Reward Interfaces with Best-Effort Implementations

Rafael Dewes
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Rayna Dimitrova
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract

Interface theories, notably interface automata, serve as expressive frameworks for component-based design, specifying component behavior and interaction in concurrent systems. Traditional interface formalisms specify assumptions that a component's environment must satisfy and the guarantees that each component provides. This qualitative view of component interaction based on imposing strict assumptions and Boolean guarantees may, however, not be expressive enough to capture the system's allowed or desired behaviors under different environments.

In this paper, we introduce reward interfaces to support component-based design while accommodating multi-valued correctness requirements and adaptive best-effort satisfaction of component's guarantees. Building upon interface automata, our framework enables modeling a rich class of quantitative component specifications. We propose formal notions of implementation, refinement and compatibility for reward interfaces. We study a class of reward interfaces with automata-based representations, for which we provide algorithms for checking compatibility and refinement, and existence of best-effort implementations. Our framework offers a comprehensive approach to reward interface specification and design.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Component-based design, interface automata, quantitative specifications

Digital Object Identifier 10.4230/LIPIcs.CSL.2026.30

1 Introduction

In system design and analysis, component-based techniques are essential for managing the increasing size and complexity of modern systems. Specification theories, particularly those based on interfaces or contracts [3, 9, 14], provide structured frameworks to address the challenges posed by concurrent systems. Interface theories [15, 18, 10] are especially suited for expressing interactions and dependencies of subsystems, offering formal specifications for analysis and verification. Here, each component is modeled by an interface, enabling a modular, independent design process, while ensuring compatibility between components.

One well-studied formalism are automata-based stateful interfaces, building on de Alfaro and Henzinger's seminal work on *interface automata* [14]. They model components with distinct input and output actions, synchronizing with other components. Key properties are compatibility, which allows abstraction of multiple components into a single subsystem, and refinement, which supports independent component implementation. Interface automata have been extended to express more complex system behaviors. For example, resource interfaces [6] handle quantitative aspects like resource usage, while modal interfaces [20, 23, 21] capture richer properties such as liveness, expanding the expressivity of the framework. Further extensions [25, 24] incorporate state-based contracts, which enables concepts like shared memory in interface theories. A core motivation for interface theories is to simplify the design process by relieving designers from defining responses for every possible input. Typically, interfaces adopt an optimistic stance, allowing components to assume specific behavior from their environment. Inputs are either allowed or disallowed, with no obligation to handle

disallowed inputs. This assumption holds as long as the system is closed, meaning all inputs come from other components. However, for open systems, where external inputs are beyond our control, these strong assumptions must be reconsidered.

We introduce a novel approach of *reward interfaces*, building on interface automata, that aims to enhance the design process while accommodating an unrestricted external environment. Central to our approach is the concept of *good-enough* satisfaction, based on the idea of good-enough synthesis [2], which says satisfaction of requirements is only necessary under feasible input conditions. This notion enables designers to focus on essential aspects of system behavior, removing the task of considering more explicit environment restrictions. It also naturally lends itself to the quantitative domain, introducing an additional direction in expressing complex properties to interface specifications, such as graceful degradation.

We consider an automaton structure coupled with a multi-valued reward function, defined over sequences of inputs and outputs of the interface, and require best-effort satisfaction from its implementation. While this generally loosens obligations, it selects for high-quality implementations if they exist. The interface automaton provides strict assumptions and guarantees, but is limited to safety properties. The reward function is more flexible, and able to express a substantial class of properties including liveness. This keeps the automaton simple and interpretable, offloading complicated requirements to the function without obscuring essential interaction restrictions. Our approach allows designers to succinctly capture complex system behaviors, which we illustrate below using a simple example.

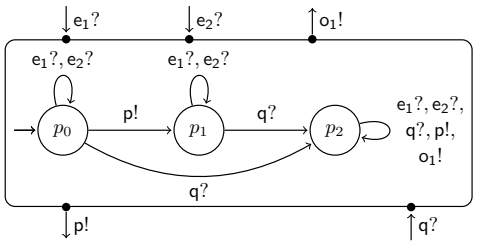
▶ **Example 1.** Suppose we want to design a message distribution system S. The system S can receive messages $\mathsf{e}_1, \mathsf{e}_2$ and should relay the received messages in the respective order via the output actions o_1 and o_2. The system is limited in that it cannot produce an output while receiving an input, and it cannot send more than one output at a time. If the system has to handle both e_1 and e_2 during an execution, this will consume more resources.

Formally, S is modeled via the input actions $\mathsf{e}_1, \mathsf{e}_2$, controlled by the external environment, and output actions o_1, o_2 and o_3, with the following requirements: Once e_1 occurs, then S must output o_1, and once e_2 occurs, S must output o_2. If both inputs are received, then o_1 and o_2 must be produced in the same order. S must not perform actions o_1, o_2 before e_1 or e_2, respectively, has happened. The additional output o_3 is unconstrained by the specification.

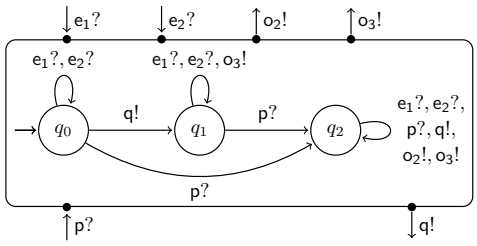
The specification of S is formalized as a *reward function* $\mathcal{F}_\mathsf{S}$ which maps the possible execution traces of S to numerical values reflecting the extent to which the requirements are satisfied. Concretely, $\mathcal{F}_\mathsf{S}$ maps (possibly infinite) sequences σ over the alphabet $\Sigma_\mathsf{S} = \{\mathsf{e}_1, \mathsf{e}_2, \mathsf{o}_1, \mathsf{o}_2, \mathsf{o}_3\}$ to values $v \in \{0, \frac{1}{4}, \frac{1}{2}, 1\}$.

Executions where o_1 occurs after an e_1, and no input e_2 is received, are awarded the maximum value of 1. The same value 1 is assigned to executions where the only input is e_2, and o_2 occurs after. If both e_1 and e_2 occur, and both o_1 and o_2 occur in the respective order, the achieved value will be $\frac{1}{2}$. This represents the more demanding input behavior from the environment, abstracting a lowered efficiency, and is not a penalty on the system. The value is lowered to $\frac{1}{4}$ if the order is reversed. Executions where an input is erroneously or not at all relayed are assigned value 0, violating the specification. We also give value 0 to sequences without any input, or an infinite sequence of inputs, as the system would be unable to produce output if continuously receiving inputs from the environment.

Since the inputs $\mathsf{e}_1, \mathsf{e}_2$ are under the control of the environment, the system cannot force them to occur. This means that no implementation of S can guarantee positive satisfaction of $\mathcal{F}_\mathsf{S}$. A more realistic requirement is to ask S to achieve the maximal satisfaction value possible for each sequence of input actions produced by the environment. In this *best-effort* view, the obligations on S depend on the input provided by the environment. Thus, if the



(a) Interface automaton A_P with additional output action p and additional input action q.



(b) Interface automaton A_Q with additional output action q and additional input action p.

Figure 1 Interface automata describing the components of the system in Example 1.

environment does not provide S with an input, S is not expected to achieve value higher than 0. Formally, we identify the so called $(\mathcal{F}_\mathsf{S}, v)$-*hopeful input sequences*, which allow for achieving satisfaction value v. Here, for $v = 1$ these are the sequences of the form $\mathsf{e_1}^+$ or $\mathsf{e_2}^+$.

Suppose that internally, S is to be designed as the composition of two components P and Q. Component P is responsible for producing the output $\mathsf{o_1}$, and component Q for outputs $\mathsf{o_2}, \mathsf{o_3}$. This is expressed via the *interface automata* A_P and A_Q modeling the allowed interactions of P and Q, as depicted in Figure 1. A_P and A_Q synchronize on actions p, q, and operate independently otherwise. They can enable each other to produce outputs $\mathsf{o_1}, \mathsf{o_2}$.

However, interface automata alone are unable to capture the quantitative requirement expressed via the function $\mathcal{F}_\mathsf{S}$. The modeling framework of *reward interfaces* that we propose in this paper addresses this limitation. Reward interfaces equip interface automata with *reward functions* to capture the quantitative specifications that components must satisfy. In the coming sections, we will see how reward interfaces for P and Q model local quantitative specifications such that their combination captures the high-level specification $\mathcal{F}_\mathsf{S}$. ⌟

As Example 1 shows, our framework offers an elegant way to describe systems of interacting components. Our reward interfaces build directly on interface automata, adding expressivity through the reward function without necessarily making the automata more complex.

We introduce the formal definition of *reward interfaces* in Section 3, and define reward functions in a general way to maintain a high degree of flexibility. The notions of compatibility and refinement on reward interfaces are defined in Section 4 and Section 5 respectively, and we show that these entail desirable properties of interfaces. In Section 6 we provide a collection of algorithmic solutions for assessing these properties for a specific class of interfaces. Concretely, we consider reward functions with finite range that are represented as automata. For brevity, some of the missing proofs are presented in full in the appendix.

2 Preliminaries

In this section, we introduce necessary notation and preliminaries. Further, we recall and adapt the definition of interface automata and related notions from [14].

Languages and Automata

For an alphabet Σ, the set $\Sigma^\infty := \Sigma^* \cup \Sigma^\omega$ contains all finite and infinite words over Σ. Given a sequence $\sigma \in (\Sigma \cup \Sigma')^\infty$, we denote with $\sigma|_{\Sigma'}$ the projection of σ to Σ', that is, $\sigma|_{\Sigma'} \in (\Sigma')^\infty$ is the sequence obtained from σ by removing all letters in $\Sigma \setminus \Sigma'$. For a sequence $\sigma \in \Sigma^\infty$, we denote with $\sigma[i] \in \Sigma$ the letter of σ at the i-th position.

A non-deterministic finite automaton (NFA) over an alphabet Σ is a tuple $\mathcal{N} = \langle Q, \Sigma, \delta, Q_0, F\rangle$ with finite set of states Q, initial states $Q_0 \subseteq Q$, transition relation $\delta \subseteq Q \times \Sigma \times Q$, and a set of accepting states $F \subseteq Q$. A run of $\mathcal{N}$ on a finite word $\sigma = a_1 \dots a_n \in \Sigma^*$ is a finite sequence $\rho = \rho_0 \rho_1 \dots \rho_n \in Q^*$ such that $\rho_0 \in Q_0$ and for every $i < n$, it holds that $\rho_{i+1} \in \delta(\rho_i, \sigma_{i+1})$. A run ρ is accepting if and only if $\rho_n \in F$. A non-deterministic Büchi automaton (NBA) $\mathcal{B} = \langle Q, \Sigma, \delta, Q_0, B\rangle$ on infinite words instead has a set of Büchi-accepting states $B \subseteq Q$ of which some must be visited infinitely often. A run ρ of $\mathcal{B}$ on $\sigma \in \Sigma^\omega$ is defined analogously, and is accepting if and only if for every $i \in \mathbb{N}$ there exists $j \geq i$ such that $\rho[j] \in B$. The *language of an automaton* $\mathcal{L}(\mathcal{A})$ is the set of words σ on which $\mathcal{A}$ has an accepting run. For a universal co-Büchi automaton (UCW) $\mathcal{C} = \langle Q, \Sigma, \delta, Q_0, B\rangle$, a run ρ of $\mathcal{C}$ on a word $\sigma \in \Sigma^\omega$ is accepting if and only if there exists $i \in \mathbb{N}$ such that for all $j \geq i$, $\rho[j] \notin B$, and $\sigma \in \Sigma^\omega$ is only accepted by $\mathcal{C}$ if every run is accepting. We define the size of an automaton $\mathcal{A}$ as the total number of states and transitions, i.e. $|\mathcal{A}| = |Q| + |\delta|$.

We use $\mathbb{R}$ to denote the set of real numbers, and define $\infty, -\infty$ as the elements where $\infty > n > -\infty$ for all $n \in \mathbb{R}$. The set $\mathbb{R}_{-\infty} = \mathbb{R} \cup \{-\infty\}$ then contains the real numbers and $-\infty$. We define conventionally the infimum over the empty set as $\inf(\emptyset) = \infty$.

Interface Automata

Interface automata [14] are a modeling formalism for specifying the interactions between components and their environment. Components communicate via synchronization on input and output actions. In contrast to the interface automata in [14], we distinguish *external inputs* Σ^E, which are external to the overall system and broadcast to all components, from inputs Σ^I that come from system components outside of A and are subject to assumptions specified by A. The external inputs enable synchronization between multiple components on the same input action, and are used to model behavior uncontrollable by the system.

▶ **Definition 2** (Interface Automaton (adapted from [14])). *An interface automaton* $A = \langle V, V^{init}, \Sigma^I, \Sigma^O, \Sigma^E, \Sigma^H, \mathcal{T}\rangle$ *is a tuple where:*

- *V is a finite set of states and $V^{init} \subseteq V$ is a set of initial states. We require that V^{init} contains at most one state. If $V^{init} = \emptyset$, then A is called* empty*.*
- *$\Sigma^I, \Sigma^O, \Sigma^E, \Sigma^H$ are mutually disjoint finite sets of* input, output, external input *and* internal *actions respectively. Let $\Sigma_A := \Sigma^I \cup \Sigma^O \cup \Sigma^E \cup \Sigma^H$ be the set of all actions of A.*
- *$\mathcal{T} \subseteq V \times \Sigma_A \times V$ is a set of transitions such that for every state $v \in V$ it holds that:*
  - *for every external input action $a \in \Sigma^E$, there exists $v' \in V$ such that $(v, a, v') \in \mathcal{T}$;*
  - *for all $a \in \Sigma^E \cup \Sigma^I$ and $v', v'' \in V$ with $(v, a, v') \in \mathcal{T}$ and $(v, a, v'') \in \mathcal{T}$ we have $v' = v''$.*

Note that we require A to be *input enabled on external input actions* Σ^E, i.e., it accommodates any external input in every state. We also require that A is *input-deterministic* on $\Sigma^E \cup \Sigma^I$, in order to ensure compositionality of parallel composition [15].

We define the size of A as $|A| = |V| + |\mathcal{T}|$. For $u \in V$, we define the interface automaton $A^u = \langle V, \{u\}, \Sigma^I, \Sigma^O, \Sigma^E, \Sigma^H, \mathcal{T}\rangle$ obtained from A by replacing the set of initial states by $\{u\}$.

