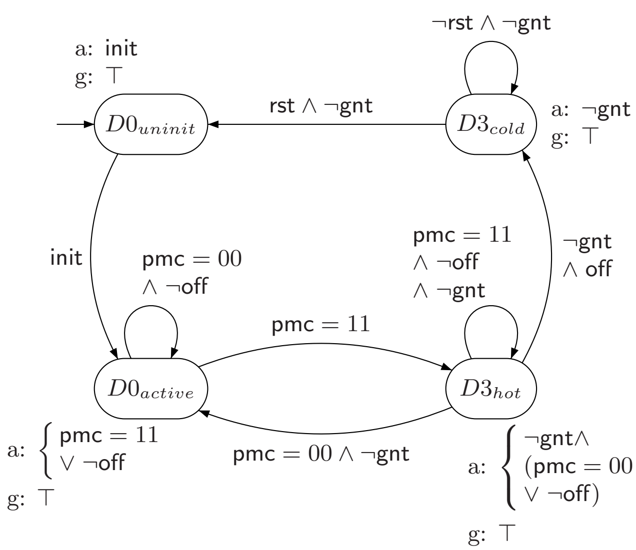

Figure 6: Power management view P.

EXAMPLE 3. *Figure 4 shows a different specification S_2 for the FIFO buffer. The interface requires that no two consecutive operations of either enqueuing or dequeuing occur, because in state q_2 neither* enq *nor* deq *is allowed. The guarantee is $\top$ in each state. Figure 5 shows the shared refinement $S_1 \sqcap S_2$ of the two specifications of the FIFO buffer, where* enq *and* deq *are abbreviated by* e *and* d*, and the conjunction operator $\wedge$ is omitted. The dashed box corresponds to the usual synchronized product of automata. To obtain the shared refinement, additional transitions are leaving this box when the assumption of one of the specifications is violated. From then on, only the assumption and the guarantee of the other specification need to hold.*

The shared refinement has the flavor of a greatest lower bound for the refinement relation $\preceq$, because $M \sqcap N$ refines both M and N, and every interface refining both M and N also refines $M \sqcap N$. Since the relation $\preceq$ is not necessarily a partial order (it is reflexive and transitive, but not necessarily antisymmetric), the notion of greatest lower bound is not well-defined as it may not be unique. However, the partial order defined in the usual way over the equivalence classes of the relation $\preceq \cap \preceq^{-1}$ has shared refinement as greatest lower bound operator.

THEOREM 8 (GREATEST LOWER BOUND). *Let M and N be two well-formed stateful interfaces. If $M \sim N$, then $M \sqcap N \preceq M$ and $M \sqcap N \preceq N$, and for all well-formed stateful interfaces S, if $S \preceq M$ and $S \preceq N$ then $S \preceq M \sqcap N$.*

PROOF. Let M and N be stateful interfaces such that $M \sim N$ and let $M \sqcap N = \langle X_P^I, X_P^O, Q_P, \hat{q}_P, \varphi_P, \psi_P, \rho_P \rangle$. First, we show that $M \sqcap N \preceq M$. We have $X_P^I \cap X_P^O = \emptyset$ and thus $(X_P^I \cup X_M^I) \cap (X_P^O \cup X_M^O) = \emptyset$. Let $R \subseteq Q_P \times Q_M$ be the union of $R_1 = \{((q,r),q) \mid (q,r) \in Q_P\}$ and $R_2 = \{(q,q) \mid q \in Q_M\}$. We have $(\hat{q}_P, \hat{q}_M) \in R$ and for all $((q,r),q) \in R_1$,

(1) $\varphi_M(q) \to \varphi_P(q,r)$ since $\varphi_P(q,r) \equiv \varphi_M(q) \vee \varphi_N(r)$,

(2) $\psi_P(q,r) \to \psi_M(q)$ since $\psi_P(q,r) \equiv \psi_M(q) \wedge \psi_N(r)$,

(3) if $\varphi_M(q) \wedge \psi_P(q,r) \wedge \rho_P((q,r),p') \wedge \rho_M(q,q')$, then either $p' = q'$ and then $(p',q') \in R_2$, or $p' = (q',r')$ for some $r' \in Q_N$ and then $(p',q') \in R_1$.

