A Modal Interface Theory for Component-based Design

Jean-Baptiste Raclet  
INRIA Grenoble - Rhônes-Alpes, France  
 jean-baptiste.raclet@inrialpes.fr

Eric Badouel  
INRIA/IRISA Rennes, France  
eric.badouel@irisa.fr

Albert Benveniste  
INRIA/IRISA Rennes, France  
albert.benveniste@irisa.fr

Benoît Caillaud  
INRIA/IRISA Rennes, France  
benoit.caillaud@irisa.fr

Axel Legay  
INRIA/IRISA Rennes, France  
axel.legay@irisa.fr

Roberto Passerone  
University of Trento, Italy  
roberto.passerone@unitn.it

Abstract. This paper presents the modal interface theory, a unification of interface automata and modal specifications, two radically dissimilar models for interface theories. Interface automata is a game-based model, which allows to make assumptions on the environment and propose an optimistic view for composition: two components can be composed if there is an environment where they can work together. Modal specification is a language theoretic account of a fragment of the modal mu-calculus logic with a rich composition algebra meeting some methodological requirements but which does not allow to distinguish between the environment and the component. The present paper contributes a thorougher unification of the two theories by correcting a first attempt in this direction by Larsen et al., drawing a complete picture of the modal interface algebra, and pushing even further the comparison between interface automata, modal automata and modal interfaces.

The work reported here is based on the earlier work in [46] and [45].

Keywords: Component-based System, Compositional Reasoning, Interface Theory, Interface Automata, Modal Specifications.

*This work was funded in part by the European IP-SPEEDS project number 033471 and the European STREP-COMBEST project number 215543.
1. Introduction

Nowadays, systems are tremendously big and complex, resulting from the assembling of several components. These many components are in general designed by teams, working independently but with a common agreement on what the interface of each component should be. As a consequence, mathematical foundations that allow to reason at the abstract level of interfaces is a very active research area. According to our understanding of industrial needs (see [11] for a discussion), an interface theory is at least subject to the following requirements:

1. **Satisfaction and satisfiability are decidable.** Interfaces should be seen as specifications whose models are its possible implementations. It should thus be decidable whether an interface admits an implementation and whether a given component implements a given interface.

2. **Refinement entails substituability.** Refinement allows one to replace, in any context, an interface by a more detailed version of it. Refinement should entail substituability of interface implementations, meaning that every implementation satisfying a refinement also satisfies the larger interface. For the sake of controlling design complexity, it is desirable to be able to decide whether there exists an interface that refines two different interfaces. This is called *shared refinement* [25]. In many situations, we are looking for the greatest lower bound, i.e., the shared refinement that could be refined by any other shared refinement.

3. **Interfaces are closed under conjunction.** It is the current practice that early requirements capture relies on Doors Databases, or even Excel files containing possibly many textual requirements. Under the current practice, little formal support exists to handle them. Moving ahead can be envisioned by formalizing the notation used for individual requirements. This can be, e.g., achieved by relying on so-called semi-formal languages [19], whose sentences are translatable into predefined behavioral patterns according to several viewpoints. Alternatively, graphical scenario languages could be considered [23, 33]. Composing viewpoints within a given subsystem calls for the support of the concept of *conjunction* of interfaces in order to combine requirements and check their satisfiability.

4. **Composition supports independent design.** The interface theory should also provide a combination operator on interfaces, reflecting the standard composition of implementations by, e.g. parallel product. This operation must be associative and commutative to guarantee independence in the development. Depending on the model, a notion of compatibility for composition may also be considered, i.e., there can be cases where two systems cannot be composed.

5. **Interface quotient supports incremental design and component reuse.** A quotienting operation, dual to composition is crucial to perform incremental design. Consider a desired global specification and the specification of a preexisting component; the quotient specification describes the part of the global specification that remains to be implemented.

6. **A verification procedure.** In addition to the fact that an interface already represents a set of properties, one should be able to verify if an interface satisfies a set of requirements written in some specification language.
7. Encompassing interfaces with dissimilar alphabets. Complex systems are built by combining subsystems possessing dissimilar alphabets for referencing ports and variables. It is thus important to properly handle those different alphabets when combining interfaces.

Building good interface theories has been the subject of intensive studies (see e.g., [32, 4, 17, 26, 29, 6, 5]). In this paper we will concentrate on two models: (1) interface automata [4] and (2) modal specifications [34]. Interface automata is a game-based variation of input/output automata which deals with open systems, their refinement and composition, and put the emphasis on interface compatibility. Modal specifications is a language theoretic account of a fragment of the modal mu-calculus logic [28] which admits a richer composition algebra with product, conjunction and residuation operators.

Modal specifications correspond to deterministic modal automata [34], i.e., automata whose transitions are typed with may and must modalities. A modal specification thus represents a set of models; informally, a must transition is available in every component that implements the modal specification, while a may transition needs not be. The components that implement modal specifications are prefix-closed languages, or equivalently deterministic automata/transition systems.

Satisfiability of modal specifications is decidable. Refinement between modal specifications coincides with models inclusion. Conjunction is effectively computed via a product-like construction. It can be shown that the conjunction of two modal specifications correspond to their greatest lower bound. Combination of modal specifications, handling synchronization products à la Arnold and Nivat [10], and the dual quotient combinators can be efficiently handled in this setting [43, 44].

In interface automata [4], an interface is represented by an input/output automaton [38], i.e., an automaton whose transitions are labeled with input or output actions. The semantics of such an automaton is given by a two-player game: an Input player represents the environment, and an Output player represents the component itself. Interface automata do not encompass any notion of model, because one cannot distinguish between interfaces and implementations.

Refinement between interface automata corresponds to the alternating refinement relation between games [8], i.e., an interface refines another one if its environment is more permissive whereas its component is more restrictive. Shared refinement is defined in an ad-hoc manner [25] for a particular class of interfaces [20]. Contrary to most interfaces theories, the game-based interpretation offers an optimistic treatment of composition: two interfaces can be composed if there exists at least one environment (i.e., one strategy for the Input player) in which they can interact together in a safe way (i.e., whatever the strategy of the Output player is). This is referred as compatibility of interfaces. A quotient, which is the adjoint of the game-based composition, has been proposed in [16] for the deterministic case.

It is worth mentioning that, in existing works on interface automata and modal specification, there is nothing about dissimilar alphabets. This is somehow surprising as it seems to be a quite natural question when performing operations that involve several components, e.g., conjunction, composition, and quotient. As we shall see in this paper, an explicit mechanism to handle dissimilar alphabets is not needed when considering interface automata. Indeed, this model does not propose conjunction. For the case of composition/quotient, we shall see that the notion is implicitly encompassed in the definition of compatibility. Conjunction and quotient operators [34, 43, 44] that have been proposed for modal specification do not take dissimilar alphabet into account. One thus need to extend those operators to this more general
setting. This is one of the subjects of this paper.

As a conclusion, both models have advantages and disadvantages:

- Interface automata is a model that allows to make assumptions on the environment, which is mainly useful to derive a rich notion for composition. In addition, the notion of dissimilar alphabets is not needed. Unfortunately, the model is incomplete as conjunction and shared refinement are not defined.

- Modal specification is a rich language algebra model on which most of requirements for a good interface theory can be considered. Unfortunately, may and must modalities are not sufficient to derive a rich notion for composition including compatibility. Moreover, the notion of dissimilar alphabets is missing.

It is thus worth considering unification of the frameworks of interface automata and modal specifications. A first attempt was made by Larsen et al. [35, 40] who considered modal interfaces that are modal specifications whose actions are also typed in input or output attributes. A modal interface can just be viewed as a modal specification except for the composition operation and the fact we have to deal with the added modalities which may be complicated. The refinement for modal interface is the refinement for modal specification, while the composition for is the one from interface automata. In [35, 40], Larsen et al. have shown that the refinement for modal specifications is compatible with the composition operation for interface automata. The main problem with their results is that the composition operator is incorrect. Indeed, contrary to what is claimed by the authors, the composition operator in [35, 40] is not monotone with respect to satisfaction. This fails to ensure that two compatible interfaces may be implemented separately. Moreover, requirements such as dissimilar alphabets, conjunction, and component reuse are not considered.

The present paper adds a new stone to the cathedral of results on interface theories by (1) proposing a new theory for dissimilar alphabets, (2) correction for the modal interface composition operator presented in [35, 40], (3) pushing further the comparison between interface automata, modal automata and modal specifications and modal interfaces, and (4) reasoning on architectural design for modal interfaces.

The rest of the paper is organized as follows. In Sections 2 and 3 we recap the theory for modal specifications and interface automata, respectively. In Section 4, we present the complete theory for modal interfaces and correct the error in [35, 40]. Section 5, is dedicated to architectural design. Finally, in Section 6, we draw our conclusion and discuss future extensions for the model of modal interfaces.

2. Modal specifications

This section starts with an overview of existing results developed for modal specifications in [34, 43, 44] (Sections 2.1, 2.2 and 2.3). We also propose a new methodology to encompass dissimilar alphabets (Section 2.4).
2.1. The Framework

Following our previous work [43, 44, 46], we will define modal specifications in term of languages, knowing that they can also be represented with deterministic automata whose transitions are typed with \textit{may} and \textit{must} modalities [34]. We start with the following definition.

\textbf{Definition 2.1. (Modal specification)}

A modal specification is a tuple $\mathcal{S} = (A, \textit{must}, \textit{may})$, where $A$ is a finite alphabet and:

$$\textit{must}, \textit{may} : A^* \rightarrow 2^A$$

are partial functions satisfying the following consistency condition:

$$\textit{must}(u) \subseteq \textit{may}(u). \quad (1)$$

If we have $a \in \textit{may}(u)$, then this means that $a$ is allowed after the trace $u$ whereas $a \in \textit{must}(u)$ indicates that $a$ is required after $u$. By negation, $a \notin \textit{may}(u)$ means that $a$ is disallowed after $u$. The latter is often written $a \in \textit{mustnot}(u)$. The condition (1) naturally imposes that every required action is also allowed. We shall sometimes write $\textit{may}_i$, and $\textit{must}_i$ to refer to the entities involved in the definition of $\mathcal{S}_i$.

Modal specifications that generate regular languages can be represented by \textit{deterministic modal automata}, i.e., deterministic finite-word automata with two types of transitions: solid transitions if the action is required in the source state and dashed transition if is allowed but not required. The concept is illustrated with the following example.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{modal_specification.eps}
\caption{The modal specification \textit{Fun} accepts the automaton}
\end{figure}

\textbf{Example 2.1.} Consider a producer whose alphabet of actions includes $\textit{msg}$ for when the producer send a message as well as two kinds of acknowledgment for transmission: $\textit{ack}$ in case of success and $\textit{nack}$ in case of failure. Assume also the existence of an action $\textit{extra}$ which occurs when extra resources are requested to dispatch a message.

A functional specification \textit{Fun} for the producer is given in Figure 1(a). It specifies that a $\textit{msg}$ may be sent again. Moreover every $\textit{msg}$ may be acknowledged. Additionally, the producer may request extra resources at any moment.

When composing specifications, discrepancies between the modal informations carried out by the specifications may appear. We then consider \textit{pseudo-modal specifications} (also called \textit{mixed transition}
systems in [24]), denoted $pS$; they are triples satisfying Definition 2.1 with the exception of (1). For $pS$ a pseudo-modal specification, a word $u \in A^*$ is called **consistently specified** in $pS$ if it satisfies (1) and **inconsistent** otherwise; modal specifications correspond exactly to the subclass of consistent pseudo-modal specifications, that is pseudo-specifications such that every $u \in A^*$ is consistently specified.

For $pS = (A, \text{must}, \text{may})$ a pseudo-modal specification, the **support** of $pS$ is the least prefix-closed language $L_{pS}$ such that:

(i) $\epsilon \in L_{pS}$, where $\epsilon$ denotes the empty word; and

(ii) $u \in L_{pS}$ and $a \in \text{may}(u)$ imply $u.a \in L_{pS}$.

### 2.2. Implementation, refinement and consistency

In this section, we study the concepts of **implementation**, **refinement** and **consistency**. We start with implementation, which is also called **model**.

**Definition 2.2. (implementation)**

A prefix-closed language $I \subseteq A^*$ is an implementation of a pseudo-modal specification $pS = (A, \text{must}, \text{may})$, denoted by $I \vdash pS$, if

$$\forall u \in I \Rightarrow \text{must}(u) \subseteq I_u \subseteq \text{may}(u)$$

where $I_u$ is the set of actions $a \in A$ such that $u.a \in I$.

**Example 2.2.** A model for the specification given in Figure 1(a) is presented in Figure 1(b). It indicates that every message will be acknowledged either positively or negatively. Moreover, an extra resource is requested if the message has to be re-emitted.

**Lemma 2.1.** Let $I \subseteq A^*$ be a prefix-closed language and $pS$ a pseudo-modal specification over $A$. If $I \vdash pS$, then $I \subseteq L_{pS}$ holds and every word of $I$ is consistently specified in $pS$.

The concept of thorough refinement follows immediately from definition 2.2 by comparing, through set inclusion, the sets of implementations associated to two modal specifications.

**Definition 2.3.** There exists a thorough refinement between Specification $S_1$ and Specification $S_2$ if and only if any model of $S_1$ is also a model of $S_2$.

Thorough refinement has been extensively studied in [36] and compared to the more syntactic notion of modal refinement that is recalled hereafter.