Let $A = \langle V, V^{init}, \Sigma^I, \Sigma^O, \Sigma^E, \Sigma^H, \mathcal{T}\rangle$ be an interface automaton. An *execution* of A is an alternating sequence $v_0, a_0, v_1, a_1, \dots$ of states $v_i \in V$ and actions $a_i \in \Sigma_A$ such that $(v_i, a_i, v_{i+1}) \in \mathcal{T}$ for all $i \geq 0$. An *execution fragment* is a finite prefix $v_0, a_0, v_1, a_1, \dots, v_n$ of an execution ending in a state. A state $v \in V$ is *reachable in A* if there exists an execution fragment starting in some $v_0 \in V^{init}$ that ends in v. We denote with $\mathsf{Reach}(A) \subseteq V$ the set of states reachable in A. For $\Sigma' \subseteq \Sigma_A$, let $\mathsf{Enabled}_A(v, \Sigma') := \{a \in \Sigma' \mid \exists v'.\ (v, a, v') \in \mathcal{T}\}$ be the set of actions from Σ' *enabled in state v*.

Two key notions in the theory of interface automata are the *composability* and *product* of automata. Whether two interface automata are composable depends on their actions. They must not share input, output and internal actions, but may have joint external inputs. For two composable interface automata, we can construct their product. This is formalized in the definitions below. For the remainder of this section, let $A_P = \langle V_P, V_P^{init}, \Sigma_P^I, \Sigma_P^O, \Sigma_P^E, \Sigma_P^H, \mathcal{T}_P \rangle$ and $A_Q = \langle V_Q, V_Q^{init}, \Sigma_Q^I, \Sigma_Q^O, \Sigma_Q^E, \Sigma_Q^H, \mathcal{T}_Q \rangle$ be interface automata.

▶ **Definition 3** (Composability [14]). *Two interface automata A_P and A_Q are* composable *if $\Sigma_P^I \cap \Sigma_Q^I = \emptyset$, $\Sigma_P^O \cap \Sigma_Q^O = \emptyset$, $\Sigma_P^H \cap \Sigma_Q = \emptyset$, and $\Sigma_Q^H \cap \Sigma_P = \emptyset$.*

We define $\mathsf{Shared}(A_P, A_Q) := (\Sigma_P^I \cap \Sigma_Q^O) \cup (\Sigma_Q^I \cap \Sigma_Q^O)$*, and denote with* $\mathsf{JointE}(A_P, A_Q) := \Sigma_P^E \cap \Sigma_Q^E$ *the set of joint external inputs of A_P and A_Q.*

▶ **Definition 4** (Product). *Let A_P and A_Q be composable. Their* product $A_P \otimes A_Q = \langle V_{P\otimes Q}, V_{P\otimes Q}^{init}, \Sigma_{P\otimes Q}^I, \Sigma_{P\otimes Q}^O, \Sigma_{P\otimes Q}^E, \Sigma_{P\otimes Q}^H, \mathcal{T}_{P\otimes Q} \rangle$ *is the interface automaton with the components defined as follows:*

- $V_{P\otimes Q} := V_P \times V_Q$, $V_{P\otimes Q}^{init} = V_P^{init} \times V_Q^{init}$,
- $\Sigma_{P\otimes Q}^I = (\Sigma_P^I \cup \Sigma_Q^I) \setminus \mathsf{Shared}(A_P, A_Q)$, $\Sigma_{P\otimes Q}^O = (\Sigma_P^O \cup \Sigma_Q^O) \setminus \mathsf{Shared}(A_P, A_Q)$, $\Sigma_{P\otimes Q}^E = \Sigma_P^E \cup \Sigma_Q^E$, $\Sigma_{P\otimes Q}^H = (\Sigma_P^H \cup \Sigma_Q^H) \cup \mathsf{Shared}(A_P, A_Q)$,
- $$\begin{aligned} \mathcal{T}_{P\otimes Q} = \{((v_P, v_Q), \alpha, (v_P', v_Q')) \mid{}& \alpha \in (\mathsf{Shared}(A_P, A_Q) \cup \mathsf{JointE}(A_P, A_Q)) \\ & \wedge (v_P, \alpha, v_P') \in \mathcal{T}_P \wedge (v_Q, \alpha, v_Q') \in \mathcal{T}_Q\} \\ \cup \{((v_P, v_Q), \alpha, (v_P', v_Q)) \mid{}& \alpha \notin (\mathsf{Shared}(A_P, A_Q) \cup \mathsf{JointE}(A_P, A_Q)) \\ & \wedge (v_P, \alpha, v_P') \in \mathcal{T}_P\} \\ \cup \{((v_P, v_Q), \alpha, (v_P, v_Q')) \mid{}& \alpha \notin (\mathsf{Shared}(A_P, A_Q) \cup \mathsf{JointE}(A_P, A_Q)) \\ & \wedge (v_Q, \alpha, v_Q') \in \mathcal{T}_Q\}. \end{aligned}$$

Note that the actions $\mathsf{Shared}(A_P, A_Q)$ become internal actions of $A_P \otimes A_Q$, and P and Q must synchronize on shared and joint external input actions.

Given two composable interface automata A_P and A_Q, a product state $(v_P, v_Q) \in V_P \times V_Q$ is called an *illegal state* of the product automaton $A_{P\otimes Q}$ if and only if there exists $a \in \mathsf{Shared}(A_P, A_Q)$ such that $a \in (\mathsf{Enabled}_P(v_P, \Sigma_P^O) \setminus \mathsf{Enabled}_Q(v_Q, \Sigma_Q^I)) \cup (\mathsf{Enabled}_Q(v_Q, \Sigma_Q^O) \setminus \mathsf{Enabled}_P(v_P, \Sigma_P^I))$. That is, illegal states are ones in which a shared action can be produced as an output of one of the interface automata but is not allowed as an input by the other one. Let $\mathsf{Illegal}(A_P, A_Q)$ be the set of illegal states of $A_P \otimes A_Q$.

We say that two interface automata A_P and A_Q are *compatible* if there exists a way for the rest of the system, supplying the inputs $\Sigma_{P\otimes Q}^I$ to ensure that illegal states are avoided. This gives rise to the notion of *legal environment* for A_P and A_Q, which we recall next.

▶ **Definition 5** (Legal Environment and Compatibility[14]). *A* legal environment *for (A_P, A_Q) is a non-empty interface automaton $A_R = \langle V_R, V_R^{init}, \Sigma_R^I, \Sigma_R^O, \Sigma_R^E, \Sigma_R^H, \mathcal{T}_R \rangle$ where:*

1. $\Sigma_R^E = \Sigma_{P\otimes Q}^E$, $\Sigma_R^I = \Sigma_{P\otimes Q}^O$, $\Sigma_R^O = \Sigma_{P\otimes Q}^I$, $\Sigma_R^H = \emptyset$.
2. *A_R is composable with $A_P \otimes A_Q$, and* $\mathsf{Illegal}(A_P \otimes A_Q, A_R) = \emptyset$.
3. $\mathsf{Reach}((A_P \otimes A_Q) \otimes A_R) \cap (\mathsf{Illegal}(A_P, A_Q) \times V_R) = \emptyset$.

Two interface automata A_P and A_Q are considered compatible *if they are non-empty, composable and there exists a legal environment for (A_P, A_Q).*

Intuitively, a legal environment A_R for (A_P, A_Q) represents the remaining system beyond A_P and A_Q. Condition 3 requires that the inputs provided by A_R to the two interfaces steer them away from illegal states in the product.

▶ **Definition 6** (Composition of Interface Automata[14]). *Given composable interface automata A_P and A_Q, a product state $(v_P, v_Q) \in V_P \times V_Q$ is called* compatible *if there exists a legal environment for $(A_P^{v_P}, A_Q^{v_Q})$. Let $\mathsf{Comp}(A_P, A_Q)$ be the set of compatible product states. The* composition *$A_P \parallel A_Q$ of A_P and A_Q is defined by restricting $A_P \otimes A_Q$ to $\mathsf{Comp}(A_P, A_Q)$. Formally, $A_P \parallel A_Q = \langle V_{P\otimes Q} \cap \mathsf{Comp}(A_P, A_Q), V^{init}_{P\otimes Q} \cap \mathsf{Comp}(A_P, A_Q), \Sigma^I_{P\otimes Q}, \Sigma^O_{P\otimes Q}, \Sigma^E_{P\otimes Q}, \Sigma^H_{P\otimes Q}, \mathcal{T}_{P\otimes Q} \cap (\mathsf{Comp}(A_P, A_Q) \times \Sigma_{P\otimes Q} \times \mathsf{Comp}(A_P, A_Q))\rangle$.*

Intuitively, the compatible product states are those from which an environment can prevent reaching illegal states. Thus, A_P and A_Q are compatible if and only if $V^{init}_{P\otimes Q} \subseteq \mathsf{Comp}(A_P, A_Q)$. Furthermore, for every $(v_P, v_Q) \in \mathsf{Comp}(A_P, A_Q)$, all the external input actions $\Sigma^E_{P\otimes Q}$ lead to product sates in $\mathsf{Comp}(A_P, A_Q)$. If this was not the case, (v_P, v_Q) would not be in $\mathsf{Comp}(A_P, A_Q)$, since an external input action leading to an illegal state cannot be prevented by a legal environment. If an action $a \in \Sigma^O_{P\otimes Q}$ leads from (v_P, v_Q) to $\mathsf{Illegal}(A_P, A_Q)$, we prune all a-transitions from (v_P, v_Q). Therefore, for compatible interface automata, the composition $A_P \parallel A_Q$ is a non-empty interface automaton conforming to Definition 2.

Another concept, crucial for independent implementability of interfaces, is *refinement*. Refinement of interface automata [14] is defined via *alternating simulation*, recalled below.

For an interface automaton A and state v, let $\varepsilon\text{-}\mathsf{closure}_A(v)$ be the set of states of A that can be reached from v using only internal actions from Σ^H. We define:

$$\begin{array}{lcl} \mathsf{ExtEn}^{I,E}_A(v) & := & \{a \mid \forall u \in \varepsilon\text{-}\mathsf{closure}_A(v).\ a \in \mathsf{Enabled}_A(u, \Sigma^I_A \cup \Sigma^E_A)\} \text{ and} \\ \mathsf{ExtEn}^{O}_A(v) & := & \{a \mid \exists u \in \varepsilon\text{-}\mathsf{closure}_A(v).\ a \in \mathsf{Enabled}_A(u, \Sigma^O_A)\}. \end{array}$$

to be the sets of externally enabled input and output actions at v. Further, for actions $a \in \mathsf{ExtEn}^{I,E}_A(v) \cup \mathsf{ExtEn}^{O}_A(v)$, let $\mathsf{ExtDest}_A(v, a) = \{u' \mid \exists (u, a, u') \in \mathcal{T}_A.\ u \in \varepsilon\text{-}\mathsf{closure}_A(v)\}$.

▶ **Definition 7** (Alternating Simulation [14]). *A binary relation $\succeq \subseteq V_P \times V_Q$ is an* alternating simulation from the interface automaton A_Q to the interface automaton A_P *if for all $v_P \in V_P$ and $v_Q \in V_Q$, $v_P \succeq v_Q$ implies that:*

1. $\mathsf{ExtEn}^{I,E}_P(v_P) \subseteq \mathsf{ExtEn}^{I,E}_Q(v_Q)$ *and* $\mathsf{ExtEn}^{O}_Q(v_Q) \subseteq \mathsf{ExtEn}^{O}_P(v_P)$.
2. *For all $a \in \mathsf{ExtEn}^{I,E}_P(v_P) \cup \mathsf{ExtEn}^{O}_Q(v_Q)$ and $v'_Q \in \mathsf{ExtDest}_Q(v_Q, a)$, there exists a state $v'_P \in \mathsf{ExtDest}_P(v_P, a)$ such that $v'_P \succeq v'_Q$.*

The existence of an alternating simulation from A_Q to A_P guarantees that the interactions of A_P are preserved in A_Q. In particular, A_Q does not impose more assumptions on the environment than A_P, and satisfies all the output restrictions of A_P. That is, any environment compatible with A_P also is compatible with A_Q.

▶ **Definition 8** (Interface Automata Refinement [14]). *For interface automata A_P and A_Q, we say that A_Q* refines *A_P, written $A_Q \preceq A_P$, if and only if $\Sigma^E_Q = \Sigma^E_P$, $\Sigma^I_Q \supseteq \Sigma^I_P$, $\Sigma^O_Q \subseteq \Sigma^O_P$, and there exists an alternating simulation $\succeq$ from A_Q to A_P, and states $v_P \in V^{init}_P$ and $v_Q \in V^{init}_Q$, such that $v_P \succeq v_Q$.*

Thus, the relation $A_Q \preceq A_P$ guarantees that A_Q must accept at least all inputs $\Sigma^E_P \cup \Sigma^I_P$, and may not add any new outputs w.r.t. Σ^O_P. The alternating simulation ensures that for any input, the outward behavior of Q matches that of P.

3 Reward Interfaces

We now introduce *reward interfaces*, the central notion of the framework we propose. They build on the classical interface automata, extending them with an additional quantitative requirement, defined as a *reward function* that assigns numerical values to sequences of observable actions of the interface. Essentially, a reward function defines a *quantitative language* [8] over the alphabet of non-internal actions of the given interface automaton.

▶ **Definition 9** (Reward Interface). *A* reward interface *is a pair* $P = (A_P, \mathcal{F}_P)$ *where* $A_P = \langle V_P, V_P^{init}, \Sigma_P^I, \Sigma_P^O, \Sigma_P^E, \Sigma_P^H, \mathcal{T}_P \rangle$ *is an interface automaton and* $\mathcal{F}_P : (\Sigma_P^{Obs})^\infty \to \mathbb{R}_{-\infty}$ *is a partial function that assigns a value from* $\mathbb{R}_{-\infty}$ *to sequences over the alphabet* $\Sigma_P^{Obs} := \Sigma_P^I \cup \Sigma_P^O \cup \Sigma_P^E$ *of non-internal actions of* A_P.

Intuitively, $\mathcal{F}_P$ expresses a quantitative specification, by associating reward values with sequences in $(\Sigma_P^{Obs})^\infty$. The reward value of a sequence describes how well the respective observed behavior satisfies the quantitative requirement. Since $\mathcal{F}_P$ is part of the interface specification, it is defined in terms of the actions Σ_P^{Obs} that are visible to the component's environment. We deliberately do not impose further restrictions on the functions $\mathcal{F}_P$, in order to retain full generality of the proposed framework. In Section 6 we discuss possible instantiations, in which the reward functions have a natural finite representation.