Analogous results trivially hold for R_2, and therefore R is a witness for $M \sqcap N \preceq M$. We have $M \sqcap N \preceq N$ by a symmetrical argument.

Second, we show that $S \preceq M$ and $S \preceq N$ implies $S \preceq M \sqcap N$ for all S. Since $X_P^I \cap X_P^O = \emptyset$, it is easy to show that $(X_S^I \cup X_P^I) \cap (X_S^O \cup X_P^O) = \emptyset$. Let R_1 be a witness for $S \preceq M$, and R_2 be a witness for $S \preceq N$. Let $R \subseteq Q_S \times Q_P$ be the union of $R_1 \cup R_2$ and $R_3 = \{(s,(q,r)) \mid (s,q) \in R_1 \text{ and } (s,r) \in R_2\}$. We have $(\hat{q}_S, \hat{q}_P) \in R$ because $(\hat{q}_S, \hat{q}_M) \in R_1$ and $(\hat{q}_S, \hat{q}_N) \in R_2$. Moreover, for all $(s,(q,r)) \in R_3$, we have

(1) $\varphi_P(q,r) \to \varphi_S(s)$ since $\varphi_P(q,r) \equiv \varphi_M(q) \vee \varphi_N(r)$, $\varphi_M(q) \to \varphi_S(s)$, and $\varphi_N(r) \to \varphi_S(s)$,

(2) $\psi_S(s) \to \psi_P(q,r)$ since $\psi_S(s) \to \psi_M(q)$, $\psi_S(s) \to \psi_N(r)$ and $\psi_P(q,r) \equiv \psi_M(q) \wedge \psi_N(r)$, and

(3) if $\varphi_P(q,r) \wedge \psi_S(s) \wedge \rho_S(s,s') \wedge \rho_P((q,r),p')$, then either

   (a) $p' = (q',r')$ and $\varphi_M(q) \wedge \varphi_N(r) \wedge \rho_M(q,q') \wedge \rho_N(r,r')$, and then $\varphi_M(q) \wedge \psi_S(s) \wedge \rho_S(s,s') \wedge \rho_M(q,q')$ so that $(s',q') \in R_1$, and symmetrically $(s',r') \in R_2$, and therefore $(s',p') \in R_3$, or

   (b) $p' = q' \in Q_M$ and $\varphi_M(q) \wedge \rho_M(q,p')$ and then $(s',p') \in R_1$, or

   (c) $p' = r' \in Q_N$ and $(s',p') \in R_1$ by a symmetric argument.

Analogous results trivially hold for all $(s,p) \in R_1 \cup R_2$, and therefore R is a witness for $S \preceq M \sqcap N$. □

Finally, we have the following associativity property of shared refinement, which follows from the greatest lower bound property of Theorem 8.

THEOREM 9. *Given three well-formed stateful interfaces M, N, and S, either both $(M \sqcap N) \sqcap S$ and $M \sqcap (N \sqcap S)$ are undefined, or they are both defined, and then they refine each other.*

4. REUSE OF PCI DEVICES

We illustrate the use of shared refinement for specifications of the peripheral interconnection of components on a bus. On the one hand, for the PCI bus, we consider the functional specification as described in [2] and the power management interface [9]. On the other hand, we consider the functional specification of a different peripheral bus. Finally, we show that the shared refinement of these three specifications is an interface of any implementation of a device that satisfies the power management specification, and is functionally compatible with both busses.

Power management interface.