**Definition 2.4. (modal refinement)**

Let $pS_1$ and $pS_2$ be two pseudo-modal specifications. The specification $pS_1$ refines $pS_2$, written $pS_1 \leq pS_2$, if and only if, for all $u \in L_1$, $\text{may}_1(u) \subseteq \text{may}_2(u)$ and $\text{must}_1(u) \supseteq \text{must}_2(u)$. 
It is easy to see that refinement is a preorder relation that implies inclusion of supports: \( \mathcal{L}_{\mathcal{S}_1} \subseteq \mathcal{L}_{\mathcal{S}_2} \). Any two modal specifications \( \mathcal{S}_1 \) and \( \mathcal{S}_2 \) such that \( \mathcal{S}_1 \subseteq \mathcal{S}_2 \subseteq \mathcal{S}_1 \) have equal supports \( \mathcal{L} = \mathcal{L}_{\mathcal{S}_1} = \mathcal{L}_{\mathcal{S}_2} \) and for all \( u \in \mathcal{L} \), \( \text{may}_1(u) = \text{may}_2(u) \) and \( \text{must}_1(u) = \text{must}_2(u) \). Said differently, equivalent modal specifications differ only outside of their support. A unique representant \( \mathcal{S} = (A, \text{must}, \text{may}) \) of equivalence classes of modal specifications is defined by assuming that for all \( u \notin \mathcal{L}_\mathcal{S} \), \( \text{must}(u) = \emptyset \) and \( \text{may}(u) = A \). In the sequel, only modal specifications satisfying this property are considered. Under this assumption, modal refinement is a partial order relation on modal specifications.

Moreover, in [43, 44, 12], it is shown that modal refinement for modal specifications is sound and complete, i.e., it is equivalent to thorough refinement\(^1\). For nondeterministic modal specifications, the thorough refinement is PSPACE-hard [9] (and also EXPTIME). As the modal refinement is P-complete, a faster decision procedure exists in the deterministic case.

The following result relates implementations to consistency, for a pseudo-modal specification:

**Theorem 2.1. (consistency [43, 44])**

Let \( \mathcal{P}_\mathcal{S} \) be a pseudo-modal specification. Either \( \mathcal{P}_\mathcal{S} \) possesses no implementation, or there exists a largest (for refinement order) modal specification \( \rho(\mathcal{P}_\mathcal{S}) \) having the same alphabet of actions and such that \( \rho(\mathcal{P}_\mathcal{S}) \subseteq \mathcal{P}_\mathcal{S} \). In addition, \( \rho(\mathcal{P}_\mathcal{S}) \) possesses the same set of implementations as \( \mathcal{P}_\mathcal{S} \).

The modal specification \( \rho(\mathcal{P}_\mathcal{S}) \) is called the pruning of \( \mathcal{P}_\mathcal{S} \). It is obtained from \( \mathcal{P}_\mathcal{S} \) through the following steps:

1. Start from \( R_0 \), a copy of \( \mathcal{P}_\mathcal{S} \);

2. Let \( U_0 \) be the set of words inconsistently specified in \( R_0 \), meaning that \( u \in U_0 \) does not satisfy condition (1). For each \( u \in U_0 \), set \( \text{may}_{R_0}(u) = A \) and \( \text{must}_{R_0}(u) = \emptyset \). Then, for each word \( v \in A^* \) such that \( v.a = u \) for some \( u \in U_0 \) and \( a \in A \), remove \( a \) from \( \text{may}_{R_0}(v) \). Performing these two operations yields a pseudo-modal specification \( R_1 \) such that \( U_0 \) is consistently specified in \( R_1 \). Since we only have removed inconsistently specified words from \( \mathcal{L}_{R_0} \), by Lemma 2.1, \( R_1 \) and \( R_0 \) possess identical sets of implementations.

3. Observe that, if \( a \in \text{must}_{R_1}(v) \), then \( v \) becomes inconsistently specified in \( R_1 \). So we repeat the above step on \( R_1 \), by considering \( U_1 \), the set of words \( u \) inconsistently specified in \( R_1 \). Let \( \Delta_1 \subseteq U_0 \times U_1 \) be the relation consisting of the pairs \((u, v)\) such that \( v.a = u \) for some \( a \) and \( v \) is inconsistently specified in \( R_1 \). Note that \( v \) is a strict prefix of \( u \).

4. Repeating this, we get a sequence of triples \((R_k, U_k, \Delta_k)_{k \geq 0} \) such that 1) \( \bigcup_{m \leq k} U_m \) is consistently specified in \( R_{k+1} \), and 2) \( \text{may}_{R_{k+1}}(v) \subseteq \text{may}_{R_k}(v) \) for each \( v \), with strict inclusion whenever \( v.a = u \) for some \( u \in U_k \), and 3) \( \Delta_{k+1} \subseteq U_k \times U_{k+1} \) is the relation consisting of the pairs \((u, v)\) such that \( v.a = u \) for some \( a \) and \( v \) is inconsistently specified in \( R_{k+1} \) — again, \( v \) is a strict prefix of \( u \).

5. Call \textit{chain} a sequence \( u_0, u_1, \ldots \) of words such that \((u_k, u_{k+1}) \in \Delta_{k+1} \) for every \( k \geq 0 \). Since \( u_{k+1} \) is a strict prefix of \( u_k \), every chain is of length at most \(|u_0| \). Thus, every inconsistently specified word of \( \mathcal{P}_\mathcal{S} \) is removed after finitely many steps of the above algorithm. This proves that the procedure eventually converges. The limit \( \rho(\mathcal{P}_\mathcal{S}) \) is consistent and is given by:

---

\(^1\)Completeness of modal refinement does not hold for nondeterministic modal automata [36]. It holds in our case since we work with specifications (for which determinism is hardwired).
\[
\text{may}(u) = \bigcap_k \text{may}_{R_k}(u)
\]
\[
\text{must}(u) = \begin{cases} 
\text{must}_{\mathcal{S}}(u) & \text{if } \text{must}_{\mathcal{S}}(u) \subseteq \text{may}(u) \\
\emptyset & \text{otherwise}
\end{cases}
\]

The above procedure terminates in finitely many steps if the support of the pseudo-modal specification is rational which is, in particular, the case of pseudo-modal specifications originated from a deterministic pseudo-modal automaton. This procedure also entails a sufficient condition for the satisfiability problem: a pseudo-modal specification admits a model if and only if there is no word \( u \in \mathcal{L}_\mathcal{S} \) such that \( u \) is inconsistent and for all prefix \( v \) of \( u \) if \( u = v.a.v' \) then \( a \in \text{must}(v) \). Hence this problem is NLOGSPACE-complete; it is PSPACE-hard for nondeterministic pseudo-modal specifications [9].

### 2.3. Operations on modal specifications

**Greatest Lower Bound:** The set of all pseudo-modal specifications equipped with modal refinement \( \leq \) is a lattice. We denote by \( \mathcal{S}_1 \& \mathcal{S}_2 \) the Greatest Lower Bound (GLB) of \( \mathcal{S}_1 \) and \( \mathcal{S}_2 \) defined over the same alphabet. The GLB \( \mathcal{S}_1 \& \mathcal{S}_2 \) is defined by:

\[
\text{may}(u) = \text{may}_1(u) \cap \text{may}_2(u)
\]
\[
\text{must}(u) = \text{must}_1(u) \cup \text{must}_2(u)
\]

Observe that, even if \( \mathcal{S}_1 \) and \( \mathcal{S}_2 \) satisfy (1), it is not guaranteed that \( \mathcal{S}_1 \& \mathcal{S}_2 \) does too. Hence, by using theorem 2.1, for \( \mathcal{S}_1 \) and \( \mathcal{S}_2 \) two modal specifications, we define \( \mathcal{S}_1 \land \mathcal{S}_2 \) as being the (uniquely defined) modal specification

\[
\mathcal{S}_1 \land \mathcal{S}_2 = \rho(\mathcal{S}_1 \& \mathcal{S}_2).
\]

GLB satisfies the following key property, which relates it to logic formulas:

**Theorem 2.2. (conjunctive interfaces [43, 44])**

Let \( \mathcal{I} \) be a prefix-closed language and \( \mathcal{S}_1 \) and \( \mathcal{S}_2 \) be modal specifications:

\[
\mathcal{I} \models \mathcal{S}_1 \land \mathcal{S}_2 \iff \mathcal{I} \models \mathcal{S}_1 \text{ and } \mathcal{I} \models \mathcal{S}_2
\]

The following holds regarding supports: \( \mathcal{L}_{\mathcal{S}_1 \land \mathcal{S}_2} \subseteq \mathcal{L}_{\mathcal{S}_1} \cap \mathcal{L}_{\mathcal{S}_2} \), with equality if and only if no pruning is needed, i.e., \( \mathcal{S}_1 \land \mathcal{S}_2 = \mathcal{S}_1 \& \mathcal{S}_2 \).

**Composition:** Let \( \mathcal{S}_1 \) and \( \mathcal{S}_2 \) be two modal specifications over the same alphabet. Their composition \( \mathcal{S}_1 \otimes \mathcal{S}_2 \) is defined by:

\[
\text{may}(u) = \text{may}_1(u) \cap \text{may}_2(u)
\]
\[
\text{must}(u) = \text{must}_1(u) \cap \text{must}_2(u)
\]

The following theorem shows that composition ensures substitutability.
Theorem 2.3. (substitutability in composition [43, 44])
Let $\mathcal{I}_1, \mathcal{I}_2$ be two prefix-closed languages and $S_1, S_2, S'_1$ and $S'_2$ be modal specifications:

1. If $S'_1 \leq S_1$ and $S'_2 \leq S_2$, then $S'_1 \otimes S'_2 \leq S_1 \otimes S_2$.
2. If $\mathcal{I}_1 \models S_1$ and $\mathcal{I}_2 \models S_2$, then $\mathcal{I}_1 \times \mathcal{I}_2 \models S_1 \otimes S_2$, where $\mathcal{I}_1 \times \mathcal{I}_2 = \mathcal{I}_1 \cap \mathcal{I}_2$.
3. The following holds regarding supports: $\mathcal{L}_{S_1 \otimes S_2} = \mathcal{L}_{S_1} \cap \mathcal{L}_{S_2}$.

Residuation: We now discuss the residuation operation which was introduced in [43, 44]. We will show that this operation is the adjoint of composition. For $S_1$ and $S_2$ two modal specifications, we first define their pseudo-quotient $S_1 // S_2$ according to the following disjunctive and exhaustive cases:

- $a \in \text{may}(u) \cap \text{must}(u)$ if $a \in \text{must}_1(u)$ and $a \in \text{must}_2(u)$
- $a \in \text{must}(u) \setminus \text{may}(u)$ if $a \in \text{must}_1(u)$ and $a \notin \text{must}_2(u)$
- $a \in \text{may}(u) \setminus \text{must}(u)$ if $a \in \text{may}_1(u)$ and $a \notin \text{must}_1(u)$
- $a \notin \text{may}(u) \cup \text{must}(u)$ if $a \notin \text{may}_1(u)$ and $a \notin \text{may}_2(u)$

Observe that, due to the second case, $S_1 // S_2$ is not consistent. Having defined $S_1 // S_2$, using the pruning operation of theorem 2.1, we can now set

$$S_1 / S_2 = \rho(S_1 // S_2) \quad (5)$$

Any prefix-closed language $\mathcal{I} \subseteq A^*$ can be viewed as a modal specification whose set of must coincides with may: $\forall u \in A^*, \text{must}(u) = \text{may}(u) = \mathcal{I}_u$. Using this embedding, the quotient of two prefix-closed languages can be defined. Observe that, because of the fourth rule, the quotient of two languages is a modal specification that is not a language.

We now show that the quotient operation is indeed the adjoint of the composition operation:

Theorem 2.4. (residuation [43, 44])
Let $\mathcal{I}_1, \mathcal{I}_2$ be prefix-closed languages and $S_1, S_2, S$ be modal specifications:

1. $S_1 \otimes S_2 \leq S$ if and only if $S_2 \leq S / S_1$
2. $\forall \mathcal{I}_1 : [\mathcal{I}_1 \models S_1 \Rightarrow \mathcal{I}_1 \times \mathcal{I}_2 \models S] \iff \mathcal{I}_2 \models S / S_1$. 


Example 2.3. Quotient and conjunction are illustrated in Figure 2. Suppose one aims at realizing a system whose behavior is given by the left-hand side specification: every message must be acknowledged positively. For this purpose, a preexisting component conformed to the middle-hand side specifications is available in the context; it implements the specification $Fun$ of Figure 1(a) with the additional assumption that the communication channel never distributes negative acknowledgement. Then, the product of the context with any implementation of the right-hand side specification is guaranteed to be an implementation of the desired behavior.

2.4. Dissimilar Alphabets

Complex systems are built by composing and combining many subsystems or components. Clearly, those objects should possess their own local alphabet of ports and variables. Dealing with those local aspects when developing the fundamental services seems like a trivial notice but has deep technical consequences. As we shall see in this section, modalities appear as an elegant solution to address alphabet equalization with appropriate flexibility.

Let us first recall how alphabet equalization is performed for the shuffle product of languages. For $w$ a word over some alphabet $A$, and $B \subseteq A$, let $pr_B(w)$ denote the word over $B$ obtained by erasing, from $w$, all symbols not belonging to $B$. For $L$ a language over $A$ and $B \subseteq A \subseteq C$, the restriction of $L$ to $B$ is the language $L_{\uparrow B} = \{ u \in B^* | u = pr_B(w), w \in L \}$ and the extension of $L$ to $C$ is the language $L_{\uparrow C} = \{ u \in C^* | pr_A(u) \in L \}$. The shuffle product $L_1 \times L_2$ of the two languages $L_1 \subseteq A_1^*$ and $L_2 \subseteq A_2^*$ is then defined as

$$L_1 \times L_2 = (L_1)_{\uparrow C} \cap (L_2)_{\uparrow C},$$

where $C = A_1 \cup A_2$.