▶ **Example 10.** Continuing from Example 1, we show a possible reward interface $P = (A_P, \mathcal{F}_P)$ for component P. The restrictions from A_P (Figure 1a) ensure that output $\mathsf{o_1}$ can only occur after input q. The requirement that $\mathsf{o_1}$ must occur after $\mathsf{e_1}$, and not before, is modeled via the reward function $\mathcal{F}_P$, mapping sequences over $\Sigma_P^{Obs} = \{\mathsf{e_1}, \mathsf{e_2}, \mathsf{p}, \mathsf{q}, \mathsf{o_1}\}$ to values. The reward function $\mathcal{F}_P$ follows $\mathcal{F}_\mathsf{S}$: We award value 1 if only one type of input $\mathsf{e_1}$ or $\mathsf{e_2}$ occurs, and require respectively $\mathsf{o_1}$ and p, instead of $\mathsf{o_2}$, as P has no knowledge nor control over $\mathsf{o_2}$. Similarly, if both $\mathsf{e_1}$ and $\mathsf{e_2}$ occur, the order of $\mathsf{o_1}$ and p must reflect that to achieve value $\frac{1}{2}$, otherwise the value is lowered to $\frac{1}{4}$. In both cases when $\mathsf{e_2}$ occurs, $\mathsf{o_1}$ and p should not be performed infinitely, otherwise the assigned value is 0.

Note that while we could for example encode the safety requirement that there is no $\mathsf{o_1}$ before the first occurrence of $\mathsf{e_1}$ as part of the interface automaton, it would make the automaton structure more complicated. Furthermore, as the internal order of $\mathsf{o_1}$ and $\mathsf{e_1}$ is less relevant for the interaction with component Q, which is not concerned with $\mathsf{o_1}$, it is more meaningful to capture that in the reward function $\mathcal{F}_P$ rather than A_P.

A reward function $\mathcal{F}_Q$ for a reward interface $Q = (A_Q, \mathcal{F}_Q)$ is defined analogously. ⌟

For $v \in \mathbb{R}_{-\infty}$ and $\sim \in \{<, \leq, \geq, >\}$, we define $\mathcal{F}_P^{\sim v} := \{\sigma \in (\Sigma_P^{Obs})^\infty \mid \mathcal{F}_P(\sigma) \sim v\}$ to be the set of words which $\mathcal{F}_P$ maps to some value $\sim v$. When we write $\mathcal{F}_P(\sigma) \sim v$, we implicitly mean that $\mathcal{F}_P(\sigma)$ is defined. We define $\mathit{Vals}(\mathcal{F}_P) := \{v \in \mathbb{R}_{-\infty} \mid \exists \sigma \in (\Sigma_P^{Obs})^\infty.\ \mathcal{F}(\sigma) = v\}$.

Let us fix an interface automaton $A = \langle S, S^{init}, \Sigma^I, \Sigma^O, \Sigma^E, \Sigma^H, \mathcal{T} \rangle$ for the rest of this section. To evaluate implementations of an interface, we define the set of *traces of* A as

$$\begin{aligned} \mathsf{Traces}(A) \quad &:= \quad \{\sigma \in \Sigma_A^\omega \mid \exists \text{ execution of } A \text{ on } \sigma\} \\ &\cup \quad \{\sigma \in \Sigma_A^* \mid \exists \text{ exec. of } A \text{ on } \sigma \text{ ending in } s : \mathsf{Enabled}(s, \Sigma^O \cup \Sigma^H) = \emptyset\}. \end{aligned}$$

That is, $\mathsf{Traces}(A)$ is the set that consists of the infinite words over Σ_A for which there exists an infinite execution, as well as the finite words over Σ_A where an execution ends in a state with no output or internal action possible. That is, we consider maximal traces, taking into account that the environment inputs are not forced to occur.

For a given input sequence $\sigma_{E,I} \in (\Sigma^E \cup \Sigma^I)^\infty$, we define $\mathsf{Traces}(A, \sigma_{E,I}) := \{\sigma \in \mathsf{Traces}(A) \mid \sigma|_{(\Sigma^E \cup \Sigma^I)} = \sigma_{E,I}\}$ to be the subset of $\mathsf{Traces}(A)$ containing the words consistent with $\sigma_{E,I}$. These words represent all possible behaviors of A when the environment provides the inputs specified by $\sigma_{E,I}$. If $\Gamma \subseteq \Sigma_A$ is an alphabet such that $\Gamma \supseteq \Sigma^I \cup \Sigma^E$, we define $\mathsf{Traces}_\Gamma(A, \sigma_{E,I}) := \{\gamma \in \Gamma^\infty \mid \exists \sigma \in \mathsf{Traces}(A, \sigma_{E,I}) \wedge \sigma|_\Gamma = \gamma\}$ to be the projection of $\mathsf{Traces}(A, \sigma_{E,I})$ on Γ.

Reward functions impose no assumptions on the environment of a component. However, the value that a component can possibly achieve may depend on the behaviour of the environment. Therefore, we require that components implementing a reward interface satisfy the quantitative specification to the best extent possible with respect to the input provided by the component's environment. This intuition is formalized by the *good-enough* criterion for interface automata (also used to model implementations).

First, we adapt to our setting the notion of *hopeful* sequences [2], which is used to characterize the input sequences for which a given reward value is possible. Note that in our setting, we consider asynchronous executions, such that there can be several consecutive inputs with no output in-between, or such that from some point on we only have outputs. This is in contrast to hopeful inputs in [2], which are defined for synchronous executions. We also allow arbitrary hidden actions, which may interleave with the non-internal actions.

▶ **Definition 11** (Hopeful Sequences). *Given alphabets Σ and Γ such that $\Gamma \subseteq \Sigma$, a function $\mathcal{F} : \Sigma^\infty \to \mathbb{R}_{-\infty}$ and $v \in \mathbb{R}_{-\infty}$, we say that a sequence $\gamma \in \Gamma^\infty$ is* $(\mathcal{F}, v)$-hopeful *if and only if there exists $\sigma \in \Sigma^\infty$ such that $\gamma = \sigma|_\Gamma$, $\mathcal{F}(\sigma)$ is defined, and $\mathcal{F}(\sigma) \geq v$. We denote the set of $(\mathcal{F}, v)$-hopeful Γ-sequences by $\mathit{Hopeful}(\mathcal{F}, v, \Gamma) := \{\gamma \in \Gamma^\infty \mid \exists \sigma \in \Sigma^\infty.\ \gamma = \sigma|_\Gamma \wedge \mathcal{F}(\sigma) \geq v\}$.*

We use Γ here to denote a subset of Σ, which will typically be instantiated to be a subset of the inputs $\Sigma^I \cup \Sigma^E$, as we will see in the following definition. This notion describes the potential quality of the environment, characterizing the inputs to an interface as hopeful with respect only to specific values of $\mathcal{F}$. The hopefulness of an input sequence does not depend on the interface automaton itself, but only on the reward function $\mathcal{F}$. An interface automaton is *good-enough* with respect to a reward function $\mathcal{F}$, if, intuitively, the traces it generates on any $(\mathcal{F}, v)$-hopeful input sequence achieve reward at least v.

▶ **Definition 12** (Good-Enough Interface Automaton). *Consider an interface automaton A and a reward function $\mathcal{F} : \Delta^\infty \to \mathbb{R}_{-\infty}$. We say that A is* good-enough *with respect to $\mathcal{F}$ if and only if for every value $v \in \mathit{Vals}(\mathcal{F})$, every input sequence $\sigma_{E,I} \in \mathit{Hopeful}(\mathcal{F}, v, (\Sigma_A^E \cup \Sigma_A^I) \cap \Delta)$, and every $\sigma \in \mathsf{Traces}_{(\Delta \cap \Sigma_A)}(A, \sigma_{E,I})$ it holds that $\mathcal{F}(\sigma)$ is defined and $\mathcal{F}(\sigma) \geq v$.*

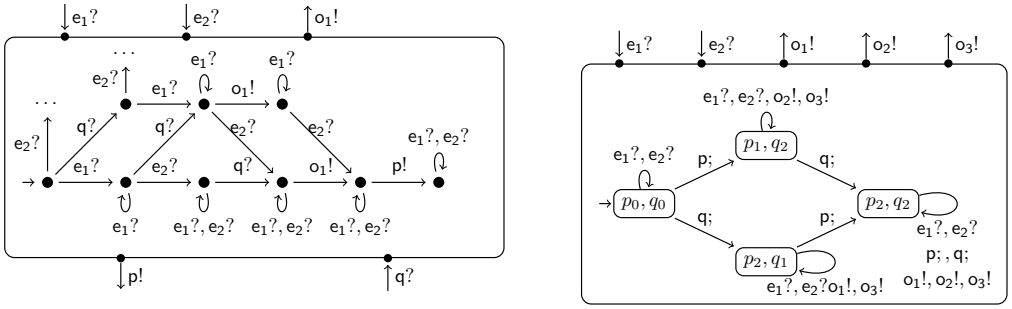
This definition is in the spirit of [2], as a good-enough interface automaton must perform to the best extent possible according to $\mathcal{F}$, but only for the input it receives. In Definition 12 we used a general alphabet Δ. This is useful for the definition of compatibility of reward interfaces with respect to a given reward function.

Now we define the notion of *(best-effort) implementation* of a reward interface, which must be good-enough with respect the interface's reward function.

▶ **Definition 13** (Best-Effort Implementation). *An interface automaton S is an* implementation *of a reward interface $P = (A_P, \mathcal{F}_P)$ if and only if it satisfies the following conditions.*

1. *S refines the interface automaton A_P.*
2. *S is* good-enough *with respect to the function $\mathcal{F}_P$.*

We denote with $\mathsf{Imp}(P)$ the set of all implementations of P.



(a) (Part of) implementation S_P of P. **(b)** Composition $A_P \parallel A_Q$.

Figure 2 Interface automata for an implementation of P and composition of A_P and A_Q.

An implementation of a reward interface P is an instance of a classical interface automaton, as it does not have a reward function. In the original theory, there is no distinction between an interface and its implementation, as the interface is defined by the automaton structure alone. When considering an interface in isolation, the external inputs Σ^E and inputs Σ^I are treated in the same way. However, we differentiate between the two when considering the interface in the context of rest of the system, which generates the inputs Σ^I.

▶ **Example 14.** Let us examine how a best-effort implementation for P could act. With respect to the reward function $\mathcal{F}_P$ from Example 10, a possible implementation S_P for P waits for e_1 or e_2, and then expects input q to follow with o_1, or produces output p, respectively. We give a portion of the respective implementation in Figure 2a, for the behaviors where e_1 occurs first. The missing part has analogous structure for the case when e_2 is received first.

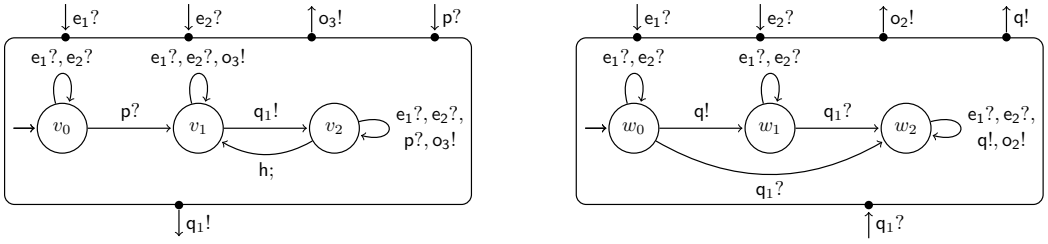
As the implementation must not perform o_1 unless e_1 was received, it waits until e_1 happens. Should q never appear, S_P has no obligation to perform o_1. This is because, even though $\mathcal{F}_P$ does not require q, Definition 12 considers the traces of S_P, where q must be read before o_1 can be produced. While P is unable to control or observe o_2, it exercises control by withholding p until e_2, which is represented in $\mathcal{F}_P$, and therefore in implementation S_P. If instead, the implementation could immediately produce p, the value for $\mathcal{F}_P$ would be lower since it could be that no e_2 occurred. ⌟

4 Compatibility and Composition of Reward Interfaces

In this section, we lift the notions of interface compatibility and composition to reward interfaces. Following the classical interface theory, compatibility of reward interfaces requires the existence of a legal environment for the underlying interface automata. For the quantitative part of our reward interfaces, we define *compatibility with respect to a "joint" reward function* $\mathcal{F}$, which, intuitively, is a quantitative specification with respect to which the composition of the implementations of the two interfaces must be good enough. In top-down component-based design, we need to ensure that the local reward functions for interfaces guarantee that the composition of their implementations is good-enough with respect to a given high-level reward function $\mathcal{F}$. This is precisely the $\mathcal{F}$-compatibility criterion we define.

For this section, let $P = (A_P, \mathcal{F}_P)$ and $Q = (A_Q, \mathcal{F}_Q)$ be composable reward interfaces with $A_P = \langle V_P, V_P^{init}, \Sigma_P^I, \Sigma_P^O, \Sigma_P^E, \Sigma_P^H, \mathcal{T}_P \rangle$ and $A_Q = \langle V_Q, V_Q^{init}, \Sigma_Q^I, \Sigma_Q^O, \Sigma_Q^E, \Sigma_Q^H, \mathcal{T}_Q \rangle$, and $A_P \otimes A_Q = \langle V_{P\otimes Q}, V_{P\otimes Q}^{init}, \Sigma_{P\otimes Q}^I, \Sigma_{P\otimes Q}^O, \Sigma_{P\otimes Q}^E, \Sigma_{P\otimes Q}^H, \mathcal{T}_{P\otimes Q} \rangle$ their product [Definition 4].

We now define $\mathcal{F}$-compatibility, for a given reward function $\mathcal{F} : (\Sigma_{P\otimes Q}^{Obs})^\infty \to \mathbb{R}_{-\infty}$ with alphabet consisting of the non-internal actions of the product automaton $A_P \otimes A_Q$.



(a) Interface automaton A_{Q_1}.

(b) Interface automaton A_{Q_2}.

Figure 3 Splitting of Q into two components Q_1 and Q_2 as described in Example 16.

▶ **Definition 15** ($\mathcal{F}$-Compatibility). *The reward interfaces $P = (A_P, \mathcal{F}_P)$ and $Q = (A_Q, \mathcal{F}_Q)$ are* $\mathcal{F}$-compatible *for a given function $\mathcal{F} : (\Sigma^{Obs}_{P\otimes Q})^\infty \to \mathbb{R}_{-\infty}$ if and only if A_P and A_Q are compatible and for all implementations $S_P \in \mathsf{Imp}(P)$ and $S_Q \in \mathsf{Imp}(Q)$ of P and Q respectively, if S_P and S_Q are composable, then $S_P \parallel S_Q$ is good-enough with respect to $\mathcal{F}$.*

▶ **Example 16.** We now illustrate that the reward interfaces P and Q are $\mathcal{F}_\mathsf{S}$-compatible for the reward function $\mathcal{F}_\mathsf{S}$ from Example 1. In particular, the composition of P and Q refines the high-level specification of S expressed by $\mathcal{F}_\mathsf{S}$. Clearly, P and Q are composable, and since $\mathsf{Illegal}(A_P, A_Q) = \emptyset$, we can construct their composition $A_P \parallel A_Q$ (shown in Figure 2b).