A PCI function is a device that can be connected to the PCI bus, and which has to implement its own power management interface. According to the PCI Bus Power Management Interface Specification [9], each PCI function can be in one of four power-management states: $D0$, $D1$, $D2$, or $D3$ in decreasing order of power consumption. The states $D0$ and $D3$ are further split into $D0_{uninit}$, $D0_{active}$, and $D3_{hot}$, $D3_{cold}$. To comply with the PCI standard, all PCI functions have to support the $D0$ and $D3$ states.

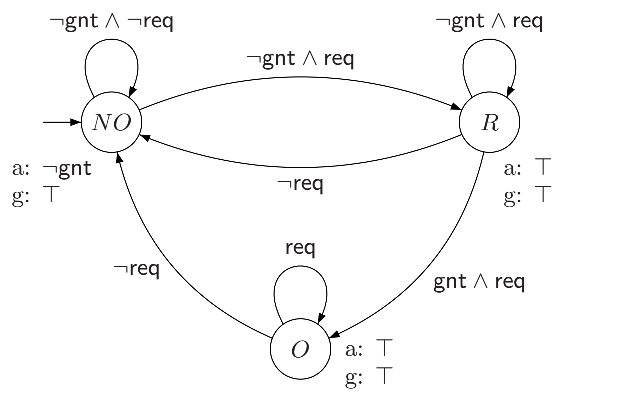

Figure 7: First functional view F_1.

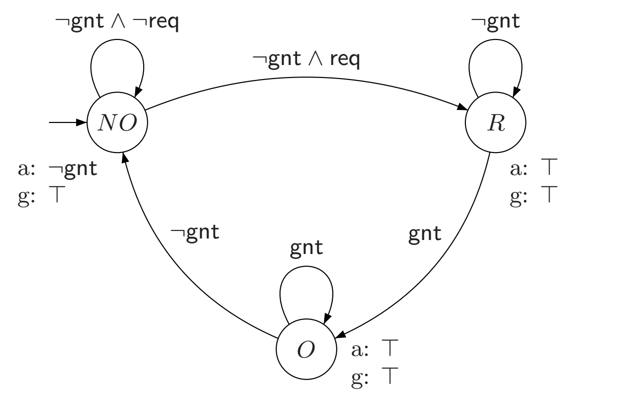

Figure 8: Second functional view F_2.

In Figure 6, we present a stateful interface P for controlling the different power states for the PCI bus. The input signals are $X^I = \{\mathsf{init}, \mathsf{rst}, \mathsf{pmc}, \mathsf{gnt}, \mathsf{off}\}$, and there is no output signal. In state $D0_{uninit}$ the PCI function must be initialized by the system software (variable init) in order to be put in the active state $D0_{active}$. Functions in $D3_{hot}$ can move to the $D0_{active}$ state and back via software by writing to the function's PMCSR register (variable pmc). The difference between the two $D3$ states is defined by the absence ($D3_{cold}$) or presence ($D3_{hot}$) of voltage V_{cc} (regulated by the off signal). A PCI function can be transitioned into $D3_{cold}$ states either by software (variable off) or by physically removing the power from its host PCI device. Functions in $D3_{cold}$ can only get to $D0_{uninit}$ by reapplying V_{cc} and asserting the reset signal (rst) to the function's host PCI device.

Additionally, the device may receive a signal gnt when it has been granted to access the PCI bus, but this should not happen when it is in one of the low-consumption states $D3$.

Bus request management.

We consider the functional specification described in [2] (see Figure 7) for connection with the PCI bus, and we define a different specification for a different peripheral bus (see Figure 8). F_1 and F_2 are stateful interfaces of the device that can be attached to the corresponding peripheral bus. Interfaces F_1 and F_2 have the three states $NotOwner$, $Request$, and $Owner$ (of the device). The input variable is gnt, and the output variable is req to request the bus.