The shuffle product uses inverse projection to equalize alphabets. The same holds for automata over dissimilar alphabets and their synchronous product.
Using modalities allows for a neutral procedure for equalizing alphabets. The principle is as follows. Observe that, by (4),

\[ a \in \text{must}_1(s) \quad \text{and} \quad a \in \text{whatever}_2(s) \]

\[ \Downarrow \]

\[ a \in \text{whatever}(s) \]

holds if the two interfaces are combined using parallel composition (here, whatever denotes an arbitrary modality). Similarly, by (2),

\[ a \in \text{may}_1(s) \quad \text{and} \quad a \in \text{whatever}_2(s) \]

\[ \Downarrow \]

\[ a \in \text{whatever}(s) \]

holds if the two interfaces are combined using conjunction. The observation above reveals our solution: alphabet extension is performed by setting the specific modalities for extended traces:

- \textit{may} for the case of the conjunction \( \land \);
- \textit{must} for the case of the parallel composition \( \otimes \).

These two types of alphabet extensions are called \textit{weak} and \textit{strong}. This is a key contribution of our work as it will provide us with a very elegant way of dealing with dissimilar alphabets.

**Definition 2.5. (weak and strong extensions)**

Let \( pS = (A, \text{must}_{pS}, \text{may}_{pS}) \) be a pseudo-modal specification and let \( C \supseteq A \).

1. The \textit{weak extension} of \( pS \) to \( C \) is the pseudo-modal specification \( pS_{\uparrow C} = (C, \text{must}, \text{may}) \) such that \( \forall v \in C^* : \)

\[
\begin{align*}
\text{must}(v) & = \text{must}_{pS}(\text{pr}_A(v)) \\
\text{may}(v) & = \text{may}_{pS}(\text{pr}_A(v)) \cup (C - A)
\end{align*}
\]

2. The \textit{strong extension} of \( pS \) to \( C \) is the pseudo-modal specification \( pS_{\downarrow C} = (C, \text{must}, \text{may}) \) such that \( \forall v \in C^* : \)

\[
\begin{align*}
\text{must}(v) & = \text{must}_{pS}(\text{pr}_A(v)) \cup (C - A) \\
\text{may}(v) & = \text{may}_{pS}(\text{pr}_A(v)) \cup (C - A)
\end{align*}
\]

Regarding supports, the following equalities hold: \( L(s_{\uparrow C}) = L(s_{\downarrow C}) = (L_S)_{\uparrow C} \). We are now ready to extend the operations of Sections 2.2 and 2.3 to the case of dissimilar alphabets.

**Definition 2.6.** Let \( pS, pS_i \) and \( S_i \) be pseudo-modal or modal specifications over alphabets \( A, A_i \) for \( i = 1, 2 \), respectively. The relations and operations of section 2.2 are redefined as follows:

[weak implementation; \( C \supseteq A \)]

\[ I \subseteq C^* \models_w pS \iff I \models pS_{\uparrow C} \]

[strong implementation; \( C \supseteq A \)]

\[ I \subseteq C^* \models_s pS \iff I \models pS_{\downarrow C} \]
[weak refinement; $A_2 \supseteq A_1$]

\[ pS_2 \leq_w pS_1 \iff pS_2 \leq pS_{1 \uparrow A_2} \]

[strong refinement; $A_2 \supseteq A_1$]

\[ pS_2 \leq_s pS_1 \iff pS_2 \leq pS_{1 \uparrow A_2} \]

[operators; $C = A_1 \cup A_2$]

\[
S_1 \wedge S_2 = S_{1 \uparrow C} \wedge S_{2 \uparrow C} \\
S_1 \odot S_2 = S_{1 \downarrow C} \odot S_{2 \downarrow C} \\
S_1 \oslash S_2 = S_{1 \uparrow C} \oslash S_{2 \downarrow C}
\]

Note the careful use of weak and strong extensions in the different operations. The results of sections 2.2 and 2.3 are slightly weakened as indicated next.

**Theorem 2.5.** Let $S$, $S_i$ and $S'_i$ be modal specifications respectively defined over $A, A_i$ and $A'_i$, for $i = 1, 2$.

1. Weak and strong implementation / refinement relations are related as follows:

\[ \models_s \subseteq \models_w \text{ and } \leq_s \subseteq \leq_w \]

2. Weak and strong modal refinement are both sound and complete w.r.t. weak and strong thorough refinement, respectively:

\[
S_2 \leq_w S_1 \iff \{ I \mid I \models_w S_2 \} \subseteq \{ I \mid I \models_w S_1 \} \\
S_2 \leq_s S_1 \iff \{ I \mid I \models_s S_2 \} \subseteq \{ I \mid I \models_s S_1 \}
\]

3. The following holds regarding conjunction:

\[ \models_w S_1 \wedge S_2 \iff \models_w S_1 \text{ and } \models_w S_2 \]

4. Theorem 2.3 still holds when alphabets are different, provided that *strong* refinement and implementation are used — it is actually false if weak refinement or implementation are used:

- If $S'_1 \leq_s S_1$ and $S'_2 \leq_s S_2$, then $S'_1 \odot S'_2 \leq_s S_1 \odot S_2$;
- If $I_1 \models_s S_1$ and $I_2 \models_s S_2$, then $I_1 \times I_2 \models_s S_1 \odot S_2$;
- $S'_1 \leq_w S_1$ and $S'_2 \leq_w S_2$ do not imply that $S'_1 \odot S'_2 \leq_w S_1 \odot S_2$ in general;
- $I_1 \models_w S_1$ and $I_2 \models_w S_2$ do not imply that $I_1 \times I_2 \models_w S_1 \odot S_2$ in general.
5. Relations between the quotient and the composition operators are preserved provided additional assumptions on alphabets:

\[
\begin{align*}
S_2 \leq_s S/S_1 & \quad \Rightarrow S_1 \otimes S_2 \leq_s S \\
A_1 \subseteq A & \\
S_1 \otimes S_2 \leq_s S & \quad \Rightarrow S_2 \leq_s S/S_1 \\
A_2 \supseteq A \cup A_1 & \\
\mathcal{I}_1 \models S_1 \text{ and } \mathcal{I}_2 \models S/S_1 & \quad \Rightarrow \mathcal{I}_1 \times \mathcal{I}_2 \models S \\
A_1 \subseteq A & \\
\forall \mathcal{I}_1 : \mathcal{I}_1 \models S_1 & \Rightarrow \mathcal{I}_1 \times \mathcal{I}_2 \models S \\
\text{and } A_{\mathcal{I}_2} \supseteq A \cup A_1 & \quad \Rightarrow \mathcal{I}_2 \models S/S_1
\end{align*}
\]

Observe that the last sub-statement of statement 5 refines Theorem 2.4.

Example 2.4. Consider now a second specification \(\text{Rel}\) for a producer in Figure 3(a) dealing with reliability: messages are negatively acknowledged until the system is reset. The specification \(\text{Fun}\) in Figure 1(a) and \(\text{Rel}\) are defined on different alphabets: the action \(\text{reset}\) is not part of the alphabet of \(\text{Fun}\) and similarly for \(\text{extra}\) in \(\text{Rel}\). The conjunction of the two aspects is depicted in Figure 3(b); observe that the modalities of the transitions labeled by \(\text{reset}\) are directly inherited from these in \(\text{Rel}\).

3. Interface Automata

In [4], de Alfaro and Henzinger introduced interface automata, that are automata whose transitions are typed with \(\text{input}\) and \(\text{output}\) actions rather than with modalities. In this section, we briefly overview the theory of interface automata and refer the reader to [4, 2] for more details.

Definition 3.1. An interface automaton is a tuple \(\mathcal{P} = (X, x_0, A, \rightarrow)\), where \(X\) is the set of states, \(x_0 \in X\) is the initial state, \(A\) is the alphabet of actions, and \(\rightarrow \subseteq X \times A \times X\) is the transition relation.
We decompose $A = A? \uplus A!$, where $A?$ is the set of inputs and $A!$ is the set of outputs. In the rest of the paper, we shall often use $a?$ to emphasize that $a \in A?$ and $a!$ for $a \in A!$. We will also use $x \xrightarrow{a} y$ to emphasize that $(x,a,y) \in \rightarrow$. Observe that if we consider deterministic interface automata, then we can propose a language-based definition similar to the one we gave for modal specifications.

The semantic of an interface automaton is given by a two-player game between: an input player that represents the environment (the moves are the input actions), and an output player that represents the component itself (the moves are the output actions). Input and output moves are in essence orthogonal to modalities. Interface automata are operational models that do not distinguish between an interface and one of its models. More precisely, the model of an interface automaton is any of its refinement. As a consequence, the notion of refinement coincides with the one of satisfaction. Moreover, any interface automaton is always satisfiable except if it is empty.

**Example 3.1.** Two interface automata are depicted in Figure 4 (this example is adapted from [4]). The client $Cl$ in Figure 4(a) is defined over the alphabet $\{ok?, fail?\} \uplus \{msg!\}$. The action fail? never occurs which encodes the assumption that the environment of the action never transmit a fail to the client. The server $Serv$ in Figure 4(b) is defined over the alphabet $\{msg?, ack?, nack?\} \uplus \{sent!, ok!, fail!\}$; when $msg$ is invoked, the server tries to send the message and resents it if the first transmission fails. If both transmission fail, the component reports failure ($fail!$), otherwise it reports success ($ok!$).

Alternatively, properties of interfaces are described in game-based logics, such as ATL or ATL* [7] whose complexities are PSPACE and PTIME-complete, respectively. Refinement between interface automata corresponds to the alternating refinement relation between games [8], i.e., an interface refines another one if its environment is more permissive whereas its component is more restrictive. This problem is known to be PTIME-complete. There is no notion of component reuse and shared refinement is defined in an ad-hoc manner [25].

![Figure 4. Two interface automata to be composed](image)

**Remark 3.1.** Contrary to input/output automata, interface automata are generally not input-enabled. Refinement of input/output automata corresponds to simulation between traces. If the model was not input-enabled, then a refinement could accept less inputs than its abstraction. Game-based approach allows to avoid such situation even when the system is not input enabled.

The main advantage of the game-based approach appears in the definition of composition and compatibility between interface automata. Following [2], two interface automata are composable if they have

---

2Recall that a system is input-enabled if it can react to any input action in any moment.
disjoint sets of output actions compose by synchronizing on shared actions and interleave asynchronously all other actions.

**Definition 3.2. (Product of interface automata)**

Let $P_1 = (X_1, x_{01}, A_1, \rightarrow_1)$ and $P_2 = (X_2, x_{02}, A_2, \rightarrow_2)$ be two interface automata. The product between $P_1$ and $P_2$ is an interface automaton $P_1 \times P_2 = (X, x_0, A, \rightarrow)$, where

- $X = X_0 \times X_1$;
- $x_0 = x_{01} \times x_{02}$;
- $A = A_1 \cup A_2$, and $A? = (A_1? \cup A_2?) \setminus ((A_1? \cap A_2!) \cup (A_2? \cap A_1!))$, and $A! = A_1! \cup A_2!$;
- $\rightarrow$ is defined as follows:
  - For each action $a \in A$ such that $a \notin A_1 \cap A_2$, there exists a transition $(x_1, y_1) \xrightarrow{a} (x_2, y_2)$ iff there exists $(x_1) \xrightarrow{\rightarrow_1} (x_2)$ and $y_1 = y_2$ or $(y_1) \xrightarrow{\rightarrow_2} (y_2)$ and $x_1 = x_2$.
  - For each action $a \in A_1? \cap A_2?$, there exists a transition $(x_1, y_1) \xrightarrow{a?} (x_2, y_2)$ iff there exists $(x_1) \xrightarrow{\rightarrow_1} (x_2)$ and $(y_1) \xrightarrow{\rightarrow_2} (y_2)$.
  - For each $a \in (A_1? \cap A_2!) \cup (A_2? \cap A_1!)$, there exists a transition $(x_1, y_1) \xrightarrow{a!} (x_2, y_2)$ iff there exists $(x_1) \xrightarrow{\rightarrow_1} (x_2)$ and $(y_1) \xrightarrow{\rightarrow_2} (y_2)$.

![Diagram](image.png)

**(a) Product $Cl \times Serv$**

**(b) Composition $Cl \parallel Serv$**

Figure 5. Compatibility of $Serv$ and $Cl$

Since interface automata are not necessarily input-enable (which allows to make assumptions on the environment), in the product $P_1 \times P_2$ of two interface automata $P_1$ and $P_2$, there may be illegal states where one of the automata may produce an output action that is also in the input alphabet of the other automaton, but is not accepted at this state. In most of existing models for interface theories that are based on an input output setting, the interfaces would be declared to be incompatible. This is a pessimistic approach that can be avoided by exploiting the game-based semantic. Indeed, the game semantic allows to propose an optimistic approach:

"Two interfaces can be composed and are compatible if there is at least one environment where they can work together (i.e., where they can avoid the illegal states)."

Deciding whether there exists an environment where the two interfaces can work together is equivalent to checking whether the environment in the product of the interfaces has a strategy to always avoid
illegal states. This can be viewed as a reachability game whose complexity is linear \cite{4}. The set of states from which the environment has a strategy to avoid the illegal states whatever the component does can be recursively computed as follows.