To see that P and Q are $\mathcal{F}_\mathsf{S}$-compatible, consider any pair of implementations S_P and S_Q. Now we explain why the composition $S_P \parallel S_Q$ must be good-enough with respect to $\mathcal{F}_\mathsf{S}$. We examine $(\mathcal{F}_\mathsf{S}, v)$-hopeful input sequences, and consider the expected behaviors of S_P and S_Q. $\mathcal{F}_P$ requires S_P to produce o_1 once e_1 was received, and not before. The same holds with respect to $\mathcal{F}_Q$, and o_2 and e_2. Since $\mathcal{F}_P$ and $\mathcal{F}_Q$ further constrain p and q, respectively, this ensures that both S_P and S_Q will each have the opportunity to output o_1 or o_2 respectively, once required. Thus, any pair of best-effort implementations of P and Q together will operate as a best-effort implementation of the high-level reward function $\mathcal{F}_\mathsf{S}$.

In top-down design, we may further refine our model, by splitting Q into two components Q_1 and Q_2. Suppose for example that Q_1 handles p, and can output o_3 and some q_1. Q_2 outputs o_2 and q, and synchronizes with Q_1 via q_1. The interface automata A_{Q_1} and A_{Q_2} are in Figure 3. The reward function $\mathcal{F}_{Q_1}$ requires that q_1 only occurs once e_2 did. Similarly, $\mathcal{F}_{Q_2}$ requires q after e_1. Through action q_1, Q_1 can prevent Q_2 from incorrectly producing o_2.

In the product $A_{Q_1} \otimes A_{Q_2}$, the state (v_1, w_2) is illegal, because A_{Q_1} can output q_1 from v_1, but w_2 in A_{Q_2} can not accept it. In the composition $A_{Q_1} \parallel A_{Q_2}$, state (v_1, w_2) is therefore removed. It is easy to see that the composition of any pair of implementations of Q_1 and Q_2 is good enough with respect to $\mathcal{F}_Q$. Thus, Q_1 and Q_2 are $\mathcal{F}_Q$-compatible. ⌟

The $\mathcal{F}$-compatibility of A_P and A_Q guarantees that $(A_P \parallel A_Q, \mathcal{F})$ is a reward interface, as it requires that the interface automata A_P and A_Q are compatible, and $\mathcal{F}$ has the right domain. Furthermore, the second condition of $\mathcal{F}$-compatibility ensures that P and Q can be implemented independently, and composing the resulting implementations yields a good-enough implementation of $(A_P \parallel A_Q, \mathcal{F})$. By checking $\mathcal{F}$-compatibility of two reward interfaces for a given reward function $\mathcal{F}$, we can establish whether their individual reward functions are aligned to guarantee the good-enough satisfaction of the specification $\mathcal{F}$.

In bottom-up design, on the other hand, we want to construct the composition of two interfaces by *combining their reward functions*, in order to express the guarantees of two interfaces when put together. For this, we need to provide a definition of *composition of reward functions*. The definition is parametrized by a function $comb : \mathbb{R}_{-\infty} \times \mathbb{R}_{-\infty} \to \mathbb{R}_{-\infty}$, which specifies how we combine the numeric values assigned by the two functions.

The observable behaviours of the composition $P \parallel Q$ are over the alphabet $\Sigma^{Obs}_{P\otimes Q} = (\Sigma^{Obs}_P \cup \Sigma^{Obs}_Q) \setminus \mathsf{Shared}(P,Q)$ of non-internal actions of $P \parallel Q$. Thus, the domain of the composed reward function must be the set of sequences over $\Sigma^{Obs}_{P\otimes Q}$. Our goal is to define the composition of $\mathcal{F}_P$ and $\mathcal{F}_Q$ in a way that captures the combined value achieved by any pair of implementations of the two interfaces. Since the implementations of each of the interfaces are constrained by the respective reward function, we cannot assume that any pair of implementations will be cooperating. Therefore, the value assigned to a sequence $\sigma \in (\Sigma^{Obs}_{P\otimes Q})^\infty$ corresponds to the worst case over all the traces produced by some pair of composable implementations that are compatible with the sequence $\sigma|_{\Sigma^{I,E}_{P\otimes Q}}$ of inputs in σ.

▶ **Definition 17** (Reward Function Composition). *Let $comb : \mathbb{R}_{-\infty} \times \mathbb{R}_{-\infty} \to \mathbb{R}_{-\infty}$. We define the* composition $\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q : (\Sigma^{Obs}_{P\otimes Q})^\infty \to \mathbb{R}_{-\infty}$ *of the functions $\mathcal{F}_P$ and $\mathcal{F}_Q$, such that for $\sigma \in (\Sigma^{Obs}_{P\otimes Q})^\infty$ where there exist composable $S'_P \in \mathsf{Imp}(P)$ and $S'_Q \in \mathsf{Imp}(Q)$ and $\sigma' \in \mathsf{Traces}(S'_P \parallel S'_Q)$ such that $\sigma'|_{\Sigma^{Obs}_{P\otimes Q}} = \sigma$, we let*

$$\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q(\sigma) := \inf\{ \quad comb(\mathcal{F}_P(\sigma''|_{\Sigma^{Obs}_P}), \mathcal{F}_Q(\sigma''|_{\Sigma^{Obs}_Q})) \mid S_P \in \mathsf{Imp}(P) \text{ and } S_Q \in \mathsf{Imp}(Q) \text{ composable}, \sigma'' \in \mathsf{Traces}(S_P \parallel S_Q, \sigma|_{\Sigma^{I,E}_{P\otimes Q}})\},$$

and $\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q(\sigma)$ is undefined otherwise.

By taking the worst case in the definition of $\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q$, we guarantee that as long as A_P and A_Q are compatible, the reward interfaces P and Q are $\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q$-compatible, as shown in the next proposition. Composing P and Q results in the reward interface $(A_P \parallel A_Q, \mathcal{F}_P \nabla_{comb} \mathcal{F}_Q)$. P and Q can be implemented independently, and the composition of their implementations will be good enough with respect to $\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q$.

▶ **Proposition 18** ($\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q$-Compatibility). *Let $comb : \mathbb{R}_{-\infty} \times \mathbb{R}_{-\infty} \to \mathbb{R}_{-\infty}$. For all reward interfaces $P = (A_P, \mathcal{F}_P)$ and $Q = (A_Q, \mathcal{F}_Q)$ with A_P and A_Q compatible, it holds that P and Q are $\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q$-compatible.*

We discard any observable traces not resulting from implementations of P and Q, restricting the observable behaviors of interface automata good enough w.r.t. $\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q$ to those resulting from some $S_P \parallel S_Q$ where $S_P \in \mathsf{Imp}(P)$ and $S_Q \in \mathsf{Imp}(Q)$. This composition is associative if the function *comb* used to combine the values is associative, and satisfies some monotonicity and continuity conditions (Proposition 29).

▶ **Proposition 19** (Quality of the Reward Function Composition). *Consider reward interfaces $P = (A_P, \mathcal{F}_P)$ and $Q = (A_Q, \mathcal{F}_Q)$ with A_P and A_Q compatible, and a reward function $\mathcal{F}$ such that P and Q are $\mathcal{F}$-compatible. Then, $\mathsf{Imp}(A_P \parallel A_Q, \mathcal{F}_P \nabla_{comb} \mathcal{F}_Q) \subseteq \mathsf{Imp}(A_P \parallel A_Q, \mathcal{F})$.*

▶ **Example 20.** Let us consider components P and Q from Example 1 from the perspective of bottom-up design. Recall the reward function $\mathcal{F}_P$ from Example 14, where we impose that for input sequences containing both e_1 and e_2, there must be outputs o_1 and p in the respective order. Now, suppose instead we use a function $\mathcal{F}'_P$, which ignores the requirements on p, meaning that it only asks for o_1 to occur after e_1 and q, and nothing else. For those input sequences with both e_1 and e_2, $\mathcal{F}'_P$ will always award a value of $\frac{1}{2}$ as long as o_1 is produced. Note that this does not disallow p, it simply does not require it.

A possible behavior of an implementation S'_P of $(A_P, \mathcal{F}'_P)$ is then to produce o_1 once receiving q, and be idle otherwise. In particular, it will never produce p. Implementations S_Q of Q will, as before, produce o_2 only after receiving e_2 and p.

It is easy to see that the new function $\mathcal{F}'_P$ allows implementations of $(A_P, \mathcal{F}'_P)$ which prevent Q from producing o_2, disabling the best value w.r.t. $\mathcal{F}_Q$. Since P has no knowledge of $\mathcal{F}_Q$, we cannot assume that S_P will be more cooperative than required by A_P and $\mathcal{F}'_P$. As A_P and A_Q are compatible, choosing *comb* to be the sum, we have the guarantee that any $S_P \parallel S_Q$ is good enough w.r.t. $\mathcal{F}'_P \triangledown_{comb} \mathcal{F}_Q$. This guarantee is however weaker than $\mathcal{F}_\mathsf{S}$. In this case, that means that for the mentioned input sequences featuring e_1 and e_2, where previously the best value of $\frac{1}{2} + \frac{1}{2}$ was possible, we will now achieve at least $\frac{1}{2} + 0$.

In bottom-up design, we can use $(A_P \parallel A_Q, \mathcal{F}'_P \triangledown_{comb} \mathcal{F}_Q)$ as the reward interface for the composition of $P' = (A_P, \mathcal{F}'_P)$ and Q. This captures the guarantees of the composed components at a higher level of the system, without the need to devise a new reward function and checking if P and Q are compatible with respect to it. ⌟

5 Reward Interface Refinement

Supporting different levels of abstraction for the same component is helpful in the design process, as it allows a coarse description of interactions and dependencies while also defining internal details. *Refinement* enables this independent implementability of an interface, permitting extended functionality as long as original obligations are maintained. For a reward interface Q to be a refinement of a reward interface P, we require as usual that Q works within the same environments as P, and that an implementation of Q is also an implementation of P. This means, in particular, that, similarly to classical interface automata, a refinement of P must accept at least all inputs of P and not produce more outputs than P. In addition, we must consider the reward function w.r.t. best-effort implementations.

The refinement relation for reward interfaces can be defined as the conjunction of refinement between the respective interface automata [Definition 8] and refinement between the respective reward functions. We define the notion of refinement for quantitative reward functions in the context of good-enough satisfaction. Intuitively, for a function $\mathcal{F}_Q$ to refine a function $\mathcal{F}_P$, each input sequence for Q must be *"as hopeful"* w.r.t. $\mathcal{F}_Q$ as it is w.r.t. $\mathcal{F}_P$, and the value assigned by $\mathcal{F}_Q$ to a trace must be matched by the value assigned by $\mathcal{F}_P$ to the corresponding trace in P. This captures the idea that $\mathcal{F}_Q$ inherits the obligations of $\mathcal{F}_P$ and assigns values in a way that is possible in $\mathcal{F}_P$. We formalize this intuition as the existence of a *helper function* $r_{PQ} : \mathbb{R}_{-\infty} \to \mathbb{R}_{-\infty}$, that suitably relates the ranges $\mathit{Vals}(\mathcal{F}_P)$ and $\mathit{Vals}(\mathcal{F}_Q)$ of the functions. The idea is that r_{PQ} preserves the relative order of the values in $\mathcal{F}_P$, as it relates to both the hopefulness of input traces and values assigned to the respective full traces.

▶ **Definition 21** (Reward Function Refinement)**.** *Suppose that for the reward interfaces $P = (A_P, \mathcal{F}_P)$ and $Q = (A_Q, \mathcal{F}_Q)$ we have $\Sigma^I_P \subseteq \Sigma^I_Q$, $\Sigma^E_P \subseteq \Sigma^E_Q$ and $\Sigma^O_Q \subseteq \Sigma^O_P$. We say that $\mathcal{F}_Q$* refines *$\mathcal{F}_P$, denoted $\mathcal{F}_Q \preceq \mathcal{F}_P$, iff there exists a function $r_{PQ} : \mathbb{R}_{-\infty} \to \mathbb{R}_{-\infty}$ such that for every value $v \in \mathit{Vals}(\mathcal{F}_P)$, the following two conditions are satisfied.*

1. *For every $\sigma^{I,E}_Q \in (\Sigma^I_Q \cup \Sigma^E_Q)^\infty$, if $\sigma^{I,E}_Q|_{\Sigma_P} \in \mathit{Hopeful}(\mathcal{F}_P, v, \Sigma^I_P \cup \Sigma^E_P)$, then $\sigma^{I,E}_Q \in \mathit{Hopeful}(\mathcal{F}_Q, r_{PQ}(v), \Sigma^I_Q \cup \Sigma^E_Q)$.*
2. *For every $\sigma_Q \in (\Sigma^{Obs}_Q)^\infty$, if $\mathcal{F}_Q(\sigma_Q) \geq r_{PQ}(v)$ then $\mathcal{F}_P(\sigma_Q|_{\Sigma_P}) \geq v$.*

Definition 21 captures the nature of refinement, because while it retains priority brackets for obligations of the original function, we can also introduce new dependencies or arbitrary subdivisions within each priority. The next example illustrates the refinement relation for reward functions and demonstrates the use of the helper function for establishing the relation.

▶ **Example 22.** Going back to the reward interface for S with $\mathcal{F}_\mathsf{S}$ from Example 1, suppose we want to modify the guarantees, requiring S to output o_1 or o_2 for a period of time. More concretely, we ask for o_1 to hold as often as possible from the point e_1 is first received until the first occurrence of e_2, as to capture the change in input. We add the same requirement on o_2 if e_2 occurs first. We modify the existing reward function $\mathcal{F}_\mathsf{S}$ to $\mathcal{F}_R$, where we multiply the assigned value by a factor of $\frac{2}{3}$ if this new condition is violated.

Clearly, a subset of traces that had value 1 for $\mathcal{F}_\mathsf{S}$ are now assigned value $\frac{2}{3}$, namely where the other output o_3 is interjected. The same holds for traces going from $\frac{1}{2}$ to $\frac{1}{3}$. The new requirement will always be violated if the order was not preserved in the first place, giving us $\frac{1}{6}$ instead of $\frac{1}{4}$. However, the old obligations are kept: Any $(\mathcal{F}_\mathsf{S}, 1)$-hopeful input will be paired with a trace that for $\mathcal{F}_R$ achieves at least $\frac{2}{3}$. The function $r_{\mathsf{S}R}$ will then be $r_{\mathsf{S}R} = \{(0,0), (\frac{1}{4}, \frac{1}{6}), (\frac{1}{2}, \frac{1}{3}), (1, \frac{2}{3})\}$. Thus, the priorities from $\mathcal{F}_\mathsf{S}$ are preserved in the refinement, even though we are able to strengthen requirements in the reward function $\mathcal{F}_R$. ⌟

The refinement relation for reward interfaces combines the two refinement relations.