As required by the PCI bus, the stateful interface F_1 specifies that the device either keeps requesting the bus until it is granted, or it withdraws the request and goes back to the $NotOwner$ state. Further, the stateful interface F_2 gives a specification of a different bus, where the device is expected to send a req signal once and then wait to be granted.

Note that the two specifications F_1 and F_2 do not refine each other: the trace starting with $\mathsf{req} \wedge \neg\mathsf{gnt}, \mathsf{req} \wedge \mathsf{gnt}, \mathsf{req} \wedge \neg\mathsf{gnt}, \neg\mathsf{req} \wedge \mathsf{gnt}, \ldots$ violates the assumption $\neg\mathsf{gnt}$ of the state $NotOwner$ in interface F_2, while it can be continued according to the specification F_1. Therefore the device should recognize that it has to continue the execution according to the specification F_1. Further, the trace starting with $\mathsf{req} \wedge \neg\mathsf{gnt}, \neg\mathsf{req} \wedge \mathsf{gnt}, \neg\mathsf{req} \wedge \mathsf{gnt}, \ldots$ is allowed by specification F_2 but not by F_1.

Now we consider the interfaces $C_1 = P \sqcap F_1$ and $C_2 = P \sqcap F_2$. Any component that refines C_1 satisfies the power management specification P, and at the same time, such a component is compatible with a PCI bus as described with the functional interface F_1. An analogous statement holds for C_2. However, in order to enable the reuse of components, one may require that the same component refines both C_1 and C_2, or equivalently refines $C_1 \sqcap C_2 = P \sqcap F_1 \sqcap F_2$. Moreover, any component which refines both interfaces C_1 and C_2 has to refine $S = P \sqcap F_1 \sqcap F_2$, as it is the weakest interface with this property. Note that C_1 and C_2 are shared refinable, because P, F_1, and F_2 have trivial guarantees. Therefore, we can construct the stateful interface S, which has $5 \cdot 4 \cdot 4 - 1 = 79$ states.

The device implementations that are compliant with both busses are exactly those that refine S. The interface S ensures that the guarantees of all three specifications P, F_1, and F_2 are satisfied as long as the assumptions of the three specifications are met. If the inputs no longer conform to the power specification P, then only the guarantees of the functional specifications F_1 and F_2 can be maintained. If the inputs no longer comply with the PCI specification F_1, then only the guarantees for F_2 and P can be ensured. A similar statement holds for F_2. Finally, if the inputs violate the assumptions of two of the three specifications, then S moves to a copy of the third interface and behaves according to that interface.

5. DISCUSSION

The shared refinement of two interfaces represents the most permissive interface that refines both interfaces. This can be viewed as a greatest lower bound property for the refinement relation (Theorem 8), which is defined as alternating simulation. Classically, the parallel composition also defines a greatest lower bound, but for the trace inclusion relation. For instance, in automata theory, the language $L(A\|B)$ of the parallel product of two automata is exactly the set of all traces that are common to the languages of A and B, i.e., $L(A\|B) = L(A) \cap L(B)$. Notice that $\cap$ is the greatest lower bound for set inclusion.

In the theory of interfaces, the set of traces of the parallel composition $M\|N$ may not be the intersection of the traces of M and N. More precisely, there exist two well-formed interfaces M over variables X_M and N over X_N

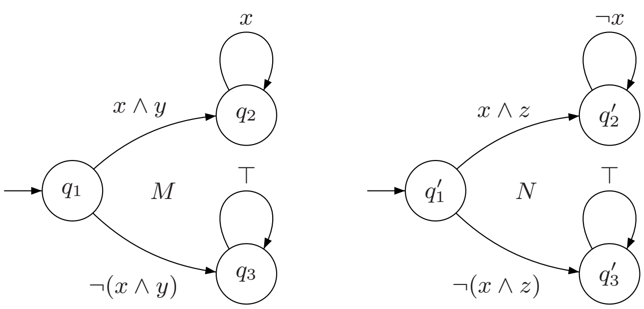

Figure 9: Two interfaces M and N.