Let $\text{Illegal}(\mathcal{P}_1, \mathcal{P}_2)$ is the subset of pairs $(x_1, x_2) \in X_1 \times X_2$ such that there exists

- either an action $a \in A_1! \cap A_2?$ with $x_1 \xrightarrow{a!} 1$ but not $x_2 \xrightarrow{a?} 2$
- or an action $a \in A_2! \cap A_1?$ with $x_2 \xrightarrow{a!} 2$ but not $x_1 \xrightarrow{a?} 1$

where $x \xrightarrow{a}$ means that $x \xrightarrow{a} y$ for some state $y$. If illegal states exist in the product $\mathcal{P}_1 \times \mathcal{P}_2$, there may still exist refinements of it that possess no illegal states. Such a refinement specifies how the use of the resulting product should be restricted in order to guarantee that illegal states cannot be reached. As proved in \cite{4}, such a largest refinement is obtained by backward pruning $\mathcal{P}_1 \times \mathcal{P}_2$ as follows. For $Y \subseteq X$, the set of states of $\mathcal{P}_1 \times \mathcal{P}_2$, let $\text{pre}_1^!(Y)$ be the subset $Z \subseteq X$ of states $z$ such that $z \xrightarrow{a!} y$ for some $y \in Y$ and $a! \in A!$ (an output action of the product). Let $\text{pre}_1^!(Y) = Y$ and, for $k \geq 0$, $\text{pre}_1^{k+1}(Y) = \text{pre}_1(\text{pre}_1^k(Y))$ and let $\text{pre}_1^*(Y) = \bigcup_k \text{pre}_1^k(Y)$.

The desired pruning consists in:

- Removing $\text{pre}_1^*(\text{Illegal}(\mathcal{P}_1, \mathcal{P}_2))$ from $X$, and
- Removing transitions to states in $\text{pre}_1^*(\text{Illegal}(\mathcal{P}_1, \mathcal{P}_2))$, and
- Removing unreachable states.

The result of applying the pruning to $\mathcal{P}_1 \times \mathcal{P}_2$ is denoted by $\mathcal{P}_1 \parallel \mathcal{P}_2$ and is called the composition of the two interface automata. $\mathcal{P}_1$ and $\mathcal{P}_2$ are called compatible if applying the pruning leaves the initial state \cite{4}. We now recall the two following theorems from \cite{4} that show that interface automata support independent design and substituability.

**Theorem 3.1.** \cite{4}
The composition operation for interface automata is associative and commutative.

**Theorem 3.2.** \cite{4}
Let $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ be three interface automata. If $\mathcal{P}_2$ refines $\mathcal{P}_1$ and the set of shared actions $\mathcal{P}_2 \parallel \mathcal{P}_3$ of is included in the set of shared actions of $\mathcal{P}_1 \parallel \mathcal{P}_3$, then $\mathcal{P}_2 \parallel \mathcal{P}_3$ refines $\mathcal{P}_1 \parallel \mathcal{P}_3$.

**Example 3.2.** The product of the interface automata in Figure 4 is represented in Figure 5(a). The gray state is illegal as the server wants to report a failure (\texttt{fail}) which is not accepted as an input by the client. The result of applying the pruning operation is then depicted in Figure 5(b).

In \cite{16}, Bhaduri has proposed a quotient operation that is the adjoint of the composition operation. This quotient, which is defined for deterministic fragment only, can be characterized with the following theorem.
Theorem 3.3. Consider 2 deterministic interface automata $I_1$ and $I_2$ and let $I_2^\perp$ be the interface $I_2$ where input and output actions have been exchanged. If $I_1$ and $I_2^\perp$ are compatible, then there exists $I$ such that
\begin{enumerate}
  \item $I_1 \parallel I \leq I_2$,
  \item for each $I'$ such that $I_1 \parallel I' \leq I_2$, we have $I' \leq I$, and
  \item $I$ is given by $(I_1 \parallel I_2^\perp)^\perp$.
\end{enumerate}

The theorem above states that, contrary to the case of modal automata, the quotient for interface automata can be derived from the composition operation with a simple switch between input and output actions.

Remark 3.2. The operations between interface automata that have been defined so far do not require an explicit treatment of dissimilar alphabets as it is the case for modal specifications.

Remark 3.3. A fragment of the theory of interface automata and its variable-based extension has been implemented in the tool TICC \cite{6, 1}.

4. On Modal Interfaces

We now present the full theory for modal interfaces. Modal interfaces is an extension of modal specifications where actions are also typed with input and output. This addition allows to propose notions of composition and compatibility for modal specifications in the spirit of interface automata.

The first account on compatibility for modal interfaces was proposed in \cite{35, 40}. In this section, we propose a full interface theory for modal interfaces, which includes composition, product, conjunction, and component reuse via quotient. Moreover, we show that the composition operator proposed in \cite{35, 40} is incorrect and we propose a correction.

We shall start our theory with the definition of profiles which are used to type actions of modal specifications with input and output modalities.

4.1. Profiles

Given an alphabet of actions $A$, a profile is a function $\pi : A \mapsto \{?,!\}$, labeling actions with the symbols ? (for inputs) or ! (for outputs). We write “$a?$” (respectively, $a!$) to express that “$\pi(a) = ?$” (respectively, $\pi(a) = !$). The set of $a \in A$ such that $\pi(a) = ?$ (respectively, $\pi(a) = !$) is denoted $A_?$ (respectively $A_!$). We shall sometimes write by abuse of language, $\pi = (A_?, A_!)$.

We now discuss operations on profiles. We consider a profile $\pi_1 = (A_1?, A_1!)$ defined over $A_1$ and a profile $\pi_2 = (A_2?, A_2!)$ defined over $A_2$.

Refinement between profiles. Profile $\pi_2$ refines $\pi_1$, denoted $\pi_2 \leq \pi_1$, if and only if $A_2 \supseteq A_1$ and both profiles coincide on $A_1$: $\forall a \in A_1, \pi_2(a) = \pi_1(a)$.

Proposition 4.1. The refinement $\leq$ between profiles is transitive.
Proposition 4.4. Let \( S \) with input \( A \) and output \( A' \), is the \( \pi = (A', A!) \) over \( A_1 \cup A_2 \) such that:

\[
\pi_1 \otimes \pi_2 : \begin{cases} 
A! &= (A_1! \cup A_2!) \\
A? &= (A_1? \cup A_2?) \setminus A!
\end{cases}
\]

Proposition 4.2. Let \( \pi_1', \pi_2', \pi_1, \) and \( \pi_2 \) be profiles, the product between profiles is monotonic with respect to the refinement: if \( \pi_1' \leq \pi_1 \) and \( \pi_2' \leq \pi_2 \), then \( \pi_1' \otimes \pi_2' \leq \pi_1 \otimes \pi_2 \).

Conjunction between profiles. The conjunction of \( \pi_1 \) and \( \pi_2 \), denoted \( \pi_1 \wedge \pi_2 \), is the greatest lower bound of the profiles, whenever it exists. More precisely, the conjunction of profiles \( \pi_1 \) and \( \pi_2 \) is defined if and only if both profiles coincide on their common alphabet: \( \forall a \in A_1 \cap A_2, \pi_1(a) = \pi_2(a) \). Whenever defined, the conjunction \( \pi_1 \wedge \pi_2 \) coincides with \( \pi_1 \) for every letter in \( A_1 \) and with \( \pi_2 \) for every letter in \( A_2 \).

Proposition 4.3. Let \( \pi_1, \pi_2 \) and \( \pi \) be profiles, \( \pi \leq \pi_1 \wedge \pi_2 \) if and only if \( \pi \leq \pi_1 \) and \( \pi \leq \pi_2 \). Moreover if for a profile \( \pi' \) we have \( \pi' \leq \pi_1 \) and \( \pi' \leq \pi_2 \), then \( \pi' \leq \pi_1 \wedge \pi_2 \).

Quotient between profiles. The quotient of \( \pi_1 \) and \( \pi_2 \), denoted \( \pi_1 / \pi_2 \), is defined as the adjoint of \( \otimes \), if it exists, namely \( \pi_1 / \pi_2 = \max\{\pi \mid \pi \otimes \pi_2 \leq \pi_1\} \). More precisely, \( \pi_1 / \pi_2 \) is defined when \( A_1? \cap A_2! = \emptyset \), and is thus equal to the profile \( \pi = (A', A!) \) such that:

\[
\pi_1 / \pi_2 : \begin{cases} 
A! &= (A_1! \cup A_2!) \setminus (A_1! \cap A_2!) \\
A? &= [(A_1? \cup A_2?) \setminus A!] \cup (A_1! \cap A_2!)
\end{cases}
\]

Proposition 4.4. Let \( \pi, \pi_1, \) and \( \pi_2 \) be profiles respectively defined over the \( A, A_1 \) and \( A_2 \):

- if \( \pi_1 \otimes \pi_2 \leq \pi \) and \( A_2 \supseteq A \cup A_1 \), then \( \pi_2 \leq \pi / \pi_1 \);
- if \( \pi_2 \leq \pi / \pi_1 \) and \( A_1 \subseteq A \), then \( \pi_1 \otimes \pi_2 \leq \pi \).

4.2. The framework of modal interfaces

We now formally introduce modal interfaces that are modal specification whose actions are also labeled with input and output attributes. We will consider the language representation in the spirit of [44, 43, 46], while Larsen et al. followed the automata-based representation (the two representations are equivalent).

Definition 4.1. (Modal Interface)

A modal interface is a pair \( \mathcal{C} = (S, \pi) \), where \( S \) is a modal specification on the alphabet \( A_S \) and \( \pi : A_S \to \{?, !\} \) is a profile.

A model for a modal interface is a tuple \( (I, \pi') \), where \( I \) is a prefix-closed language and \( \pi' \) is a profile for \( I \). We say that \( (I, \pi') \) strongly implements \( (S, \pi) \), written \( (I, \pi') \models_s (S, \pi) \), if \( I \models_s S \) and \( \pi' \leq \pi \), and similarly for weak implementation. We say that \( (S_2, \pi_2) \leq_s (S_1, \pi_1) \) if \( S_2 \leq_s S_1 \) and \( \pi_2 \leq_s \pi_1 \), with corresponding definition for weak refinement \( \leq_w \). The composition of two models is the pair that results from the shuffle product \( \times \) of their prefix-closed languages and of the product of their profiles.
4.3. Operations on modal interfaces

Operations on modal specifications directly extend to operations on modal interfaces. We have the following definition.

**Definition 4.2.** Consider two modal interfaces $C_1 = (S_1, \pi_1)$ and $C_2 = (S_2, \pi_2)$, and let $\star \in \{\land, \otimes, /\}$. If $\pi_1 \star \pi_2$ is defined, then $C_1 \star C_2 = (S_1 \star S_2, \pi_1 \star \pi_2)$.

The following theorem states that all the characteristic properties of modal specifications directly extend to modal interfaces.

**Theorem 4.1.** Propositions stated in Theorem 2.5 extend to modal interfaces.

4.4. On compatibility for modal interfaces

In this section, we take advantage of profiles to define a notion of composition with compatibility issue for modal interfaces. We shall recap the solution proposed in [35, 40], then we shall show a counter example to Theorem 10 in [35] and then propose our correction. We first recap the translation from interface automata to modal interfaces, which will help to make the link between modalities and input or output actions.

4.4.1. From interface automata to modal interfaces

We recap the translation from interface automata to modal automata that has been proposed in [35]. In this section, we extend this translation to modal specification, the language-extension corresponding to modal automata.

We consider an interface automaton $P = (X, x_0, A, \rightarrow)$. We assume $P$ to be deterministic and we let $L_P$ denote the (prefix-closed) language defined by $P$. The alphabet of $S_P$ is $A_{S_P} = A$ and modalities are defined for all $u \in A_{P}^*$:

\[
\begin{align*}
a? &\in \text{must}_{S_P}(u) & \text{if} & \ u.a? \in L_P \\
a! &\in \text{may}_{S_P}(u) \setminus \text{must}_{S_P}(u) & \text{if} & \ u.a! \in L_P \\
a? &\in \text{may}_{S_P}(u) \setminus \text{must}_{S_P}(u) & \text{if} & \ u \in L_P \\
& & \text{and} & \ u.a? \notin L_P \tag{8} \\
a! &\notin \text{may}_{S_P}(u) & \text{if} & \ u \in L_P \\
& & \text{and} & \ u.a! \notin L_P \\
a &\in \text{may}_{S_P}(u) \setminus \text{must}_{S_P}(u) & \text{if} & \ u \notin L_P.
\end{align*}
\]

Theorem 1 of [35] shows that, with the above correspondence, alternating simulation for interface automata and modal refinement for modal interfaces coincide. Regarding supports, we have:

\[
L_{S_P} = L_P \uplus \{ u.a?.v \mid u \in L_P, u.a? \notin L_P, v \in A_P^* \}. \tag{9}
\]
It is worth making some comments about this translation, given by formulas (8,9). Regarding formula (9), the supporting language $\mathcal{L}_{SP}$ allows the environment to violate the constraints set on it by the interface automaton $P$. When this happens—formally, the environment exits the alternating simulation relation—the component considers that the assumptions under which it was supposed to perform are violated, so it allows itself breaching its own promises and can perform anything afterward. One could also see the violation of assumptions as an exception. Then, $\mathcal{L}_{SP}$ states no particular exception handling since everything is possible. Specifying exception handling then amounts to refining this modal interface.

Formula (8) refines (9) by specifying obligations. Case 1 expresses that the component must accept from the environment any input within the assumptions. Case 2 indicates that the component behaves according to best effort regarding its own outputs actions. Finally, cases 3 and 4 express that the violation of its obligations by the environment are seen as an exception, and that exception handling is unspecified and not mandatory. This embedding is illustrated in Figure 6.

![Figure 6. Embedding of the interface automaton $Cl$ from Figure 4(a) into a modal interface](image)

4.4.2. The composition by Larsen et al. and the bug in Theorem 10 of [35]

We now consider the notion of compatibility for two Modal Interfaces $C_1 = (S_1, \pi_1)$ and $C_2 = (S_2, \pi_2)$ with $S_1$ defined over $A_1$ and $S_2$ defined over $A_2$. We assume that $C_1$ and $C_2$ do not share common output actions (which is the composability requirement similar to the one for interface automata). We first compute the product between $C_1$ and $C_2$ following Definition 4.3.