▶ **Definition 23** (Reward Interface Refinement). *A reward interface $Q = (A_Q, \mathcal{F}_Q)$* refines *a reward interface $P = (A_P, \mathcal{F}_P)$, written as $Q \preceq P$, if and only if $A_Q \preceq A_P$ and $\mathcal{F}_Q \preceq \mathcal{F}_P$.*

The rest of this section is dedicated to establishing that the refinement relation we defined has the properties necessary for compositional design. We establish that the refinement relation $\preceq$ between reward interfaces is a preorder, i.e. it is reflexive and transitive (Proposition 30). In particular, transitivity of refinement is important, as it allows for an iterative design process, where each refinement needs to only consider the last to maintain a refinement relation to the original specification. It is established by composing the respective helper functions.

As a consequence, an implementation of a refinement is then also an implementation of the original interface. Since the implementation must be good-enough w.r.t. the refined function, by refinement we can show it is also good-enough w.r.t. the original function.

▶ **Theorem 24** (Implementation of a Refinement). *If a reward interface $Q = (A_Q, \mathcal{F}_Q)$ refines a reward interface $P = (A_P, \mathcal{F}_P)$, then $\mathsf{Imp}(Q) \subseteq \mathsf{Imp}(P)$.*

The other direction of Theorem 24 does not hold in general. First of all, a reward interface constrains the set of implementations through both the interface automaton and the reward function, while the refinement relation treats those two components separately, which makes it stronger than the inclusion relation between the sets of implementations. Moreover, the refinement relation on reward functions alone is stronger than the inclusion between the respective sets of good-enough automata.

The last properties we establish enable independent implementability, by guaranteeing that refinement does not impair the original context of an interface. They concern refinement in the function composition and the property of *substitutability*. Intuitively, substitutability states that a refinement of a reward interface P remains $\mathcal{F}$-compatible with any reward interfaces that are $\mathcal{F}$-compatible with P.

▶ **Theorem 25.** *Consider $P = (A_P, \mathcal{F}_P)$ and $Q = (A_Q, \mathcal{F}_Q)$ with A_P and A_Q compatible, and let $P' = (A_{P'}, \mathcal{F}_{P'})$ be a non-empty refinement of P, with $(\Sigma^I_{P'} \setminus \Sigma^I_P) \cap \Sigma^O_Q = \emptyset$. Then if comb is monotonically increasing, $(A_{P'} \parallel A_Q, \mathcal{F}_{P'} \nabla_{comb} \mathcal{F}_Q) \preceq (A_P \parallel A_Q, \mathcal{F}_P \nabla_{comb} \mathcal{F}_Q)$. If P and Q are $\mathcal{F}$-compatible for a reward function $\mathcal{F}$, then P' and Q are also $\mathcal{F}$-compatible.*

With that, we established that the refinement for reward interfaces in Definition 23 indeed lifts the ideas of interface refinement to the best-effort quantitative setting and has the properties to enable component-based design of systems with quantitative specifications.

6 Checking Compatibility, Refinement, and Implementability

In this section, we study the algorithmic aspects of our theory of reward interfaces, in the context of an automata-based finite representation of the reward functions. We first introduce this representation. Then, we define the decision problems of interest, namely checking compatibility, refinement and implementability of reward interfaces, and outline algorithms for each of them for our representation. Lastly, we discuss quantitative temporal logics and weighted automata as representations of reward functions, and relate them to the representation studied in this section.

6.1 Automata-Based Finite Representation

The framework we introduced is designed to be general, including the representation of the reward function. Here, we study one possible representation, which is based on automata over finite and infinite words. We need both finite automata to model terminating components, as well as ω-automata to account for infinite behaviors of reactive systems.

We consider functions $\mathcal{F} : \Sigma^\infty \to \mathbb{R}$ such that $\mathit{Vals}(\mathcal{F})$ is finite, and which have a finite representation as a set of pairs of automata $\Phi = \{(\mathcal{B}_v, \mathcal{N}_v) \mid v \in \mathit{Vals}(\mathcal{F})\}$, where for each $v \in \mathit{Vals}(\mathcal{F})$, $\mathcal{B}_v$ is an NBA, and $\mathcal{N}_v$ is an NFA, and $\mathcal{L}(\mathcal{B}_v) \cup \mathcal{L}(\mathcal{N}_v) = \mathcal{F}^{=v}$. Intuitively, for each of the finitely many possible values of $\mathcal{F}$, the set of finite (resp. infinite) words that $\mathcal{F}$ maps to that value is given as a NFA (resp. NBA). We denote with $[\![\Phi]\!]$ the function $\mathcal{F} : \Sigma^\infty \to \mathbb{R}$ represented by Φ. For a reward interface $P = (A_P, \mathcal{F}_P)$ where the function $\mathcal{F}_P$ is given in a finite representation Φ_P, we abuse notation and write $P = (A_P, \Phi_P)$, and $\mathit{Vals}(\Phi_P)$ instead of $\mathit{Vals}([\![\Phi_P]\!])$. The size of Φ_P is given by $|\Phi_P| = \sum_{v \in \mathit{Vals}(\mathcal{F}_P)} (|\mathcal{B}_v| + |\mathcal{N}_v|)$.

6.2 Decision Problems and Automata-Based Algorithms

Assuming reward functions $[\![\Phi]\!]$ given as defined above, with pairs of automata for each value, we now study the main decision problems for reward interfaces.

▶ **Theorem 26** (Checking Compatibility of Reward Interfaces). *For a given finitely represented reward function $[\![\Phi]\!] : (\Sigma^{Obs}_{P \otimes Q})^\infty \to \mathbb{R}$, checking if $P = (A_P, \Phi_P)$ and $Q = (A_Q, \Phi_Q)$ are $[\![\Phi]\!]$-compatible can be done in double exponential time.*

Proof Sketch. We reduce the second condition of Definition 15 to checking language inclusion for tree automata, by constructing an automaton capturing the set of joint implementations $S_P \parallel S_Q$ s.t. $S_P \in \mathsf{Imp}(P)$ and $S_Q \in \mathsf{Imp}(Q)$. To this end, we construct a deterministic interface automaton from $A_P \parallel A_Q$, which incurs an exponential blowup in the worst case. We construct two universal co-Büchi word automata, one for implementations $S_P \parallel S_Q$, and one for good-enough sequences w.r.t. $[\![\Phi]\!]$, that are converted to universal co-Büchi tree automata. Checking language inclusion can be done in exponential time, thus in total we can check reward interface compatibility in time double exponential in $|A_P| \cdot |A_Q| \cdot |\Phi_P| \cdot |\Phi_Q|$. ◀

▶ **Theorem 27** (Checking Reward Interface Refinement). *Checking if $(A_Q, \Phi_Q) \preceq (A_P, \Phi_P)$ can be done in exponential time.*

Proof Sketch. Checking $A_Q \preceq A_P$ is done in polynomial time [14]. To find a candidate function r_{PQ} according to condition 1 of Definition 21, we iteratively match values of $\mathit{Vals}(\Phi_P)$ to those in $\mathit{Vals}(\Phi_Q)$. We compare sets of hopeful input sequences for the respective pairs of values in exponential time, by checking language inclusion. If a candidate r_{PQ} was constructed, we verify condition 2 of Definition 21, by checking that for every complete trace achieving value $r_{PQ}(v)$ on $[\![\Phi_Q]\!]$, the corresponding traces achieve value v for $[\![\Phi_P]\!]$. This is done by checking language emptiness. The overall procedure runs in exponential time. ◀

▶ **Theorem 28** (Implementations of a Reward Interface)**.**
Checking if $S \in \mathsf{Imp}((A_P, \Phi_P))$ *for an interface automaton* S *can be done in polynomial time. Checking if* $\mathsf{Imp}(P) \neq \emptyset$ *for* $P = (A_P, \Phi_P)$ *can be done in time exponential in* $|A_P| \cdot |\Phi_P|$.

Proof Sketch. From Φ_P, we construct an NBA whose language contains all counterexamples a good-enough implementation must not produce, whose size is polynomial in $|\Phi_P|$. We check for language emptiness of its product with S, which can be done in polynomial time.

For checking the existence of an implementation, we construct from the automaton accepting counterexamples a deterministic parity automaton for the positive case, incurring in the worst case an exponential blow-up in the number of states. We intersect the result with a deterministic automaton obtained from A_P to restrict the language to traces with corresponding executions on A_P. From the combined automaton we construct a parity game and solve it. If there exists a winning strategy, it gives us a possible implementation of P. If there is no such strategy, then there exists no implementation. Since the number of states in the game is exponential in $|A_P| \cdot |\Phi_P|$, the overall check can be done in exponential time. ◀

6.3 Discussion on Reward Functions and their Representation

The automata-based finite representation of reward functions allows system designers to express a quantitative language with a finite set of possible values in a convenient way, by describing the languages mapped to individual values as finite automata, providing a ranking for sets of execution traces with respect to how desirable these executions are.

This finite representation enables higher-level specification languages for quantitative requirements, such as LTL[$\mathcal{F}$] [1], which is a temporal logic with quantitative features, and the derived logic LTL$_f$[$\mathcal{F}$] [5] over finite traces. Such logic formulas can be seen as functions mapping infinite and finite words, respectively, to values in $\mathbb{R}$. The range of values for LTL[$\mathcal{F}$] specifications is finite [1]. Following the construction in [1], we can translate, in exponential time, LTL[$\mathcal{F}$] and LTL$_f$[$\mathcal{F}$] formulas into the above automata representation.

A reward function can also capture a quantitative language $L_{\mathcal{W}} : \Sigma^\infty \to \mathbb{R}$, expressed with a *weighted automaton* $\mathcal{W} = \langle Q, q_I, \Sigma, \delta, \gamma \rangle$, where $\gamma : \delta \to \mathbb{Q}$ is a weight function, and a *value function* $\mathsf{V} : \mathbb{Q}^\infty \to \mathbb{R}$ [8]. These automata can serve as a concise specification in the design process. To use the finite representation above and enable the respective algorithms, we restrict the instances of V we consider to those where *Vals* finite. For the value functions *Last* or *Max* on finite sequences, and *Sup*, *LimSup* or *LimInf* on infinite sequences, we can construct in at most polynomial time [8], from a quantitative automaton $\mathcal{W}$, individual (Boolean) automata $\mathcal{W}^{\sim v}$ for threshold v and $\sim \in \{>, \geq, =, \leq, <\}$, such that $L_{\mathcal{W}^{\sim v}} = \{\sigma \in \Sigma^\infty \mid L_{\mathcal{W}}(\sigma) \sim v\}$. We can thus translate a pair of quantitative automata with those value functions into a set of automata $\{(\mathcal{B}_v, \mathcal{N}_v) \mid v \in \mathit{Vals}\}$. While our theory supports more expressive classes of quantitative languages, such as *LimAvg* or *Disc*, there the language inclusion problem becomes undecidable for nondeterministic automata [8] and deterministic automata lack many essential closure properties [7]. In the future, we plan to investigate decidable subclasses, in order to extend algorithmic support to more expressive representations of reward functions.

7 Related Work

De Alfaro and Henzinger's *interface automata* [14] have been extended in various ways, including the timed domain [16, 12], and focusing on communication and synchronization using game semantics [13]. The work on *contract-based design* by Benveniste et al. [3] provides a comprehensive overview of different interface theories, with a unified set of properties.

Modal interface theories [20, 23, 21] refine the model by introducing modalities, which enable the expression of liveness properties. Extensions by Tripakis et al. [25], or Mouelhi et al. [22] extend the expressive power of interface automata by introducing additional contracts over the input and output variables at each state and globally. A similar idea is presented concerning shared memory [24], where the pre- and post-conditions on transitions are interpreted as modification of shared data on synchronization. These approaches lift interface automata for more expressivity and finer control. We similarly address these issues with requirements on interfaces expressed as a reward function, which by its multi-valued range is able to capture complex specifications in a concise way, and additionally, we define compatibility with respect to arbitrary functions to express higher level goals.

Quantitative aspects of interface automata have been explored in the form of *resource interfaces* [6], which extend stateful interfaces to include resource labels. Each state is associated with an increase or decrease in the value of an execution if entered. This allows for naturally specifying minimum resource usage, or checking compatibility within a threshold of consumed energy. For a single interface, *reward interfaces* are able to express the same using the reward function, but our compatibility notion is less strict outside the automaton structure. We could however use a resource interface in place of the function, similarly to a weighted automaton, since it will associate a value for each execution. Still, the semantics will be different since we interpret obligations given by the function in a good-enough context.

Our approaches to algorithmically checking consistency and compatibility of interfaces reduce to synthesis questions, where we need to find a good-enough strategy for an infinite two-player game. Good-enough synthesis [2] is closely related to dominant [11] and admissible [4] strategies, which are strategies that perform as good as the best alternative, such that they are not dominated by another strategy. There has been further work on applying these ideas to the compositional synthesis setting [11, 19, 17].

8 Conclusion

We introduced a novel interface framework for best-effort quantitative requirements, called reward interfaces, building on interface automata. Our formalism can enforce high-quality implementations for unconstrained external environments, while providing notions of compatibility and refinement central to interface theories. It allows for expressing a wide range of both qualitative and quantitative properties in a concise manner, as to not increase the effort required during the component-based design process. The algorithmic solutions we presented are of considerably higher complexity than those for classic interface automata, due to the high degree of flexibility afforded by the quantitative function.

References

1 Shaull Almagor, Udi Boker, and Orna Kupferman. Formally reasoning about quality. *J. ACM*, 63(3):24:1–24:56, 2016. doi:10.1145/2875421.

2 Shaull Almagor and Orna Kupferman. Good-enough synthesis. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 541–563. Springer, 2020. doi:10.1007/978-3-030-53291-8_28.

3 Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto L. Sangiovanni-Vincentelli, Werner Damm, Thomas A. Henzinger, and Kim G. Larsen. Contracts for system design. *Found. Trends Electron. Des. Autom.*, 12(2-3):124–400, 2018. doi:10.1561/1000000053.

4 Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-admissible synthesis. *Acta Informatica*, 54(1):41–83, 2017. doi:10.1007/s00236-016-0273-2.

5 Alberto Camacho, Meghyn Bienvenu, and Sheila A. McIlraith. Finite LTL synthesis with environment assumptions and quality measures. *CoRR*, abs/1808.10831, 2018. arXiv:1808.10831.

6 Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource interfaces. In Rajeev Alur and Insup Lee, editors, *Embedded Software*, pages 117–133, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. doi:10.1007/978-3-540-45212-6_9.