that are compatible for parallel composition and such that $\mathsf{Traces}(M\|N) \subsetneq \mathsf{Traces}'(M) \cap \mathsf{Traces}'(N)$, where $\mathsf{Traces}'(M)$ (resp. $\mathsf{Traces}'(N)$) is the set of sequences $v_1, \ldots, v_n$ of valuations for $X_M \cup X_N$ that agree with a trace of M (resp. N) on variables in X_M (resp. X_N). Consider the interfaces M and N in Figure 9 over the Boolean variables x, y, and z, where $X_M^I = X_N^I = \{x\}$, $X_M^O = \{y\}$, and $X_N^O = \{z\}$. Assumptions and guarantees in all states are trivial except in q_2 and q_2', where the assumptions are $\varphi_M(q_2) \equiv x$ and $\varphi_N(q_2') \equiv \neg x$. In the parallel composition $M\|N$ (see Figure 10), the pair of states (q_2, q_2') should not be reachable, because their assumptions are incompatible. Hence, the assumption of (q_1, q_1') in $M\|N$ is strengthened to $\neg x$ in order to avoid a transition to (q_2, q_2'). So traces starting with valuation v such that $v(x) = \top$ and $v(y) = v(z) = \bot$ are not included in $\mathsf{Traces}(M\|N)$. On the other hand, M and N allow both all traces starting with valuations v_1 and v_2, respectively, such that $v_1(x) = v_2(x) = \top$ and $v_1(y) = v_2(z) = \bot$.

We note that interfaces strictly separate input and output variables, in the sense that assumptions refer to input variables only and guarantees to output variables only. This strict separation may force assumptions in the parallel composition to be stronger than intuitively necessary. In the previous example, one may expect the assumption of (q_1, q_1') to be $\top$ and its guarantee to be $\neg x \vee \neg y \vee \neg z$. This requires that guarantee predicates may refer to both input and output variables. Let us try to consider in the stateless case such an extended setting, which we call *extended interfaces*.

Note that assumptions would not be necessary anymore, as one can define the assumption φ as $\exists X^O \cdot \psi$, i.e., the allowed values of the input variables are those for which the guarantee predicate is satisfiable. So every pair of an assumption and a guarantee can be written as a single (maybe different) guarantee that would describe the same interface. The well-formedness condition (analogous of Definition 2) then would simply require that φ be satisfiable (which implies that ψ is satisfiable). However, in the following we keep assumptions and guarantees separately, because it is the natural way to think about interfaces.

If we consider extended interfaces, the definition of shared refinement could be adapted to keep the greatest lower bound property. Indeed, with the extended guarantees, we could define more permissive interfaces that refine two given interfaces. Specifically, using the notation of Definition 12, the assumption $\hat{\varphi}$ and the guarantee $\hat{\psi}$ of the shared refinement $M \sqcap N$ could be defined as

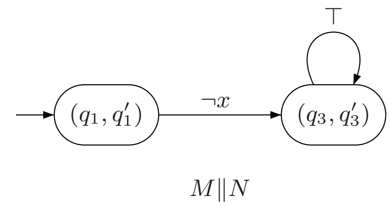

Figure 10: The parallel composition $M\|N$ for the interfaces of Figure 9.

- $\hat{\varphi} \equiv (\varphi_M \vee \varphi_N) \wedge \exists \hat{X}^O \cdot \hat{\psi}$, and
- $\hat{\psi} \equiv (\varphi_M \rightarrow \psi_M) \wedge (\varphi_N \rightarrow \psi_N)$.

The refined guarantee allows the new shared refinement to refine both M and N while the shared refinement of Definition 12 refines $M \sqcap N$. Theorem 6 holds for extended interfaces with this new definition.