We then define $\text{Illegal}(C_1, C_2)$ to be the subset of words $u$ belonging to the support of $C_1 \otimes C_2$, such that one interface may produce an output that may not be accepted as an input by the other interface:

$$\text{either}\quad \text{an action } a \in A_1! \cap A_2? \quad \text{with } a \in \text{may}_1(u_1) \setminus \text{must}_2(u_2)$$

$$\text{or}\quad \text{an action } a \in A_2! \cap A_1? \quad \text{with } a \in \text{may}_2(u_2) \setminus \text{must}_1(u_1),$$

where $u_1 = \text{pr}_{A_1}(u)$ and similarly $u_2 = \text{pr}_{A_2}(u)$. In order to get rid of illegal runs, we must first consider the words $v$ having a suffix $v'$ such that $v.v'$ is illegal and $v'$ is a sequence of outputs as no environment can prevent $v'$ to occur from $v$. For $U$ a set of words of Modal Interface $C$, let $\text{pre}_i(U)$ be the set

$$\text{pre}_i(U) = \{v \in \mathcal{L}_C | \exists a! \in \text{may}(v), v.a! \in U\}$$
Let \( \text{pre}_i^0(U) = U \), and, for \( k \geq 0 \), \( \text{pre}_i^{k+1}(U) = \text{pre}_i(\text{pre}_i^k(U)) \). Finally, let \( \text{pre}_i^\ast(U) = \bigcup_k \text{pre}_i^k(U) \).

The composition of two modal interfaces is obtained from their product by removing states in \( \text{pre}_i^\ast(U) \), following the approach outlined for interface automata. Two modal interfaces are compatible if the pruning with the illegal words do not remove the empty word. The composition between \( C_1 \) and \( C_2 \) is denoted \( C_1 \parallel C_2 \).

Theorem 10 in [35, 40] states that

\textit{(Independent Implementability). For any two composable modal interfaces} \( C_1, C_2 \) \textit{and two implementations} \( (I_1, \pi_1) \) \textit{and} \( (I_2, \pi_2) \). \textit{If} \( (I_1, \pi_1) \leq C_1 \) \textit{and} \( (I_2, \pi_2) \leq C_2 \), \textit{then it holds that} \( (I_1, \pi_1) \times (I_2, \pi_2) \leq C_1 \parallel C_2 \).”

**Figure 7.** Counterexample regarding compatibility. Grey-shaded states are to be removed.

**Example 4.1.** Figure 7 depicts two Modal Interfaces \( C_1 \) and \( C_2 \); \textit{may} \ \textit{must} actions are depicted using dashed arrows whereas solid arrows corresponds to \textit{must} actions. \( I_1 \) and \( I_2 \) are implementations of \( C_1 \) and \( C_2 \), respectively. Alphabets are indicated for each modal interface. Parallel composition according to [35] is named \( C_1 \parallel C_2 \). Word \( c?.a! \) is illegal since in the state reached after this run \( C_1 \) may offer \( b! \) whereas \( C_2 \) may (in fact will) not accept it. However, \( c?.a! \) is in the product of the two implementations.

**4.4.3. The correction**

Call \textit{exception} any word in \( L_{C_1 \otimes C_2} \) from which the environment has no strategy to prevent the occurrence of an illegal word, meaning that an illegal word can be obtained from the exception by following only output actions.
Definition 4.3. (compatibility)
The exception language of modal interfaces \( C_1 \) and \( C_2 \) is the language \( E_{C_1 \parallel C_2} = \text{pre}^+ (\text{Illegal}(C_1, C_2)) \). Modal interfaces \( C_1 \) and \( C_2 \) are said to be compatible if and only if the empty word \( \epsilon \) is not in \( E_{C_1 \parallel C_2} \).

Definition 4.4. (parallel composition)
Given two modal interfaces \( C_1 \) and \( C_2 \), the relaxation of \( C_1 \otimes C_2 \) is obtained by applying the following pseudo-algorithm to \( C_1 \otimes C_2 \):

\[
\text{for all } v \text{ in } L_{C_1 \otimes C_2} \text{ do}
\quad \text{for all } a \text{ in } A \text{ do}
\quad \quad \text{if } v \notin E_{C_1 \parallel C_2} \text{ and } v.a \in E_{C_1 \parallel C_2} \text{ then}
\quad \quad \quad \text{for all } w \text{ in } A^* \text{ do}
\quad \quad \quad \quad \text{must}(v.a.w) := \emptyset
\quad \quad \quad \quad \text{may}(v.a.w) := A
\quad \quad \end{align}
\]

If \( C_1 \) and \( C_2 \) are compatible, the relaxation of \( C_1 \otimes C_2 \) is called the parallel composition of \( C_1 \) and \( C_2 \), denoted by \( C_1 \parallel C_2 \). Whenever \( C_1 \) and \( C_2 \) are incompatible, the parallel composition \( C_1 \parallel C_2 \) is defined as the inconsistent modal specification \( \bot \).

If the environment performs an action \( a? \) to which the “if ... then ...” statement applies, then illegal words may exist for certain pairs \((I_1, I_2)\) of strong implementations of \( C_1 \) and \( C_2 \). If this occurs, then \( C_1 \parallel C_2 \) relaxes all constraints on the future of the corresponding runs — Nothing is forbidden, nothing is mandatory: the system has reached a “universal” state. This parallels the pruning rule combined with alternating simulation, in the context of interface automata.

Example 4.2. We now show that our relaxation allows to correct the counter example stated in Figure 7. We observe that our relaxation procedure yields \( [C_1 \parallel C_2]_1 \), with \( A = \{a!, b!, c?\} \), which has \( I_1 \times I_2 \) as an implementation.

Associativity of the parallel composition operator is one of the key requirements of an interface framework, since it enables independent design of sub-systems. Unlike in [35, 40], where associativity is only mentioned, we can now state the following theorem:

Theorem 4.2. The parallel composition operator is commutative and associative.

Thanks to the interplay between modalities and profiles, knowledge about exceptions is preserved by parallel composition. This is the very reason why it is associative. Indeed, the last \( a? \) action in exception runs of the form \( v.a? \) comes with a may modality. In this way, it is distinguished from normal inputs which come with a must modality. When taking the parallel composition with another modal interface with a profile such that \( a? \) is also an input, the resulting modality is a may. In this way, knowledge of the occurrence of an exception is preserved. Whenever this input action \( a? \) is composed with an output \( a! \), this results in an illegal run, meaning that an exception will be triggered earlier.

As for interface automata (Theorem 4 in [4]), strong refinement preserves compatibility, assuming that the refined modal interface does not introduce new shared actions.
Lemma 4.1. Given any three modal interfaces $C_i, i = 1...3$, such that $C_2 \leq_s C_1$ and $A_1 \cap A_3 \supseteq A_2 \cap A_3$, $\text{pr}_{A_1 \cup A_3}(E_{C_2 \parallel C_3})$ is included in $E_{C_1 \parallel C_3}$.

Corollary 4.1. (compatibility preservation)
Given any three modal interfaces $C_i, i = 1...3$, such that $C_2 \leq_s C_1$ and $A_1 \cap A_3 \supseteq A_2 \cap A_3$. $C_1$ compatible with $C_3$ implies that $C_2$ and $C_3$ are also compatible.

Contrary to interface automata for which $C_1 \parallel C_2$ is a refinement of $C_1 \otimes C_2$ [4], relaxation of modal interfaces amounts to compute an abstraction of the product:

Lemma 4.2. Given two modal interfaces $C_1$ and $C_2$:

$$C_1 \otimes C_2 \leq C_1 \parallel C_2$$

Theorem 10 stated in [35, 40] now holds for the parallel composition operator.

Theorem 4.3. (independent implementability)
For any two modal interfaces $C_1, C_3$ and two implementations $(I_1, \pi_1), (I_2, \pi_2)$ such that $(I_1, \pi_1) \models_s C_1$ and $(I_2, \pi_2) \models_s C_2$, it holds that $(I_1, \pi_1) \times (I_2, \pi_2) \models_s C_1 \parallel C_2$.

5. Methodological Considerations

So far, we have not yet study the possible links between composition/product and conjunction. Such study is of clear importance as it relates to architectural design.

In our theory, we use the term view-point when taking the conjunction of some interface with some other one, the view-point, that represents some requirements it must satisfy. In this context, one could wonder whether the same system could be specified as a single component with two view points or, alternatively, as two components, each of them being the result of the system conjuncted with one of the view points. This question amounts to relate the operation of product/composition to the one of conjunction. We distinguish between two possible designs. In the first one, called component-centric design, one first take the conjunction of the two viewpoints and then compose the results. In the second one, called viewpoint-centric design, one consider the viewpoints separately. We have the following result.

Theorem 5.1. Let $C_1, C_2, C_3$ be three modal interfaces. We have the following results.

1. $C_1 \otimes (\text{modint}_2 \land C_3) \leq (S_1 \otimes S_2) \land (S_1 \otimes S_3)$;
2. $C_1 \parallel (\text{modint}_2 \land C_3) \leq (S_1 \parallel S_2) \land (S_1 \parallel S_3)$;
3. The reverse of points 1 and 2 does not hold.

The above theorem expresses that viewpoint-centric design processes leave more room for implementations than component-centric ones.
6. Conclusion, related and future work

This paper presents a modal interface framework, a unification of interface automata and modal specifications. It is a complete theory with a powerful composition algebra that includes operations such as conjunction (for requirements composition) and residuation (for component reuse but also assume/guarantee contract based reasoning [46]). However, the core contributions of the paper are (1) a parallel composition operator that reflects a rich notion of compatibility between components, actually correcting that parallel composition proposed in [35, 40], and (2) a new theory that encompasses dissimilar alphabets.

Interface automata were first introduced as an extension of input/output automata that comes up with an optimistic approach for composition. Modal specifications have been proposed as an extension of process algebra theories [39, 34] which allows for a better distinction between successive implementations (see the introduction of [40] and [34] for some discussion). Modal interface is a model that mixes both i/o automata and modal interfaces.

There are various other approaches for interface theories (see [11] and [27] for surveys). One of them is contract [13, 40, 42], that is a representation where one keeps an explicitly distinction between assumptions on the environment and guarantees on behaviors of the system. A similar approach to ours has been developed in [37] for a non-modal process algebraic framework in which a dedicated predicate is used to model inconsistent processes.

Interface automata and modal specifications are incomparable models as must, may and input, output have orthogonal meanings. Modal specification can be viewed as an abstraction of a set of closed systems (as a modal specification does not allow to distinguish between a component and its environment). As a consequence, specification logics and verification procedures for this model [30, 31] are extensions of those defined for transition systems [41, 22]. Interface automata is a more “open” model (as it distinguishes between the component and its environment) and it is thus not surprising that specification logics and verification procedures for such model correspond to those defined for reactive systems, e.g., ATL [7]. This paper did not focus on verification procedures, but we believe that this research direction is of importance and deserves further studies.

There are several possible directions for future research. A first step would be to implement all the concepts and operations presented in the paper and evaluate the resulting tool on concrete case studies. Extensions of modal specifications can be investigated, where states are described as valuations of a set of variables just as it has been the case for interface automata [20, 6]. One should also propose definition of quotient and conjunction for interface automata.

Another promising direction would be a timed extension of modal interfaces. In [5], de Alfaro et al. proposed timed interface automata that extends timed automata just as interface automata extend finite-word automata. The semantics of a timed interface automaton is given by a timed game [3, 18], which allows to capture the timed dimension in composition, i.e., “what are the temporal ordering constraints on communication events between components? [5]?”. Up to now, composition is the only operation that has been defined on timed interface automata. In [21], Chatain et al. have proposed a notion of refinement for timed games. However monotony of parallel composition with respect to this refinement relation has not been investigated yet. In [15, 14], timed modal specifications are proposed. As modal specifications,

---

3 A closed system is a system that does not interact with an unknown environment. On contrary, an open system is a system that continuously interact with an unknown environment.
timed modal specifications admit a rich composition algebra with product, conjunction and residuation operators. Thus, a natural direction for future research would be to unify timed interface automata and timed modal specifications. This would imply a translation from timed interface automata to timed modal specifications.

Acknowledgments

We are grateful to Barbara Jobstmann and Laurent Doyen who proposed the counter example given in Section 4.4.2.

References


A. Proof of theorem 2.5

We recall and prove the different statements of the theorem.

**Statement A.1.** Weak and strong implementation / refinement relations are related as follows:

\[ \models_s \subseteq \models_w \quad \text{and} \quad \leq_s \subseteq \leq_w \]

**Proof:**
Let \( \mathcal{I} \subseteq C^* \) such that \( \mathcal{I} \models_s \rho S \), that is \( \mathcal{I} \models \rho S_{1\mathcal{C}} \). As \( \rho S_{1\mathcal{C}} \subseteq \rho S_{\mathcal{C}} \), we have \( \mathcal{I} \models \rho S_{\mathcal{C}} \) \([43, 44]\), that is: \( \mathcal{I} \subseteq C^* \models_w \rho S \). Similarly, if \( \rho S_1 \leq_s \rho S_2 \) then, \( \rho S_1 \leq \rho S_{2\mathcal{A}_1} \). As \( \rho S_{2\mathcal{A}_1} \leq \rho S_{\mathcal{A}_1} \) and by transitivity of the modal refinement relation, \( \rho S_1 \leq \rho S_{2\mathcal{A}_1} \), thus: \( \rho S_1 \leq \rho S_2 \).