7 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Expressiveness and closure properties for quantitative languages. In *Proceedings of the 2009 24th Annual IEEE Symposium on Logic In Computer Science*, LICS '09, pages 199–208, USA, 2009. IEEE Computer Society. doi:10.1109/LICS.2009.16.

8 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4):23:1–23:38, 2010. doi:10.1145/1805950.1805953.

9 Taolue Chen, Chris Chilton, Bengt Jonsson, and Marta Kwiatkowska. A compositional specification theory for component behaviours. In Helmut Seidl, editor, *Programming Languages and Systems*, pages 148–168, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-28869-2_8.

10 Chris Chilton, Bengt Jonsson, and Marta Kwiatkowska. An algebraic theory of interface automata. *Theoretical Computer Science*, 549:146–174, 2014. doi:10.1016/j.tcs.2014.07.018.

11 Werner Damm and Bernd Finkbeiner. Automatic compositional synthesis of distributed systems. In Cliff B. Jones, Pekka Pihlajasaari, and Jun Sun, editors, *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, volume 8442 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 2014. doi:10.1007/978-3-319-06410-9_13.

12 Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Timed i/o automata: a complete specification theory for real-time systems. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '10, pages 91–100, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1755952.1755967.

13 Luca de Alfaro, Leandro Dias da Silva, Marco Faella, Axel Legay, Pritam Roy, and Maria Sorea. Sociable interfaces. In Bernhard Gramlich, editor, *Frontiers of Combining Systems*, pages 81–105, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/11559306_5.

14 Luca de Alfaro and Thomas A. Henzinger. Interface automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, 2001. doi:10.1145/503271.503226.

15 Luca de Alfaro and Thomas A. Henzinger. Interface-based design. In Manfred Broy, Johannes Grünbauer, David Harel, and Tony Hoare, editors, *Engineering Theories of Software Intensive Systems*, pages 83–104, Dordrecht, 2005. Springer Netherlands.

16 Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Timed interfaces. In Alberto Sangiovanni-Vincentelli and Joseph Sifakis, editors, *Embedded Software*, pages 108–122, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. doi:10.1007/3-540-45828-X_9.

17 Rafael Dewes and Rayna Dimitrova. Compositional high-quality synthesis. In Étienne André and Jun Sun, editors, *Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part I*, volume 14215 of *Lecture Notes in Computer Science*, pages 334–354. Springer, 2023. doi:10.1007/978-3-031-45329-8_16.

18 Laurent Doyen, Thomas A. Henzinger, Barbara Jobstmann, and Tatjana Petrov. Interface theories with component reuse. In *Proceedings of the 8th ACM International Conference on Embedded Software*, EMSOFT '08, pages 79–88, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1450058.1450070.

19 Bernd Finkbeiner and Noemi Passing. Dependency-based compositional synthesis. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, volume 12302 of *Lecture Notes in Computer Science*, pages 447–463. Springer, 2020. doi:10.1007/978-3-030-59152-6_25.

20 Kim G. Larsen, Ulrik Nyman, and Andrzej Wąsowski. Modal i/o automata for interface and product line theories. In Rocco De Nicola, editor, *Programming Languages and Systems*, pages 64–79, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

21 Gerald Lüttgen and Walter Vogler. Modal interface automata. In Jos C. M. Baeten, Thomas Ball, and Frank S. de Boer, editors, *Theoretical Computer Science - 7th IFIP TC 1/WG 2.2 International Conference, TCS 2012, Amsterdam, The Netherlands, September 26-28, 2012. Proceedings*, volume 7604 of *Lecture Notes in Computer Science*, pages 265–279. Springer, 2012. doi:10.1007/978-3-642-33475-7_19.

22 Sebti Mouelhi, Samir Chouali, and Hassan Mountassir. Refinement of interface automata strengthened by action semantics. In *FESCA@ETAPS*, volume 253 of *Electronic Notes in Theoretical Computer Science*, pages 111–126. Elsevier, 2009. doi:10.1016/J.ENTCS.2009.09.031.

23 Jean-Baptiste Raclet, Éric Badouel, Albert Benveniste, Benoît Caillaud, Axel Legay, and Roberto Passerone. A modal interface theory for component-based design. *Fundam. Informaticae*, 108(1-2):119–149, 2011. doi:10.3233/FI-2011-416.

24 Ayleen Schinko, Walter Vogler, Johannes Gareis, N. Tri Nguyen, and Gerald Lüttgen. Interface automata for shared memory. *Acta Informatica*, 59(5):521–556, 2022. doi:10.1007/S00236-021-00408-8.

25 Stavros Tripakis, Ben Lickly, Thomas A. Henzinger, and Edward A. Lee. A theory of synchronous relational interfaces. *ACM Trans. Program. Lang. Syst.*, 33(4), July 2011. doi:10.1145/1985342.1985345.

A Appendix: Compatibility of Reward Interfaces

▶ **Proposition 18** ($\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q$-Compatibility). *Let $comb : \mathbb{R}_{-\infty} \times \mathbb{R}_{-\infty} \to \mathbb{R}_{-\infty}$. For all reward interfaces $P = (A_P, \mathcal{F}_P)$ and $Q = (A_Q, \mathcal{F}_Q)$ with A_P and A_Q compatible, it holds that P and Q are $\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q$-compatible.*

Proof. Let $P = (A_P, \mathcal{F}_P)$ and $Q = (A_Q, \mathcal{F}_Q)$ be reward interfaces where A_P and A_Q are compatible. To prove that P and Q are $\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q$-compatible, we need to show that for any pair of implementations $S_P \in \mathsf{Imp}(P)$ and $S_Q \in \mathsf{Imp}(Q)$ that are composable, it holds that $S_P \parallel S_Q$ is good enough w.r.t. $\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q$. For the sake of contradiction, assume that there exist composable $S_P \in \mathsf{Imp}(P)$ and $S_Q \in \mathsf{Imp}(Q)$ such that $S_P \parallel S_Q$ is not good-enough w.r.t. $\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q$. That means that for some $v \in \mathit{Vals}(\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q)$ there exists an input sequence $\sigma_{E,I} \in (\Sigma_{P\otimes Q}^{E,I})^\infty$ that witnesses this violation, that is,

- $\sigma_{E,I} \in \mathit{Hopeful}(\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q, v, \Sigma_{P\otimes Q}^{E,I})$, and
- for some $\sigma \in \mathsf{Traces}(S_P \parallel S_Q, \sigma_{E,I})$, $\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q(\sigma)$ is undef. or $\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q(\sigma) < v$.

By the first item above, together with the definition of hopeful sequences (Definition 11) and the definition of $\mathcal{F}_P \nabla_{comb} \mathcal{F}_Q$ (Definition 17), there exist composable $S'_P \in \mathsf{Imp}(P)$ and $S'_Q \in \mathsf{Imp}(Q)$ and $\sigma' \in \mathsf{Traces}(S'_P \parallel S'_Q)$ such that $\sigma'|_{\Sigma_{P\otimes Q}^{E,I}} = \sigma_{E,I}$ and

$$\inf\{comb(\mathcal{F}_P(\sigma''|_{\Sigma_P^{Obs}}), \mathcal{F}_Q(\sigma''|_{\Sigma_Q^{Obs}})) \mid S''_P \in \mathsf{Imp}(P) \text{ and } S''_Q \in \mathsf{Imp}(Q) \text{ composable}, \sigma'' \in \mathsf{Traces}(S''_P \parallel S''_Q, \sigma_{E,I})\} \geq v.$$

Since $\sigma \in \mathsf{Traces}(S_P \parallel S_Q, \sigma_{E,I})$, by the definition of $\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q$, we have that $\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q(\sigma)$ is defined. Thus, by the second item we have that $\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q(\sigma) < v$. This, together with the definition of $\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q$ implies that

$$\inf\{comb(\mathcal{F}_P(\sigma''|_{\Sigma_P^{Obs}}), \mathcal{F}_Q(\sigma''|_{\Sigma_Q^{Obs}})) \mid S''_P \in \mathsf{Imp}(P) \text{ and } S''_Q \in \mathsf{Imp}(Q) \text{ composable}, \sigma'' \in \mathsf{Traces}(S''_P \parallel S''_Q, \sigma_{E,I})\} < v.$$

This is a contradiction, which concludes the proof. ◀

▶ **Proposition 19** (Quality of the Reward Function Composition). *Consider reward interfaces $P = (A_P, \mathcal{F}_P)$ and $Q = (A_Q, \mathcal{F}_Q)$ with A_P and A_Q compatible, and a reward function $\mathcal{F}$ such that P and Q are $\mathcal{F}$-compatible. Then, $\mathsf{Imp}(A_P \parallel A_Q, \mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q) \subseteq \mathsf{Imp}(A_P \parallel A_Q, \mathcal{F})$.*

Proof. Let $P = (A_P, \mathcal{F}_P)$ and $Q = (A_Q, \mathcal{F}_Q)$ be reward interfaces with automata A_P and A_Q that are compatible. Let $\mathcal{F}$ be a reward function such that P and Q are $\mathcal{F}$-compatible.

We have to show that for every $S \in \mathsf{Imp}(A_P \parallel A_Q, \mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q)$ it holds that $S \in \mathsf{Imp}(A_P \parallel A_Q, \mathcal{F})$. For the sake of contradiction, suppose that there exists $S \in \mathsf{Imp}(A_P \parallel A_Q, \mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q)$ such that $S \notin \mathsf{Imp}(A_P \parallel A_Q, \mathcal{F})$. Since $S \preceq A_P \parallel A_Q$, this means that S is not good-enough with respect to $\mathcal{F}$. Let $\sigma \in \mathsf{Traces}(S)$ be the trace witnessing this violation. Thus, $\sigma|_{\Sigma^{E,I}_{P\otimes Q}}$ is $(\mathcal{F}, v)$-hopeful for some $v \in \mathbb{R}_{-\infty}$ and $\mathcal{F}(\sigma|_{\Sigma^{Obs}_{P\otimes Q}})$ is either undefined or strictly smaller than v. We will now show that this assumption leads to contradiction.

Let $\mathcal{I} = \{S_P \parallel S_Q \mid S_P \in \mathsf{Imp}(P), S_Q \in \mathsf{Imp}(Q), S_P \text{ and } S_Q \text{ composable}\}$ be the set of composed implementations of P and Q. Since implementations do not restrict the allowed inputs, we have that there exists $S' \in \mathcal{I}$ and $\sigma' \in \mathsf{Traces}(S')$ such that $\sigma'|_{\Sigma^{E,I}_{P\otimes Q}} = \sigma|_{\Sigma^{E,I}_{P\otimes Q}}$. This implies that $\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q(\sigma|_{\Sigma^{E,I}_{P\otimes Q}})$ is defined. Thus, since S is good-enough with respect to $\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q$, we have that $\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q(\sigma|_{\Sigma^{Obs}_{P\otimes Q}})$ must be defined as well. By the definition of $\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q$, this implies that there exist $S'' \in \mathcal{I}$ and $\sigma'' \in \mathsf{Traces}(S'')$ such that $\sigma''|_{\Sigma^{Obs}_{P\otimes Q}} = \sigma|_{\Sigma^{Obs}_{P\otimes Q}}$. Now, since $\sigma''|_{\Sigma^{E,I}_{P\otimes Q}} = \sigma|_{\Sigma^{E,I}_{P\otimes Q}}$, and by Proposition 18 S'' is good-enough with respect to $\mathcal{F}$, it must be the case that $\mathcal{F}(\sigma''|_{\Sigma^{Obs}_{P\otimes Q}}) \geq v$. Since $\sigma''|_{\Sigma^{Obs}_{P\otimes Q}} = \sigma|_{\Sigma^{Obs}_{P\otimes Q}}$, it also holds that $\mathcal{F}(\sigma|_{\Sigma^{Obs}_{P\otimes Q}}) \geq v$, which is the desired contradiction. ◀

▶ **Proposition 29** (Composition Associativity). *Let $P = (A_P, \mathcal{F}_P), Q = (A_Q, \mathcal{F}_Q)$ and $R = (A_R, \mathcal{F}_R)$ be reward interfaces with pairwise compatible interface automata. If the function comb is symmetric, then $\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q = \mathcal{F}_Q \triangledown_{comb} \mathcal{F}_P$. If comb is associative, monotonically increasing and continuous, then the reward function composition using comb is associative, that is, $(\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q) \triangledown_{comb} \mathcal{F}_R = \mathcal{F}_P \triangledown_{comb} (\mathcal{F}_Q \triangledown_{comb} \mathcal{F}_R)$.*

Proof. Let $P = (A_P, \mathcal{F}_P), Q = (A_Q, \mathcal{F}_Q)$ and $R = (A_R, \mathcal{F}_R)$ be reward interfaces with pairwise compatible interface automata. Clearly, $A_P \parallel A_Q = A_Q \parallel A_P$. The associativity of composition for compatible interface automata is established in [14], thus $(A_P \parallel A_Q) \parallel A_R = A_P \parallel (A_Q \parallel A_R)$. Let $A_{PQ} := A_P \parallel A_Q$ and $A_{QR} := A_Q \parallel A_R$.

By the definition of composition, it is clearly the case that for every $\sigma \in \Sigma^{Obs}_{P\otimes Q}$ we have that $\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q(\sigma)$ is defined if and only if $\mathcal{F}_Q \triangledown_{comb} \mathcal{F}_P(\sigma)$ is defined. When *comb* is symmetric, that, for all $a, b \in \mathbb{R}_{-\infty}$ we have that $comb(a, b) = comb(b, a)$, the definition of reward function composition (Definition 17) implies that if defined, the two values are equal.

Suppose that the function *comb* satisfies the following conditions for all $a, b, c, d, d' \in \mathbb{R}_{-\infty}$:

- (Associativity) $comb(comb(a, b), c) = comb(a, comb(b, c))$.
- (Monotonicity) If $d \leq d'$, then $comb(d, b) \leq comb(d', b)$ and $comb(a, d) \leq comb(a, d')$.
- (Continuity) For every fixed $v \in \mathbb{R}_{-\infty}$, the functions $f_1(x) = comb(x, v)$ and $f_2(y) = comb(v, y)$ are continuous on $\mathbb{R}_{-\infty}$.

Under these conditions we prove that for any sequence $\sigma \in (\Sigma^{Obs}_{P\otimes Q\otimes R})^\infty$ it either holds that $((\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q) \triangledown_{comb} \mathcal{F}_R)(\sigma) = (\mathcal{F}_P \triangledown_{comb} (\mathcal{F}_Q \triangledown_{comb} \mathcal{F}_R))(\sigma)$, or both are undefined.