EXAMPLE 4. *Consider the interfaces M and N of Example 1, which are now considered to be extented interfaces. The new guarantee of $M \sqcap N$ would be $(even(x) \rightarrow y \ mod\ 3 = 0) \wedge (x > 0 \rightarrow y \ mod\ 4 = 0)$, which is strictly weaker than the guarantee $y \ mod\ 12 = 0$ in the original setting. For instance, y would not required to be a multiple of 12 when x is a positive odd number.*

Extended interfaces seem to provide a stronger framework than classical interface theories. Unfortunately, the basic properties of stepwise refinement and independent implementability do not hold in the extended framework. Formally, the extended interfaces do not support congruence with connection, i.e., there exist a connection θ and two well-formed extended interfaces M and N such that $N \preceq M$ and M is compatible with θ, but N is not compatible with θ. Consider the stateless interfaces M and N over input variable x and output variable y with trivial assumptions and guarantees, except the guarantee of N, which is $\psi_N \equiv (y \neq x)$. Let θ be the connection $\{(y, x)\}$. The interface M is compatible with θ, and the interface $M\theta$ has the trivial assumption and guarantee $\hat{\psi}_M \equiv (y = x)$. However, even though N refines M, it is not compatible with θ because the predicate $\rho_\theta \equiv (y = x)$ contradicts the guarantee ψ_N and thus $N\theta$ would not be well-formed owing to an unsatisfiable guarantee. Hence, the analogous of Theorem 2 for extended interfaces does not hold.

We believe that the well-formedness requirement that the guarantee must be satisfiable should not be dropped in an interface theory. Well-formed interfaces should have at least one environment in which they can be embedded. In fact, by definition an interface is well-formed iff it can be used in some context [5]. On the other hand, independent implementability formalized by congruence is a crucial aspect of the theory and should not be dropped either. It turns out that interface theories in which inputs and outputs are separated are the most general known framework in which these two features coexist.

Acknowledgments

We gratefully acknowledge discussions with Albert Benveniste, who convinced us that the operation $\sqcap$ is missing from interface theories and inspired us to start this work. As part of the same debt, we also acknowledge Albert's coauthors of the talk on "Multiple viewpoints contracts and residuation," which he gave at the 2008 Workshop on Foundations of Interface Technologies (FIT): Jean-Baptiste Raclet, Eric Badouel, Benoit Caillaud, and Roberto Passerone.

6. REFERENCES

[1] R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. In *Proceedings of CONCUR: Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 1998.

[2] A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and F. Y. C. Mang. Synchronous and bidirectional component interfaces. In *Proceedings of CAV: Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 414–427. Springer, 2002.

[3] A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Resource interfaces. In *Proceedings of EMSOFT: Embedded Software*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2003.

[4] L. de Alfaro and T. A. Henzinger. Interface automata. In *Proceedings of FSE: Foundations of Software Engineering*, pages 109–120. ACM Press, 2001.

[5] L. de Alfaro and T. A. Henzinger. Interface theories for component-based design. In *Proceedings of EMSOFT: Embedded Software*, volume 2211 of *Lecture Notes in Computer Science*, pages 148–165. Springer, 2001.

[6] L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Timed interfaces. In *Proceedings of EMSOFT: Embedded Software*, volume 2491 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2002.

[7] T. A. Henzinger and S. Matic. An interface algebra for real-time components. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 253–266. IEEE Computer Society, 2006.

[8] E. A. Lee and Y. Xiong. System-level types for component-based design. In *Proceedings of EMSOFT: Embedded Software*, pages 237–253. Springer, 2001.

[9] PCI bus power management interface specification revision, 2004. http://www.pcisig.com/specifications/conventional.

[10] B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[11] E. Wandeler and L. Thiele. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In *Proceedings of EMSOFT: Embedded Software*, pages 80–89. ACM Press, 2005.

[12] E. Wandeler and L. Thiele. Interface-based design of real-time systems with hierarchical scheduling. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 243–252. IEEE Computer Society, 2006.