For the next proofs, we need to define the least upper bound for pseudo-modal specifications together with additional lemmas. We denote by \( \rho S_1 \lor \rho S_2 \) the Least Upper Bound (LUB) of \( \rho S_1 \) and \( \rho S_2 \) defined over the same alphabet. The LUB \( \rho S_1 \lor \rho S_2 \) is defined by:

\[
\begin{align*}
\text{may}(u) &= \text{may}_1(u) \cup \text{may}_2(u) \\
\text{must}(u) &= \text{must}_1(u) \cap \text{must}_2(u)
\end{align*}
\]

(11)

Recall that any prefix-closed language \( \mathcal{I} \subseteq A^* \) can be seen as a modal specification \( S_{\mathcal{I}} \) whose set of must coincides with may: \( \forall u \in A^*, \text{must}(u) = \text{may}(u) = I_u \). Using the embedding of prefix-closed languages in modal specifications, the following lemma provides an algebraic characterization of the pruning \( \rho \):

**Lemma A.1.** ([43, 44])
For \( \rho S \) a pseudo-modal specification, \( \rho(\rho S) = \bigvee_{\mathcal{I} \models \rho S} S_{\mathcal{I}} \).

**Lemma A.2.** Let \( \mathcal{I} \subseteq C \) be a prefix-closed language and \( \rho S \) a pseudo modal specification over \( A \) with \( A \supseteq C \):

\[
\begin{align*}
\mathcal{I} \models_w \rho S &\iff S_{\mathcal{I}} \leq_w \rho S \\
\mathcal{I} \models_s \rho S &\iff S_{\mathcal{I}} \leq_s \rho S
\end{align*}
\]

(12) (13)

**Statement A.2.** Weak and strong modal refinement are both sound and complete w.r.t. weak and strong thorough refinement, respectively:

\[
\begin{align*}
S_2 \leq_w S_1 &\iff \{ \mathcal{I} \mid \mathcal{I} \models_w S_2 \} \subseteq \{ \mathcal{I} \mid \mathcal{I} \models_w S_1 \} \\
S_2 \leq_s S_1 &\iff \{ \mathcal{I} \mid \mathcal{I} \models_s S_2 \} \subseteq \{ \mathcal{I} \mid \mathcal{I} \models_s S_1 \}
\end{align*}
\]

(14) (15)

**Proof:**
We consider the two directions for both strong and weak refinements.

1. We begin with \( \Rightarrow \). Let \( \mathcal{I} \subseteq C^* \) such that \( \mathcal{I} \models_w S_2 \) and \( S_2 \leq_w S_1 \). By definition \( S_2 \leq S_{1\mathcal{A}_2} \) and then \( S_{2\mathcal{C}} \leq (S_{1\mathcal{A}_2})_{\mathcal{C}} \). Since \( C \supseteq A_2 \) then \( (S_{1\mathcal{A}_2})_{\mathcal{C}} = S_{1\mathcal{C}} \) and \( S_{2\mathcal{C}} \leq S_{1\mathcal{C}} \). Thus if \( \mathcal{I} \models_w S_2 \) then \( \mathcal{I} \models S_{2\mathcal{C}} \) and \( \mathcal{I} \models S_{1\mathcal{C}} \), that is \( \mathcal{I} \models_w S_1 \). The proof is similar for the strong refinement.
2. Next, consider \( \iff \). Without loss of generality we can assume that the considered modal specifications \( S_i, i = 1, 2 \) are such that \( S_i = \rho(S_i) \), see theorem 2.1. The same holds then for the weak extensions \((S_i)_{\uparrow A_2}\). By Lemma A.1, \((S_i)_{\uparrow A_2} = \bigvee \mathcal{L}_{(S_i)_{\uparrow A_2}} \mathcal{L}\). Therefore, the right hand side of (14) implies \((S_2)_{\uparrow A_2} \leq (S_1)_{\uparrow A_2}\), which implies the left hand side of (14). The proof is similar for the strong refinement.

\[ \square \]

**Statement A.3.** The following holds regarding conjunction:

\[ \mathcal{I} \models_w S_1 \land S_2 \iff \mathcal{I} \models_w S_1 \text{ and } \mathcal{I} \models_w S_2 \]

**Proof:**

Implication \( \Rightarrow \) is immediate from the definitions and from the properties of the conjunction in the case of a fixed alphabet. For \( \iff \), the same reasoning applies as for the proof of \( \iff \) in statement A.2. \( \square \)

**Statement A.4.** Theorem 2.3 still holds when alphabets are different, provided that strong refinement and implementation are used — it is actually false if weak refinement or implementation are used:

1. If \( S_1' \leq_s S_1 \) and \( S_2' \leq_s S_2 \), then \( S_1' \otimes S_2' \leq_s S_1 \otimes S_2 \);
2. If \( \mathcal{I}_1 \models_s S_1 \) and \( \mathcal{I}_2 \models_s S_2 \), then \( \mathcal{I}_1 \times \mathcal{I}_2 \models_s S_1 \otimes S_2 \);
3. \( S_1' \leq_w S_1 \) and \( S_2' \leq_w S_2 \) do not imply that \( S_1' \otimes S_2' \leq_w S_1 \otimes S_2 \) in general;
4. \( \mathcal{I}_1 \models_w S_1 \) and \( \mathcal{I}_2 \models_w S_2 \) do not imply that \( \mathcal{I}_1 \times \mathcal{I}_2 \models_w S_1 \otimes S_2 \) in general.

**Proof:**

**Proof of statement A.4.1.** Let \( S_i = (A_i, \text{must}_i, \text{may}_i) \) and \( S_i' = (A_i', \text{must}_i', \text{may}_i') \) for \( i = 1, 2 \) and let \( A = A_1 \cup A_2 \) and \( A' = A_1' \cup A_2' \), for any \( u \in (A')^* \), the following holds:

\[
\text{may}_{S_1 \otimes S_2}(u) = \text{may}_{S_1' \uparrow A'} \otimes S_2'(u) = \text{may}_{S_1' \uparrow A'}(u) \cap \text{may}_{S_2' \uparrow A'}(u)
= [\text{may}_{1}(\text{pr}_{A_1}(u)) \cup (A' - A_1)] \cap [\text{may}_{2}(\text{pr}_{A_2}(u)) \cup (A' - A_2)]
\]  

(16)

On the other hand,

\[
\text{may}_{S_1 \otimes S_2}(u) = \text{may}_{S_1 \otimes S_2}(\text{pr}_A(u)) \cup (A' - A)
= \text{may}_{S_1 \otimes S_2}(\text{pr}_A(u)) \cup (A' - A)
= [\text{may}_{1}(\text{pr}_{A_1}(u)) \cap \text{may}_{2}(\text{pr}_{A}(u))] \cup (A' - A)
= [\text{may}_{1}(\text{pr}_{A_1}(u)) \cup (A' - A_1)] \cap [\text{may}_{2}(\text{pr}_{A_2}(u)) \cup (A' - A_2)] \cup (A' - A)
= [\text{may}_{1}(\text{pr}_{A_1}(u)) \cup (A' - A_1)] \cap [\text{may}_{2}(\text{pr}_{A_2}(u)) \cup (A' - A_2)]
\]  

(17)
As \( S'_i \leq_S S_i \) (for \( i \in \{1, 2\} \)), we have, for all \( v \in \mathcal{L}_{S'_i} \):

\[
\text{may}'(v) \subseteq \text{may}(\text{pr}_{A_i}(v)) \cup (A'_i - A_i)
\]

\[
\Rightarrow \text{may}'(v) \cup (A' - A'_i) \subseteq \text{may}(\text{pr}_{A_i}(v)) \cup (A'_i - A_i) \cup (A' - A'_i)
\]

\[
\Leftrightarrow \text{may}'(v) \cup (A' - A'_i) \subseteq \text{may}(\text{pr}_{A_i}(v)) \cup (A' - A_i)
\]

(18)

Now, every \( v \in \mathcal{L}_{S'_i} \) has the form \( v = \text{pr}_{A'_i}(u) \) for some \( u \in \mathcal{L}_{S'_1} \times \mathcal{L}_{S'_2} \); since

\[
\text{pr}_{A_i}(\text{pr}_{A'_i}(u)) = \text{pr}_{A_i}(\text{pr}_{A}(u)),
\]

combining (16), (17), and (18) yields:

\[
\text{may}_{S'_1 \otimes S'_2}(u) \subseteq \text{may}_{(S'_1 \otimes S'_2)_{A'_i}}(v).
\]

Similarly:

\[
\text{must}_{S'_1 \otimes S'_2}(u) = \left[ \text{must}'(\text{pr}_{A'_i}(u)) \cup (A' - A'_i) \right] \cap \left[ \text{must}(\text{pr}_{A'_i}(u)) \cup (A' - A'_2) \right]
\]

On the other hand,

\[
\text{must}_{(S_1 \otimes S_2)_{A'_i}}(u) = \left[ \text{must}'(\text{pr}_{A'_i}(u)) \cup (A' - A'_1) \right] \cap \left[ \text{must}(\text{pr}_{A'_i}(u)) \cup (A' - A'_2) \right]
\]

As \( S'_i \leq_S S_i \) (for \( i \in \{1, 2\} \)), we have, for all \( v \in \mathcal{L}_{S'_i} \):

\[
\text{must}'(v) \supseteq \text{must}_i(\text{pr}_{A_i}(v)) \cup (A'_i - A_i)
\]

\[
\Rightarrow \text{must}'(v) \cup (A' - A'_i) \supseteq \text{must}_i(\text{pr}_{A_i}(v)) \cup (A' - A_i)
\]

Now, every \( v \in \mathcal{L}_{S'_i} \) has the form \( v = \text{pr}_{A'_i}(u) \) for some \( u \in \mathcal{L}_{S'_1} \times \mathcal{L}_{S'_2} \); since

\[
\text{pr}_{A_i}(\text{pr}_{A'_i}(u)) = \text{pr}_{A_i}(\text{pr}_{A}(u)),
\]

we have:

\[
\text{must}_{S'_1 \otimes S'_2}(u) \supseteq \text{must}_{(S_1 \otimes S_2)_{A'_i}}(v),
\]

which completes the proof of statement A.4.1.

**Proof of statement A.4.2.** If \( \mathcal{I}_1 \models_S S_1 \) and \( \mathcal{I}_2 \models_S S_2 \) then by Lemma A.2 \( S_{I_1} \leq_S S_1 \) and \( S_{I_2} \leq_S S_2 \). By the previous statement \( S_{I_1} \otimes S_{I_2} \leq_S S_1 \otimes S_2 \). It is immediat that \( S_{I_1} \otimes S_{I_2} = S_{I_1 \times I_2} \). As a result, \( S_{I_1 \times I_2} \leq_S S_1 \otimes S_2 \) and \( I_1 \times I_2 \models_S S_1 \otimes S_2 \).

**Proof of statement A.4.3.** The following counterexample shows that composition is not monotonic wrt to the weak refinement when alphabets are different. Consider the following three modal specifications:

- \( S_1 \) with \( A_1 = \{a\} \) and \( \text{may}_1(\epsilon) = \text{must}_1(\epsilon) = \emptyset \);
• \( S'_1 \) with \( A'_1 = \{a, b\} \) and \( may'_1(e) = \{b\} \) and \( must'_1(e) = \emptyset \); 
• \( S_2 \) with \( A_{S_2} = \{b\} \) and \( may_2(e) = must_2(e) = \{b\} \)

Then \( S = S_1 \otimes S_2 \) is defined over \( \{a, b\} \) and \( may(e) = must(e) = \{b\} \); and, \( S' = S'_1 \otimes S_2 \) is defined over \( \{a, b\} \) and \( may'(e) = \{b\} \) and \( must'(e) = \emptyset \). Thus we have: \( S'_1 \leq_w S_1 \) and \( S'_1 \otimes S_2 \not{\leq_w} S_1 \otimes S_2 \).

**Proof of statement A.4.4.** The following counter-example shows that \( I_1 \models_w S_1 \) and \( I_2 \models_w S_2 \) do not imply \( I_1 \times I_2 \models_w S_1 \otimes S_2 \):

• \( S_1 \) with \( A_1 = \{a\} \) and \( may_1(e) = must_1(e) = \emptyset \); 
• \( I_1 \) with \( A_{I_1} = \{a, b\} \) and \( I_1 = \{e\} \); 
• \( S_2 \) with \( A_2 = \{b\} \) and \( may_2(e) = must_2(e) = \{b\} \); 
• \( I_2 \) with \( A_{I_2} = \{b\} \) and \( I_2 = \{e, b\} \);

Then \( I_1 \models_w S_1 \) and \( I_2 \models_w S_2 \), \( I_1 \times I_2 = \{\emptyset\} \) and \( may_{S_1 \otimes S_2}(e) = must_{S_1 \otimes S_2}(e) = \{b\} \) thus \( I_1 \times I_2 \) is not a weak implementation of \( S_1 \otimes S_2 \). \( \square \)

**Statement A.5.** Let \( pS_i \), \( pS_i \) and \( S_i \) be pseudo-modal or modal specifications over alphabets \( A, A_i \) for \( i = 1, 2 \), respectively.

1. If \( S_2 \leq_s S_i \) and \( A_1 \subseteq A \) then \( S_1 \otimes S_2 \leq_s S_i \).
2. If \( S_1 \otimes S_2 \leq_s S \) and \( A_2 \supseteq A \cup A_1 \) then \( S_2 \leq_s S_i \).
3. If \( I_1 \models_s S_1 \) and \( I_2 \models_s S_i \) and \( A_1 \subseteq A \) then \( I_1 \times I_2 \models_s S \).
4. If \( \forall I_1 : I_1 \models_s S_1 \Rightarrow I_1 \times I_2 \models_s S \) and \( A_{I_2} \supseteq A \cup A_1 \) then \( I_2 \models_s S_i \).