Let $\mathcal{F}_{PQ} := \mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q$, $\mathcal{F}_{QR} := \mathcal{F}_Q \triangledown_{comb} \mathcal{F}_R$, $PQ := (A_{PQ}, \mathcal{F}_{PQ})$ and $QR := (A_{QR}, \mathcal{F}_{QR})$, and let $A := (A_P \parallel A_Q) \parallel A_R = A_P \parallel (A_Q \parallel A_R)$. Let $\sigma \in (\Sigma^{Obs}_{P\otimes Q\otimes R})^\infty$.

First we show that $(\mathcal{F}_{PQ} \triangledown_{comb} \mathcal{F}_R)(\sigma)$ is defined iff $(\mathcal{F}_P \triangledown_{comb} \mathcal{F}_{QR})(\sigma)$ is defined.

Suppose that $(\mathcal{F}_{PQ} \triangledown_{comb} \mathcal{F}_R)(\sigma)$ is defined. Then, there exist composable implementations $S'_{PQ} \in \mathsf{Imp}(PQ)$ and $S'_R \in \mathsf{Imp}(R)$ and $\sigma' \in \mathsf{Traces}(S'_{PQ} \parallel S'_R)$ such that $\sigma'|_{\Sigma^{Obs}_{P\otimes Q\otimes R}} = \sigma$. Since S'_{PQ} is good-enough with respect to $\mathcal{F}_{PQ}$, from the properties of composition of reward functions we have that there exist composable $S'_P \in \mathsf{Imp}(P)$ and $S'_Q \in \mathsf{Imp}(Q)$ and a trace $\sigma'' \in \mathsf{Traces}(S'_P \parallel S'_Q)$ such that $\sigma''|_{\Sigma^{Obs}_{P\otimes Q}} = \sigma'|_{\Sigma^{Obs}_{P\otimes Q}}$. By the choice of σ' and σ'' we have that there exists $\sigma''' \in \mathsf{Traces}((S'_P \parallel S'_Q) \parallel S'_R)$ such that $\sigma'''|_{\Sigma^{Obs}_{P\otimes Q\otimes R}} = \sigma$. Since $\mathsf{Traces}((S'_P \parallel S'_Q) \parallel S'_R) = \mathsf{Traces}(S'_P \parallel (S'_Q \parallel S'_R))$, we can use S'_P, $S'_Q \parallel S'_R$ and σ''' as a witness to the fact that $(\mathcal{F}_P \triangledown_{comb} \mathcal{F}_{QR})(\sigma)$ is defined. The other direction of the implication is shown analogously, concluding the first part of the proof.

It remains to show that when both functions are defined on σ, then their values are equal.

By definition, we have $(\mathcal{F}_{PQ} \triangledown_{comb} \mathcal{F}_R)(\sigma) = \inf\{comb(\mathcal{F}_{PQ}(\sigma'|_{\Sigma^{Obs}_{P\otimes Q}}), \mathcal{F}_R(\sigma'|_{\Sigma^{Obs}_R}))|$ $S'_{PQ} \in \mathsf{Imp}(PQ), S'_R \in \mathsf{Imp}(R), \text{composable}, \sigma' \in \mathsf{Traces}(S'_{PQ} \parallel S'_R, \sigma|_{\Sigma^{E,I}_{P\otimes Q\otimes R}})\}$. Applying the definition to $\mathcal{F}_{PQ}$, we replace $\mathcal{F}_{PQ}(\sigma'|_{\Sigma^{Obs}_{P\otimes Q}})$ by $\inf\{comb(\mathcal{F}_P(\sigma''|_{\Sigma^{Obs}_P}), \mathcal{F}_Q(\sigma''|_{\Sigma^{Obs}_Q}))|$ $S'_P \in \mathsf{Imp}(P), S'_Q \in \mathsf{Imp}(Q), \text{composable}, \sigma'' \in \mathsf{Traces}(S'_P \parallel S'_Q, \sigma'|_{\Sigma^{E,I}_{P\otimes Q}})\}$.

Due to the monotonicity and continuity properties of $comb$, we get $(\mathcal{F}_{PQ} \triangledown_{comb} \mathcal{F}_R)(\sigma) =$ $\inf\{comb(comb(\mathcal{F}_P(\sigma''|_{\Sigma^{Obs}_P}), \mathcal{F}_Q(\sigma''|_{\Sigma^{Obs}_Q})), \mathcal{F}_R(\sigma'|_{\Sigma^{Obs}_R})) \mid S'_{PQ} \in \mathsf{Imp}(PQ), S'_R \in \mathsf{Imp}(R),$ $\text{composable}, \sigma' \in \mathsf{Traces}(S'_{PQ} \parallel S'_R, \sigma|_{\Sigma^{E,I}_{P\otimes Q\otimes R}}), S'_P \in \mathsf{Imp}(P), S'_Q \in$ $\mathsf{Imp}(Q), \text{composable}, \sigma'' \in \mathsf{Traces}(S'_P \parallel S'_Q, \sigma'|_{\Sigma^{E,I}_{P\otimes Q}})\}$. Applying reasoning similar to that in the first part of the proof, we get $(\mathcal{F}_{PQ} \triangledown_{comb} \mathcal{F}_R)(\sigma) =$ $\inf\{comb(comb(\mathcal{F}_P(\sigma'''|_{\Sigma^{Obs}_P}), \mathcal{F}_Q(\sigma'''|_{\Sigma^{Obs}_Q})), \mathcal{F}_R(\sigma'''|_{\Sigma^{Obs}_R})) \mid S'_P \in \mathsf{Imp}(P), S'_Q \in$ $\mathsf{Imp}(Q), \text{composable}, S'_R \in \mathsf{Imp}(R)$ composable with $S'_P \parallel S'_Q, \sigma''' \in \mathsf{Traces}(S'_P \parallel$ $S'_Q \parallel S'_R, \sigma|_{\Sigma^{E,I}_{P\otimes Q\otimes R}})\}$. Using the associativity of $comb$ we replace the term $comb(comb(\mathcal{F}_P(\sigma'''|_{\Sigma^{Obs}_P}), \mathcal{F}_Q(\sigma'''|_{\Sigma^{Obs}_Q})), \mathcal{F}_R(\sigma'''|_{\Sigma^{Obs}_R}))$ by $comb(\mathcal{F}_P(\sigma'''|_{\Sigma^{Obs}_P}), comb(\mathcal{F}_Q(\sigma'''|_{\Sigma^{Obs}_Q}), \mathcal{F}_R(\sigma'''|_{\Sigma^{Obs}_R})))$.

Additionally, reorganizing the composition, we obtain $(\mathcal{F}_{PQ} \triangledown_{comb} \mathcal{F}_R)(\sigma) =$ $\inf\{comb(\mathcal{F}_P(\sigma'''|_{\Sigma^{Obs}_P}), comb(\mathcal{F}_Q(\sigma'''|_{\Sigma^{Obs}_Q}), \mathcal{F}_R(\sigma'''|_{\Sigma^{Obs}_R}))) \mid S'_P \in \mathsf{Imp}(P),$ $S'_Q \parallel S'_R \in \mathsf{Imp}(QR), \text{composable}, \sigma''' \in \mathsf{Traces}(S'_P \parallel (S'_Q \parallel S'_R), \sigma|_{\Sigma^{E,I}_{P\otimes Q\otimes R}})\}$.

Using again the properties of $comb$, we get $(\mathcal{F}_{PQ} \triangledown_{comb} \mathcal{F}_R)(\sigma) =$ $\inf\{comb(\mathcal{F}_P(\sigma'''|_{\Sigma^{Obs}_P}), \mathcal{F}_{QR}(\sigma'''|_{\Sigma^{Obs}_{Q\otimes R}})) \mid S'_P \in \mathsf{Imp}(P), S'_Q \parallel S'_R \in \mathsf{Imp}(QR), \text{composable},$ $\sigma''' \in \mathsf{Traces}(S'_P \parallel (S'_Q \parallel S'_R), \sigma'|_{\Sigma^{E,I}_{P\otimes Q\otimes R}})\}$. This is precisely $(\mathcal{F}_P \triangledown_{comb} \mathcal{F}_{QR})(\sigma)$. ◀

B Appendix: Properties of Reward Interface Refinement

▶ **Proposition 30** (Refinement as Preorder). *For all reward interfaces P, Q and R, it holds that (1) $P \preceq P$ and (2) if $Q \preceq P$, and $R \preceq Q$, then also $R \preceq P$.*

Proof. Theorem 4.1 in [14] establishes that the refinement relation between classical interface automata is a preorder. Thus, we show only that reward function refinement is a preorder.

Consider reward interfaces $P = (A_P, \mathcal{F}_P)$, $Q = (A_Q, \mathcal{F}_Q)$, and $R = (A_R, \mathcal{F}_R)$.

(1) is trivial, so we show only (2). Assume that $P \succeq Q$ and $Q \succeq R$.

Since $A_P \succeq A_Q$ and $A_Q \succeq A_R$, we have alternating simulations $\succeq_{PQ} \subseteq V_P \times V_Q$ and $\succeq_{QR} \subseteq V_Q \times V_R$ from A_Q to A_P, and from A_R to A_Q, respectively. Furthermore we have that $\Sigma_P^E \subseteq \Sigma_Q^E \subseteq \Sigma_R^E$, $\Sigma_P^I \subseteq \Sigma_Q^I \subseteq \Sigma_R^I$, and $\Sigma_R^O \subseteq \Sigma_Q^O \subseteq \Sigma_P^O$. For $A_P \succeq A_R$, we define the relation $\succeq_{PR} \subset V_P \times V_R$ as $\succeq_{PR} := \{(u, v) \in V_P \times V_R \mid \exists w \in V_Q.\ u \succeq_{PQ} w \wedge w \succeq_{QR} v\}$. The relation $\succeq_{PR}$ is an alternating simulation from A_R to A_P. Thus, $A_P \succeq A_R$.

Since $\mathcal{F}_P \succeq \mathcal{F}_Q$ and $\mathcal{F}_Q \succeq \mathcal{F}_R$, there exist functions $r_{PQ} : \mathbb{R}_{-\infty} \to \mathbb{R}_{-\infty}$, and $r_{QR} : \mathbb{R}_{-\infty} \to \mathbb{R}_{-\infty}$, which satisfy the conditions of Definition 21. We define the function $r_{PR} : \mathbb{R}_{-\infty} \to \mathbb{R}_{-\infty}$ such that $r_{PR}(v) = r_{QR}(r_{PQ}(v))$ for all $v \in \mathit{Vals}(\mathcal{F}_P)$. We now show that r_{PR} satisfies the conditions of Definition 21. Let $v \in \mathit{Vals}(\mathcal{F}_P)$.

1. Let $\sigma_R^{I,E} \in (\Sigma_R^E \cup \Sigma_R^I)^\infty$ be such that $\sigma_R^{I,E}|_{\Sigma_P} \in \mathit{Hopeful}(\mathcal{F}_P, v, (\Sigma_P^E \cup \Sigma_P^I))$. We have to show that $\sigma_R^{I,E} \in \mathit{Hopeful}(\mathcal{F}_R, r_{PR}(v), (\Sigma_R^E \cup \Sigma_R^I))$. From the properties of r_{PQ}, we get $\sigma_R^{I,E}|_{\Sigma_Q} \in \mathit{Hopeful}(\mathcal{F}_Q, r_{PQ}(v), (\Sigma_Q^E \cup \Sigma_Q^I))$. From that, using the properties of r_{QR}, we get that $\sigma_R^{I,E} \in \mathit{Hopeful}(\mathcal{F}_R, r_{QR}(r_{PQ}(v)), (\Sigma_Q^E \cup \Sigma_Q^I))$. Since $r_{QR}(r_{PQ}(v)) = r_{PR}(v)$, this is precisely what we needed to show.
2. Let $\sigma_R \in (\Sigma_R^{Obs})^\infty$ be such that $\mathcal{F}_R(\sigma_R) \geq r_{PR}(v)$. We have to show that $\mathcal{F}_P(\sigma_R|_{\Sigma_P}) \geq v$. Since $r_{PR}(v) = r_{QR}(r_{PQ}(v))$, from the properties of r_{QR} we have that $\mathcal{F}_Q(\sigma_R|_{\Sigma_Q}) \geq r_{PQ}(v)$. From that, by the properties of the function r_{PQ} we get $\mathcal{F}_P((\sigma_R|_{\Sigma_Q})|_{\Sigma_P}) \geq v$. $(\sigma_R|_{\Sigma_Q})|_{\Sigma_P}$ is equal to $\sigma_R|_{\Sigma_P}$ because of the subset relationships between alphabets of P, Q, and R, as stated above, which is precisely what we had to show. ◀

▶ **Theorem 24** (Implementation of a Refinement). *If a reward interface $Q = (A_Q, \mathcal{F}_Q)$ refines a reward interface $P = (A_P, \mathcal{F}_P)$, then $\mathsf{Imp}(Q) \subseteq \mathsf{Imp}(P)$.*

Proof. Consider reward interfaces $P = (A_P, \mathcal{F}_P)$ and $Q = (A_Q, \mathcal{F}_Q)$ such that Q is a refinement of P. We have to show that $\mathsf{Imp}(Q) \subseteq \mathsf{Imp}(P)$. Let $S \in \mathsf{Imp}(Q)$. By Definition 13, we need to show that (a) $S \preceq A_P$, and (b) S is good-enough w.r.t. $\mathcal{F}_P$.

Refinement of interface automata is transitive, and since S is a refinement of A_Q, and A_Q refines A_P, condition (a) is satisfied. To establish condition (b), we have to show that for every $\sigma_{E,I} \in (\Sigma_P^E \cup \Sigma_P^I)^\infty$ with $\sigma_{E,I} \in \mathit{Hopeful}(\mathcal{F}_P, v, \Sigma_P^E \cup \Sigma_P^I)$ and every trace $\sigma \in \mathsf{Traces}_{\Sigma_P^{Obs}}(S, \sigma_{E,I})$ it holds that $\mathcal{F}_P(\sigma)$ is defined and $\mathcal{F}_P(\sigma) \geq v$.