**Proof:**

**Proof of statement A.5.1.** Suppose \( S_2 \leq_s S_i \) that is:

\[
S_2 \leq [S_i \upharpoonright_{A \cup A_1} / S_i \upharpoonright_{A \cup A_1}]_{A_2}
\]

By definition, this requires: \( A_2 \supseteq A \cup A_1 \). Moreover for all \( u \in A_2^* \):

\[
may_2(u) \subseteq [may(S_i \upharpoonright_{A \cup A_1} / S_i \upharpoonright_{A \cup A_1})(pr_{A \cup A_1}(u))] \cup [A_2 \setminus (A \cup A_1)]
\]

Thus if \( a \in may_2(u) \), either \( a \in [A_2 \setminus (A \cup A_1)] \) or, by definition of the quotient operation, one of the following cases is true:

**C.1**

\[
a \in must_{S_i \upharpoonright_{A \cup A_1}}(pr_{A \cup A_1}(u)) \cap must_{S_i \upharpoonright_{A \cup A_1}}(pr_{A \cup A_1}(u))
\]

which is equivalent to:

\[
a \in [must(pr_A(u))] \cap [must_{1}(pr_A(u)) \cup (A \setminus A_1)]
\]
C.2

\[ a \in \text{may}_{S \uparrow A \cup A_1} \left( \text{pr}_{A \cup A_1} (u) \right) \setminus \text{must}_{S \uparrow A \cup A_1} \left( \text{pr}_{A \cup A_1} (u) \right) \]

which is equivalent to:

\[ a \in \left[ \text{may} (\text{pr}_A (u)) \cup (A_1 \setminus A) \right] \setminus \left[ \text{must} (\text{pr}_A (u)) \right] \]

C.3

\[ a \in \text{mustnot}_{S \uparrow A \cup A_1} \left( \text{pr}_{A \cup A_1} (u) \right) \cap \text{mustnot}_{S \uparrow A \cup A_1} \left( \text{pr}_{A \cup A_1} (u) \right) \]

which is equivalent to\(^4\):

\[ a \in -\left[ \text{may} (\text{pr}_A (u)) \cup (A_1 \setminus A) \right] \cap -\left[ \text{may}_1 (\text{pr}_A (u)) \cup (A \setminus A_1) \right] \]

We want to prove \( S_1 \otimes S_2 \leq S \), that is:

\[ S_{\uparrow A_1 \cup A_2} \otimes S_{\uparrow A_1 \cup A_2} \leq S_{\uparrow A_1 \cup A_2} \]

As \( A_2 \supseteq A \cup A_1 \), we have \( A_1 \cup A_2 = A_2 \); thus, this is equivalent to prove:

\[ S_{\uparrow A_2} \otimes S_2 \leq S_{\uparrow A_2} \]

For any \( u \in (A_2)^* \):

\[ \text{may}_{S_{\uparrow A_2} \otimes S_2} (u) = [\text{may}_1 (\text{pr}_A (u)) \cup (A_2 \setminus A_1)] \cap [\text{may}_2 (u)] \]

On the other hand:

\[ \text{may}_{S_{\uparrow A_2}} (u) = \text{may} (\text{pr}_A (u)) \cup (A_2 \setminus A). \]

Let \( u \in L_{S_2} \) and \( a \in \text{may}_{S_{\uparrow A_2} \otimes S_2} (u) \). As \( a \in \text{may}_2 (u) \), either \( a \in [A_2 \setminus (A \cup A_1)] \), or C.1, C.2 or C.3 is true.

If \( a \in [A_2 \setminus (A \cup A_1)] \) then \( a \in (A_2 \setminus A) \) and \( a \in \text{may}_{S_{\uparrow A_2}} (u) \).

If \( a \notin [A_2 \setminus (A \cup A_1)] \) then \( a \in (A \cup A_1) \). If moreover \( a \in (A_2 \setminus A) \) then \( a \in \text{may}_{S_{\uparrow A_2}} (u) \).

If \( a \notin (A_2 \setminus A) \) then \( a \in A \) and we have to prove \( a \in \text{may} (\text{pr}_A (u)) \) in order to establish that \( a \in \text{may}_{S_{\uparrow A_2}} (u) \). We proceed by contradiction; suppose that \( a \notin \text{may} (\text{pr}_A (u)) \). As \( a \in \text{may}_2 (u) \) but \( a \notin [A_2 \setminus (A \cup A_1)] \), either C.1, C.2 or C.3 should be true:

- as \( a \notin \text{may} (\text{pr}_A (u)) \), C.1 is false;
- as \( a \notin \text{may} (\text{pr}_A (u)) \) and \( a \notin (A_1 \setminus A) \) then C.2 is false;
- as \( a \in \text{may}_{S_{\uparrow A_2} \otimes S_2} (u) \), we also have: \( a \in [\text{may}_1 (\text{pr}_A (u)) \cup (A_2 \setminus A_1)] \).

If \( a \in \text{may}_1 (\text{pr}_A (u)) \) then \( a \notin -[\text{may}_1 (\text{pr}_A (u)) \cup (A \setminus A_1)] \) and C.3 is false.

If \( a \in (A_2 \setminus A_1) \) then \( a \in (A \setminus A_1) \) and \( a \notin -[\text{may}_1 (\text{pr}_A (u)) \cup (A \setminus A_1)] \) thus C.3 is false.

\(^4\)we recall that \( \text{mustnot} (u) = -[\text{may} (u)]. \)
As a result, \( a \in \text{may}(\text{pr}_A(u)) \) and \( a \in \text{may}_{s_{A_2}}(u) \). Note that, so far, we have not used the assumption \( A_1 \subseteq A \) of statement A.5.1.

Similarly, let \( u \in (A_2)^* \):

\[
\text{must}_{s_{A_1} \odot s_2}(u) = \left[ \text{must}_1(\text{pr}_A(u)) \cup (A_2 \setminus A_1) \right] \cap [\text{must}_2(u)]
\]

On the other hand:

\[
\text{must}_{s_{A_2}}(u) = \text{must}(\text{pr}_A(u)) \cup (A_2 \setminus A)
\]

We also have \( S_2 \leq_s S / S_1 \) which entails, for all \( u \in L_{S_2} \):

\[
\text{must}_2(u) \supseteq \left[ \text{must}_{s_{A_1} \cup s_2} / s_{A_1} \cup s_2 \right](\text{pr}_{A_1}(u)) \cup \left[ A_2 \setminus (A \cup A_1) \right]
\]

By assumption, \( A_1 \subseteq A \) thus \( [A_2 \setminus (A \cup A_1)] = A_2 \setminus A \). Moreover, by definition of the quotient operation which includes a pruning step, if the action \( a \) belongs to \( \text{must}_{s_{A_1} \cup s_2} / s_{A_1} \cup s_2 \)(\text{pr}_{A_1}(u)) then:

\[
a \in \text{must}_{s_{A_1} \cup s_2} / s_{A_1} \cup s_2(\text{pr}_{A_1}(u)) \cap \text{must}_{s_{A_1} \cup s_2}(\text{pr}_{A_1}(u))
\]

which is equivalent to:

\[
a \in [\text{must}(\text{pr}_A(u))] \cap [\text{must}_1(\text{pr}_A(u)) \cup (A \setminus A_1)]
\]

Thus, if \( a \in \text{must}_{s_{A_1} \odot s_2}(u) \) then \( a \in \text{must}_2(\text{pr}_A(u)) \). Moreover, if \( a \in \text{must}(\text{pr}_A(u)) \) then \( a \in [\text{must}_1(\text{pr}_A(u)) \cup (A \setminus A_1)] \). As \( A_2 \supseteq A_1 \cup A \) and \( A_1 \subseteq A \), we also have \( (A \setminus A_1) \subseteq (A_2 \setminus A_1) \) and, as a result:

\[
a \in [\text{must}_1(\text{pr}_A(u)) \cup (A_2 \setminus A_1)].
\]

Besides, if \( a \in (A_2 \setminus A) \), as by assumption \( A_1 \subseteq A \), we also have \( a \in (A_2 \setminus A_1) \).

In conclusion, if \( a \in [\text{must}(\text{pr}_A(u)) \cup (A_2 \setminus A)] \) then \( a \in [\text{must}_1(\text{pr}_A(u)) \cup (A_2 \setminus A_1)] \). Thus:

\[
\text{must}_{s_{A_1} \odot s_2}(u) \supseteq \text{must}_{s_{A_2}}(u)
\]

**Proof of statement A.5.2.** We assume that \( S_1 \odot S_2 \leq_s S \) and \( A_2 \supseteq A \cup A_1 \). For all \( u \in L_{S_1} \times L_{S_2} \):

\[
[\text{may}_1(\text{pr}_A(u)) \cup (A_2 \setminus A_1)] \cap [\text{may}_2(u)] \subseteq \text{may}(\text{pr}_A(u)) \cup (A_2 \setminus A)
\]

Let \( u \in L_{S_2} \) and \( a \in \text{may}_2(u) \), we have to prove:

\[
a \in [\text{may}_{s_{A_2} / s_{A_1} \cup s_2}(\text{pr}_{A_1}(u))] \cup [A_2 \setminus (A \cup A_1)]
\]

This is obviously the case if \( a \in [A_2 \setminus (A \cup A_1)] \) so suppose in addition that \( a \in (A \cup A_1) \). Now two cases are possible:
• if \(a \in [\text{may}_1(pr_{A_1}(u)) \cup (A_2 \setminus A_1)]\) then \(a \in [\text{may}(pr_{A_1}(u)) \cup (A_2 \setminus A)].\) As \(a \in (A \cup A_1),\) if
\(a \in (A_2 \setminus A)\) then \(a \in (A_1 \setminus A).\) As a result, \(a \in [\text{may}(pr_{A_1}(u)) \cup (A_1 \setminus A)].\)

If \(a \notin \text{must}(pr_{A}(u))\) then according to the case C.2:
\[a \in \text{may}_{S_{\emptyset \cup A \cup A_1} / S_{1 \cup A \cup A_1}}(pr_{A \cup A_1}(u))\]

If \(a \in [\text{must}(pr_{A}(u))\) then \(a \in \text{must}_{S_1 \cup S_2}(u)\) as \(S_1 \otimes S_2 \leq_s S.\) Thus \(a \in \text{must}_{S_{1 \cup A \cup A_1}}(u)\) that is \(a \in [\text{must}_1(pr_{A_1}(u))] \cup (A_2 \setminus A_1).\) As \(a \in (A \cup A_1),\) if \(a \in (A_2 \setminus A)\) then \(a \in (A_1 \setminus A_1).\) As a result, \(a \in [\text{must}_1(pr_{A_1}(u))] \cup (A_1 \setminus A).\) From the case C.1, we deduce
\[a \in \text{may}_{S_{\emptyset \cup A \cup A_1} / S_{1 \cup A \cup A_1}}(pr_{A \cup A_1}(u)).\]

• else, \(a \notin [\text{may}_1(pr_{A_1}(u)) \cup (A_2 \setminus A_1)];\) thus, \(a \notin \text{may}_{S_1 \cup S_2}(u)\) and \(a \notin \text{must}_{S_1 \cup S_2}(u).\) This entails: \(a \notin [\text{must}(pr_{A}(u)) \cup (A_2 \setminus A)]\) as \(S_1 \otimes S_2 \leq_s S.\) Thus, as \(a \in (A \cup A_1),\) we have
\(a \notin [\text{may}_1(pr_{A_1}(u)) \cup (A \setminus A_1)]\) and \(a \notin [\text{must}(pr_{A}(u)) \cup (A_1 \setminus A)].\) By definition of the quotient operation \(a \in [\text{may}_{S_{\emptyset \cup A \cup A_1} / S_{1 \cup A \cup A_1}}(pr_{A \cup A_1}(u))].\)

We now prove that, for \(u \in \mathcal{L}_{S_2}:\)
\[\text{must} _2 (u) \supseteq [\text{must}_{S_{\emptyset \cup A \cup A_1} / S_{1 \cup A \cup A_1}}(pr_{A \cup A_1}(u))] \cup [A_2 \setminus (A \cup A_1)]\]

If \(a \in [\text{must}_{S_{\emptyset \cup A \cup A_1} / S_{1 \cup A \cup A_1}}(pr_{A \cup A_1}(u))]\) then, by the definition, \(a \in \text{must}_{S_{\emptyset \cup A \cup A_1}}(pr_{A \cup A_1}(u)),\) that is \(a \in \text{must}(pr_{A}(u)).\) Moreover, if \(a \in [A_2 \setminus (A \cup A_1)]\) then \(a \in (A_2 \setminus A)\) as we suppose \(A_2 \supseteq (A \cup A_1).\) As a result:
\[\text{must}(pr_{A}(u)) \cup (A_2 \setminus A) \supseteq [\text{must}_{S_{\emptyset \cup A \cup A_1} / S_{1 \cup A \cup A_1}}(pr_{A \cup A_1}(u))] \cup [A_2 \setminus (A \cup A_1)]\]

As \(S_1 \otimes S_2 \leq_s S,\) for \(u \in \mathcal{L}_{S_2} \times \mathcal{L}_{S_1}:\)
\[[\text{must}_1(pr_{A_1}(u)) \cup (A_2 \setminus A_1)] \cap [\text{must}_2(u)] \supseteq \text{must}(pr_{A}(u)) \cup (A_2 \setminus A)\]

Thus, for all \(u \in \mathcal{L}_{S_2}:
\[[\text{must}_{S_{\emptyset \cup A \cup A_1} / S_{1 \cup A \cup A_1}}(u)_1 \cup (u)_2] \subseteq \text{must}_2(u).\]

**Proof of statement A.5.3.** On one hand if \(I_2 \models_S S / S_1\) then by Lemma A.2, \(S_{I_2} \leq_s S / S_1.\) Moreover by statement A.5.1 we have \(S_{T_2} \otimes S \leq_s S.\)

On the other hand, \(I_1 \models_S S_1\) entails by Lemma A.2, \(S_{I_1} \leq_s S_1.\) Moreover by monotonicity of \(\otimes\) w.r.t. \(\leq_s, S_1 \otimes S_{I_2} \leq_s S_1 \otimes S_2.\)

As a result, by transitivity of \(\leq_s\) we have: \(S_{I_1} \otimes S_{I_2} \leq_s S_1 \leq_s S\) that is \(I_1 \times I_2 \models_S S.\)

**Proof of statement A.5.4.** We associate to every \(u_2 \in I_2\) the words \(u \in I_1 \times I_2\) such that \(pr_{A_{I_2}}(u) = u_2\) and also note \(u_1 = pr_{A_{I_1}}(u).\) Let \(A' = (A_{I_1} \cup A_{I_2}),\) \(a \in (I_2)_{u_2},\) we have to prove:
\[a \in [\text{may}_{S_{\emptyset \cup A \cup A_1} / S_{1 \cup A \cup A_1}}(pr_{A \cup A_1}(u_2))] \cup [A_{I_2} \setminus (A_S \cup A_S)]\]

This is obviously the case if \(a \in [A_{I_2} \setminus (A_S \cup A_S)]\) so suppose in addition that \(a \in (A_S \cup A_S).\) Now two cases are possible:
• if $a \in [(I_1)_{u_1} \cup (A' \setminus A_{I_1})]$ then, as $I_1 \times I_2 \models_s S$, $a \in [\text{may}_{S}(\text{pr}_{A_S}(u)) \cup (A' \setminus A_S)]$. As $a \in (A_S \cup A_S)$, if $a \in (A' \setminus A_S)$ then $a \in (A_S \setminus A_S)$. As a result, $a \in [\text{may}_{S}(\text{pr}_{A_S}(u)) \cup (A_S \setminus A_S)]$. As the result of a residduation has only consistent words, we deduce that the action $a$ belongs to the set $[\text{may}_{S}[A_S \cup A_S] / S_1 \uparrow A_S \cup A_S (\text{pr}_{A_S \cup A_S}(u))]$.