Since $\mathcal{F}_Q \preceq \mathcal{F}_P$, there exists a function $r_{PQ} : \mathbb{R}_{-\infty} \to \mathbb{R}_{-\infty}$ that satisfies the conditions of Definition 21. Because $\sigma_{E,I} \in \mathit{Hopeful}(\mathcal{F}_P, v, \Sigma_P^E \cup \Sigma_P^I)$, and $(\Sigma_P^E \cup \Sigma_P^I) \subseteq (\Sigma_Q^E \cup \Sigma_Q^I)$, we know by the first property of r_{PQ} that $\sigma_{E,I} \in \mathit{Hopeful}(\mathcal{F}_Q, r_{PQ}(v), \Sigma_Q^E \cup \Sigma_Q^I)$. Then, since $S \in \mathsf{Imp}(Q)$ we have that for every trace $\sigma \in \mathsf{Traces}_{\Sigma_Q^{Obs}}(S, \sigma_{E,I})$, $\mathcal{F}_Q(\sigma)$ is defined and $\mathcal{F}_Q(\sigma) \geq r_{PQ}(v)$. Applying Definition 21, every trace $\sigma \in \mathsf{Traces}_{\Sigma_Q^{Obs}}(S, \sigma_{E,I})$ is such that $\mathcal{F}_P(\sigma|_{\Sigma_P})$ is defined and $\mathcal{F}_P(\sigma|_{\Sigma_P}) \geq v$. Conditions on r_{PQ} guarantee that the matching sequence $\sigma|_{\Sigma_P}$ achieves value v for $\mathcal{F}_P$. Since $\Sigma_P^O \supseteq \Sigma_Q^O \supseteq \Sigma_S^O$, $\{\sigma|_{\Sigma_P} \mid \sigma \in \mathsf{Traces}_{\Sigma_Q^{Obs}}(S, \sigma_{E,I})\} = \{\sigma \mid \sigma \in \mathsf{Traces}_{\Sigma_P^{Obs}}(S, \sigma_{E,I})\}$. Thus, it follows that for every trace $\sigma \in \mathsf{Traces}_{\Sigma_P^{Obs}}(S, \sigma_{E,I})$, $\mathcal{F}_P(\sigma)$ is defined and $\mathcal{F}_P(\sigma) \geq v$, which is what we had to prove. ◀

▶ **Theorem 25.** *Consider $P = (A_P, \mathcal{F}_P)$ and $Q = (A_Q, \mathcal{F}_Q)$ with A_P and A_Q compatible, and let $P' = (A_{P'}, \mathcal{F}_{P'})$ be a non-empty refinement of P, with $(\Sigma_{P'}^I \setminus \Sigma_P^I) \cap \Sigma_Q^O = \emptyset$. Then if comb is monotonically increasing, $(A_{P'} \parallel A_Q, \mathcal{F}_{P'} \nabla_{comb} \mathcal{F}_Q) \preceq (A_P \parallel A_Q, \mathcal{F}_P \nabla_{comb} \mathcal{F}_Q)$. If P and Q are $\mathcal{F}$-compatible for a reward function $\mathcal{F}$, then P' and Q are also $\mathcal{F}$-compatible.*

We will differentiate Theorem 25 into Lemma 31 and Lemma 32.

▶ **Lemma 31** (Substitutability). *Let $P = (A_P, \mathcal{F}_P)$ be $\mathcal{F}$-compatible with $Q = (A_Q, \mathcal{F}_Q)$ for some function $\mathcal{F} : (\Sigma_{P \otimes Q}^{Obs})^\infty \to \mathbb{R}$. Then, any non-empty refinement $P' = (A_{P'}, \mathcal{F}_{P'})$ of P with $(\Sigma_{P'}^I \setminus \Sigma_P^I) \cap \Sigma_Q^O = \emptyset$ is $\mathcal{F}$-compatible with Q.*

Proof. Since P and Q are $\mathcal{F}$-compatible, we know that all the following hold:

- A_P and A_Q are non-empty and composable,
- there exists a legal environment A_R for (A_P, A_Q), and
- for $S_P \in \mathsf{Imp}(P)$ and $S_Q \in \mathsf{Imp}(Q)$ composable, $S_P \parallel S_Q$ is good-enough w.r.t. $\mathcal{F}$.

To prove $\mathcal{F}$-compatibility of P' and Q, we will first construct an interface automaton $A_{R'}$ that is a legal environment for $(A_{P'}, A_Q)$. We will then show that for any $S'_P \in \mathsf{Imp}(P')$ and any $S_Q \in \mathsf{Imp}(Q)$, $S'_P \parallel S_Q$ is good-enough w.r.t. $\mathcal{F}$. By $(\Sigma^I_{P'} \setminus \Sigma^I_P) \cap \Sigma^O_Q = \emptyset$ and $\Sigma^O_{P'} \subseteq \Sigma^O_P$, we know that $\mathsf{Shared}(A_{P'}, A_Q) \subseteq \mathsf{Shared}(A_P, A_Q)$, i.e. $A_{P'}$ does not synchronize with A_Q on any new actions. We define $A_{R'} = \langle V_R, V_R^{init}, \Sigma^I_{R'}, \Sigma^O_{R'}, \Sigma^E_R, \Sigma^H_R, \mathcal{T}'_R \rangle$ such that:

- $\Sigma^I_{R'} = \Sigma^I_R \setminus (\Sigma^O_P \setminus \Sigma^O_{P'})$, $\Sigma^O_{R'} = \Sigma^O_R \cup (\Sigma^I_{P'} \setminus \Sigma^I_P)$,
- $\mathcal{T}'_R = \{(v, a, v') | (v, a, v') \in \mathcal{T}_R \wedge a \in \Sigma_{R'}\}$.

That is, $A_{R'}$ is equal to A_R except for the sets of inputs and outputs actions towards $P' \otimes Q$. These are modified to account for the fact that $\Sigma^I_P \subseteq \Sigma^I_{P'}$ and $\Sigma^O_P \supseteq \Sigma^O_{P'}$. Since none of the new outputs $(\Sigma^I_{P'} \setminus \Sigma^I_P)$ of $A_{R'}$ are used in $\mathcal{T}_R$, functionally $A_{R'}$ differs from A_R only in not accepting outputs of P that are removed in P', that is $(\Sigma^O_P \setminus \Sigma^O_{P'})$. To prove that $A_{R'}$ is a legal environment for $(A_{P'}, A_Q)$ we need to show that conditions in Definition 5 are met. Meaning that $A_{R'}$ is not empty and

1. $\Sigma^E_{R'} = \Sigma^E_{P' \otimes Q}$, $\Sigma^I_{R'} = \Sigma^O_{P' \otimes Q}$, $\Sigma^O_{R'} = \Sigma^I_{P' \otimes Q}$.
2. $A_{R'}$ is composable with $A_{P'} \otimes A_Q$, and $\mathsf{Illegal}(A_{P'} \otimes A_Q, A_R) = \emptyset$.
3. $\mathsf{Reach}((A_{P'} \otimes A_Q) \otimes A_{R'}) \cap (\mathsf{Illegal}(A_{P'}, A_Q) \times V_{R'}) = \emptyset$

By the definition of $A_{R'}$ and the properties of A_R, the first two conditions hold.

For the third condition, note that since $A_{P'}$ is a refinement of A_P, $A_{P'}$ will never reject an input accepted by A_P. In particular, $\mathsf{Reach}((A_{P'} \otimes A_Q) \otimes A_{R'})$ will not contain illegal states from $(\mathsf{Illegal}(A_{P'}, A_Q) \times V_{R'})$ if $\mathsf{Reach}((A_P \otimes A_Q) \otimes A_R)$ did not, since $\mathcal{T}'_R \subseteq \mathcal{T}_R$. Thus, the third condition also holds. Therefore, $A_{R'}$ is a legal environment for (P', Q).

Now we show that for any two implementations $S_{P'} \in \mathsf{Imp}(P')$ and $S_Q \in \mathsf{Imp}(Q)$ that are composable, $S_{P'} \parallel S_Q$ is good-enough w.r.t. $\mathcal{F}$.

As $P' \preceq P$, by Theorem 24 we have that $S_{P'} \in \mathsf{Imp}(P)$. Therefore, the $\mathcal{F}$-compatibility of P and Q implies that $S_{P'} \parallel S_Q$ is good-enough w.r.t $\mathcal{F}$. ◀

▶ **Lemma 32** (Monotonicity of Reward Function Composition)**.** *Consider reward interfaces $P = (A_P, \mathcal{F}_P)$ and $Q = (A_Q, \mathcal{F}_Q)$ with A_P and A_Q compatible, and let $P' = (A'_P, \mathcal{F}'_P)$ be a non-empty refinement of P, with $(\Sigma^I_{P'} \setminus \Sigma^I_P) \cap \Sigma^I_Q = \emptyset$. If comb is monotonically increasing, then $(A'_P \parallel A_Q, \mathcal{F}'_P \triangledown_{comb} \mathcal{F}_Q) \preceq (A_P \parallel A_Q, \mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q)$.*

Proof. Let $P = (A_P, \mathcal{F}_P)$ and $Q = (A_Q, \mathcal{F}_Q)$ be such that A_P and A_Q are compatible, and $P' = (A'_P, \mathcal{F}'_P)$ be non-empty, $P' \preceq P$ and $(\Sigma^I_{P'} \setminus \Sigma^I_P) \cap \Sigma^I_Q = \emptyset$. We show that if *comb* is monotonically increasing, then $(A'_P \parallel A_Q, \mathcal{F}'_P \triangledown_{comb} \mathcal{F}_Q) \preceq (A_P \parallel A_Q, \mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q)$.

For interface automata, $A'_P \parallel A_Q \preceq A_P \parallel A_Q$. We show $(\mathcal{F}'_P \triangledown_{comb} \mathcal{F}_Q) \preceq (\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q)$.

From $\mathcal{F}'_P \preceq \mathcal{F}_P$, there is a function $r : Vals(\mathcal{F}_P) \to Vals(\mathcal{F}'_P)$ that satisfies the conditions of Definition 21. To show $(\mathcal{F}'_P \triangledown_{comb} \mathcal{F}_Q) \preceq (\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q)$, we need to construct a function $r' : \mathbb{R}_{-\infty} \to \mathbb{R}_{-\infty}$ that satisfies the conditions of Definition 21 for these functions.

For all $v \in Vals(\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q)$, let $r'(v) = \inf\{comb(r(\mathcal{F}_P(\sigma|_{\Sigma^{Obs}_P})), \mathcal{F}_Q(\sigma|_{\Sigma^{Obs}_Q})) \mid \sigma \in (\Sigma^{Obs}_{P \otimes Q})^\infty$ and $comb(\mathcal{F}_P(\sigma|_{\Sigma^{Obs}_P}), \mathcal{F}_Q(\sigma|_{\Sigma^{Obs}_Q})) \geq v\}$.

For every value $v \in Vals(\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q)$ we have:

- Condition 1:
  Let $\sigma_{E,I} \in (\Sigma^{E,I}_{P' \otimes Q})^\infty$ be such that $\sigma_{E,I}|_{\Sigma^{E,I}_{P \otimes Q}} \in \mathit{Hopeful}(\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q, v, \Sigma^{E,I}_{P \otimes Q})$. This means that $\inf\{comb(\mathcal{F}_P(\sigma|_{\Sigma^{Obs}_P}), \mathcal{F}_Q(\sigma|_{\Sigma^{Obs}_Q})) \mid S_P \in \mathsf{Imp}(P), S_Q \in \mathsf{Imp}(Q)$ composable, $\sigma \in \mathsf{Traces}(S_P \parallel S_Q, \sigma_{E,I})\} \geq v$.
  We know that $\mathsf{Imp}(P') \subseteq \mathsf{Imp}(P)$, so in particular $\mathsf{Imp}(A'_P \parallel A_Q, \mathcal{F}'_P \triangledown_{comb} \mathcal{F}_Q) \subseteq \mathsf{Imp}(A_P \parallel A_Q, \mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q)$. That means that for $\mathcal{F}'_P \triangledown_{comb} \mathcal{F}_Q$ we will consider a subset of the implementations, and therefore not more traces when applying inf, as for $\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q$. This, together with the above inequality implies $\mathcal{F}'_P \triangledown_{comb} \mathcal{F}_Q(\sigma_{E,I}) \geq r'(v)$. Then it holds that any $\sigma \in \mathsf{Traces}(A_P \parallel A_Q, \sigma_{E,I})$, for which $\mathcal{F}'_P \triangledown_{comb} \mathcal{F}_Q(\sigma)$ is defined, is a witness for $\sigma_{E,I} \in \mathit{Hopeful}(\mathcal{F}'_P \triangledown_{comb} \mathcal{F}_Q, r'(v), \Sigma^{E,I}_{P' \otimes Q})$.
- Condition 2:
  Let $\sigma' \in (\Sigma^{Obs}_{P' \otimes Q})^\infty$ be such that $(\mathcal{F}'_P \triangledown_{comb} \mathcal{F}_Q)(\sigma') \geq r'(v)$. This means that
  $\inf\{comb(\mathcal{F}'_P(\sigma|_{\Sigma^{Obs}_{P'}}), \mathcal{F}_Q(\sigma|_{\Sigma^{Obs}_Q})) \mid$
  $S'_P \in \mathsf{Imp}(P'), S_Q \in \mathsf{Imp}(Q),$ composable, $\sigma \in \mathsf{Traces}(S'_P \parallel S_Q, \sigma'|_{\Sigma^{E,I}_{P' \otimes Q}})\} \geq r'(v)$.
  We have to show that $\inf\{comb(\mathcal{F}_P(\sigma|_{\Sigma^{Obs}_P}), \mathcal{F}_Q(\sigma|_{\Sigma^{Obs}_Q})) \mid$
  $S_P \in \mathsf{Imp}(P), S_Q \in \mathsf{Imp}(Q),$ composable, $\sigma \in \mathsf{Traces}(S_P \parallel S_Q, \sigma'|_{\Sigma^{E,I}_{P \otimes Q}})\} \geq v$.
  Since $\mathcal{F}'_P \preceq \mathcal{F}_P$ we have for the function r that $\mathcal{F}'_P(\sigma) \geq r(\mathcal{F}_P(\sigma|_{\Sigma^{Obs}_P}))$. Since this function r is used in r', then $comb(r(\mathcal{F}_P(\sigma|_{\Sigma^{Obs}_P})), \mathcal{F}_Q(\sigma|_{\Sigma^{Obs}_Q})) \geq r'(v)$, meaning that $comb(\mathcal{F}_P(\sigma|_{\Sigma^{Obs}_P}), \mathcal{F}_Q(\sigma|_{\Sigma^{Obs}_Q})) \geq v$ by definition of r' and monotonicity of $comb$. Thus,
  $\inf\{comb(\mathcal{F}_P(\sigma|_{\Sigma^{Obs}_P}), \mathcal{F}_Q(\sigma|_{\Sigma^{Obs}_Q})) \mid$
  $S_P \in \mathsf{Imp}(P), S_Q \in \mathsf{Imp}(Q),$ composable, $\sigma \in \mathsf{Traces}(S_P \parallel S_Q, \sigma'|_{\Sigma^{E,I}_{P \otimes Q}})\} \geq v$.

Thus, both conditions in Definition 21 hold for r'. Hence, $(\mathcal{F}'_P \triangledown_{comb} \mathcal{F}_Q) \preceq (\mathcal{F}_P \triangledown_{comb} \mathcal{F}_Q)$. ◀