• else, $a \notin [(I_1)_{u_1} \cup (A' \setminus A_{I_1})]$; thus, $a \notin I_1 \times I_2$. As $S_1 \otimes S_2 \models_s S$, this entails: $a \notin [\text{must}_{S}(\text{pr}_{A_S}(u)) \cup (A' \setminus A_S)]$. Thus, as $a \in (A_S \cup A_S)$, we have $a \notin [\text{must}_{S}(\text{pr}_{A_S}(u)) \cup (A_S \setminus A_S)]$. If additionally, we have $a \in [\text{may}_{S}(\text{pr}_{A_S}(u)) \cup (A_S \setminus A_S)]$ then:

$$a \in [\text{may}_{S}[A_S \cup A_S] / S_1 \uparrow A_S \cup A_S (\text{pr}_{A_S \cup A_S}(u))]$$

as $a \notin \text{must}_{S}(\text{pr}_{A_S}(u))$. Otherwise, $a \notin [\text{may}_{S}(\text{pr}_{A_S}(u)) \cup (A_S \setminus A_S)]$ and for all $I_1 \models_s S_1$, we must have: $a \notin (I_1 \times I_2)$. As $a \in (I_2)_{u_2}$, this requires $a \notin (I_1)_{u_1} \cup (A' \setminus A_{I_1})$ whatever $I_1 \models_s S_1$ is. Thus we must have $a \notin (I_1)_{u_1}$ for all $I_1 \models_s S_1$ which is only possible if $a \notin [\text{must}_{S}(\text{pr}_{A_S}(u)) \cup (A_{I_2} \setminus A_S)]$. This entails, by definition of the quotient operation, $a \in [\text{may}_{S}[A_S \cup A_S] / S_1 \uparrow A_S \cup A_S (\text{pr}_{A_S \cup A_S}(u))]$.

We now prove that, for $u_2 \in I_2$:

$$\begin{align*}
(I_2)_{u_2} \supseteq & [\text{must}_{S}[A_S \cup A_S] / S_1 \uparrow A_S \cup A_S (\text{pr}_{A_S \cup A_S}(u))] \cup (A_{I_2} \setminus (A_S \cup A_S))
\end{align*}$$

If $a \in [\text{must}_{S}[A_S \cup A_S] / S_1 \uparrow A_S \cup A_S (\text{pr}_{A_S \cup A_S}(u))]$ then $a \in \text{must}_{S}(\text{pr}_{A_S}(u))$ as a pruning step is included in the definition of the quotient operation. We assume that $I_1 \times I_2 \models_s S$, thus:

$$\begin{align*}
\text{must}_{S}(\text{pr}_{A_S}(u)) \cup (A' \setminus A_S) \subseteq & (I_1 \times I_2)_u \\
\downarrow \\
\text{must}_{S}(\text{pr}_{A_S}(u)) \cup (A' \setminus A_S) \subseteq & (I_2)_{u_2} \cup (A' \setminus A_{I_2}) \\
\downarrow \\
\text{must}_{S}(\text{pr}_{A_S}(u)) \subseteq & (I_2)_{u_2} \cup (A' \setminus A_{I_2}) \\
\downarrow \\
\text{must}_{S}(\text{pr}_{A_S}(u)) \cap \neg (A' \setminus A_{I_2}) \subseteq & (I_2)_{u_2}
\end{align*}$$

As $\neg (A' \setminus A_{I_2}) = A_{I_2}$ and $\text{must}_{S}(\text{pr}_{A_S}(u)) \subseteq A_S \subseteq A_{I_2}$, we deduce that: $a \in (I_2)_{u_2}$. If $a \in [A_{I_2} \setminus (A_S \cup A_{I_1})]$ then $a \in (A' \setminus A_S)$. As previously noted, if we assume that $I_1 \times I_2 \models_s S$, then:

$$\begin{align*}
\text{must}_{S}(\text{pr}_{A_S}(u)) \cup (A' \setminus A_S) \subseteq & (I_2)_{u_2} \cup (A' \setminus A_{I_2})
\end{align*}$$

If $a \in (A' \setminus A_S)$ then, in particular, $a \in A_{I_2}$ and $a \notin (A' \setminus A_{I_2})$. As a result: $a \in (I_2)_{u_2}$. □
B. Proof of theorem 4.1

The part of the Theorem regarding $\otimes$ follows if we can prove that product $\otimes$ of profiles is monotonic with respect to the order defined on profiles. To this end, for $i = 1, 2$, let $\pi_i' \leq \pi_i$ and set $\pi' = \pi_1' \otimes \pi_2'$ and $\pi = \pi_1 \otimes \pi_2$. Decompose $A'_1! = A_i! \cup \bar{A}_i!$ and $A'_2? = A_i? \cup \bar{A}_i?$ where $\cup$ means union of disjoint sets. Thus, we have $A'! = A'_1! \cup A'_2! = (A_1! \cup \bar{A}_1!) \cup (A_2! \cup \bar{A}_2!) = A! \cup \bar{A}!$, which proves that $\pi' \leq \pi$.

Regarding quotient, by definition of the residuation for profiles, we have $\pi_1 \otimes \pi_2 \leq \pi$ if and only if $\pi_2 \leq \pi/\pi_1$, which is enough to prove that theorem 2.4 holds for modal interfaces with arbitrary alphabets. The second part of the Theorem regarding quotient amounts to proving that $\pi'_1 \leq \pi_1$ and $\pi_2 \leq \pi/\pi_1$ together imply $\pi_1 \otimes \pi_2 \leq \pi$, which was already proved.

C. Proof of Theorem 4.2

Commutativity of $\parallel$ immediately holds by definition. We now consider associativity. Let three modal interfaces $C_1, C_2, C_3$. We characterize the set of illegal words in $((C_1 \parallel C_2) \parallel C_3)$ and then prove that rearranging the parentheses will not change this set.

In the sequel we shall write $A_i$, $\text{must}_i$, $\text{may}_i$ the elements of $C_i$ (with $i = 1, 2, 3$) and $A_{ij}$, $\text{must}_{ij}$, $\text{may}_{ij}$ the elements of $C_i \otimes C_j$ (for $(i, j) \in \{1, 2, 3\} \times \{1, 2, 3\}$, such that $i \neq j$). We shall also write $u_i$ for $\text{pr}_{A_i}(u)$ and $u_{ij}$ for $\text{pr}_{A_i \cup A_j}(u)$.

Observe first that, by definition, $\otimes$ is associative. Moreover, a word $u$ is illegal in $C_1 \otimes C_2$ iff

$$\text{may}_1^i(u_1) \setminus \text{must}_2^j(u_2) \cup (\text{may}_2^j(u_2) \setminus \text{must}_1^i(u_1)) \neq \emptyset$$

where $u_i = \text{pr}_{A_i}(u)$ and $\text{may}_1^i(u_1) = \text{may}_1(u_1) \cap A_1!$ and similarly for other cases. Then, in building $C_1 \parallel C_2$ from $C_1 \otimes C_2$, relaxation of def. 4.4 applies to every word $v \in L_{C_1 \otimes C_2}$ such that

$$\exists b? \in \text{may}_{12}(v) : v.b? \in I$$

Consequently, every word in $L_{C_1 \parallel C_2}$:

$$\exists b? \in \text{may}_{12}(v), \exists w \in (A_1! \cup A_2!)^* \Rightarrow v.b?w \text{ satisfies (19).}$$

(20)

Observe that (20) rewrites as

$$\exists b? \in \text{may}_{12}(v), \exists w \in (A_1! \cup A_2!)^* \Rightarrow v.b?w \text{ satisfies (19).}$$

(22)

Apply this to the pair $(C_1 \parallel C_2, C_3)$: word $u$ is illegal in $(C_1 \parallel C_2) \otimes C_3$ iff

$$\text{may}_{12}^i(u_{12}) \setminus \text{must}_{3}^j(u_3) \cup (\text{may}_{3}^j(u_3) \setminus \text{must}_{12}^i(u_{12})) \neq \emptyset$$

(23)

where $A_{12}$ is the alphabet of $C_1 \parallel C_2$. Let $U$ be the set of all such $u$’s, and set $I = \text{pre}_{\parallel}^*(U)$. Then, in building $(C_1 \parallel C_2) \parallel C_3$ from $(C_1 \parallel C_2) \otimes C_3$, relaxation applies to every word $v \in L_{(C_1 \parallel C_2) \parallel C_3}$ satisfying
Let us summarize how the two conditions (20) was rewritten:

\[
\exists b? \in \text{may}(v), \exists w \in (A_1! \cup A_2! \cup A_3!)^* \Rightarrow v.b?.w \text{ satisfies (23).} \tag{24}
\]

Let us further analyse (23). Two cases must be considered:

1. \( u_{12} \) has reached a universal state of \( C_1 \parallel C_2 \): in this case, by (21), \( u_{12} \) has a strict prefix \( \hat{u}_{12} \in L_{C_1, C_2} \) satisfying (20), meaning that \( \hat{u}_{12}.b? \) may for some \( b? \), by subsequently performing only output actions, reach a deadlock in the product of the pair \( (C_1, C_2) \).

2. \( u_{12} \) has not reached a universal state of \( C_1 \parallel C_2 \): in this case, \( u_{12} \in L_{C_1, C_2} \) and

\[
\begin{align*}
\text{may}_{12}^1(u_{12}) &= \text{may}_{1}^1(u_1) \cup \text{may}_{2}^1(u_2) \\
\text{must}_{12}^2(u_{12}) &= (\text{must}_{1}^2(u_1) \cap \text{must}_{2}^2(u_2)) \cup (A_1 \setminus A_2) \cup (A_2 \setminus A_1)
\end{align*}
\]

Hence the non-emptiness of (23) is equivalent to \( u_{12} \) causing a deadlock in the pair \( (C_1, C_3) \) or the pair \( (C_2, C_3) \).

Let us summarize how the two conditions (20) was rewritten:

\[
\exists b? \in \text{may}(v), \exists w \in (A_1! \cup A_2! \cup A_3!)^* \Rightarrow u = v.b?.w \text{ satisfies the following condition :}
\]

- There exists a pair \( (i, j) \in \{1, 2, 3\} \times \{1, 2, 3\} \), such that \( i \neq j \) and \( u_{ij} = \text{pr}_{A_i}(u) \) possesses a prefix \( \hat{u}_{ij}.b? \) that may, for some \( b? \) and by subsequently performing only output actions, reach a deadlock in the pair \( (C_i, C_j) \).

The bottom line is that the condition and (20) is indeed symmetric with respect to the considered three modal interfaces. This proves associativity of \( \parallel \).

D. Proof of Lemma 4.1 and Corollary 4.1

Consider three modal interfaces \( C_i, i = 1 \ldots 3 \) such that \( C_2 \subseteq C_1 \) and \( A_2 \cap A_3 \subseteq A_1 \cap A_3 \). We start with a technical lemma:

**Lemma D.1.** \( \text{pr}_{A_1 \cup A_3}(\text{Illegal}(C_2, C_3)) \) is included in \( \text{Illegal}(C_1, C_3) \).

**Proof:** Consider an illegal word \( u \in \text{Illegal}(C_2, C_3) \) for \( C_2 \circ C_3 \). This means that there exists an action \( a \in A_2 \cap A_3 \) such that (i) either \( a \) is an output of \( C_2 \) and an input of \( C_3 \), such that \( a \in \text{may}_2(\text{pr}_{A_2}(u)) \) and \( a \notin \text{must}_3(\text{pr}_{A_3}(u)) \), or (ii) \( a \) is an input of \( C_2 \) and an output of \( C_3 \), such that \( a \notin \text{must}_2(\text{pr}_{A_2}(u)) \) and \( a \in \text{may}_3(\text{pr}_{A_3}(u)) \).

By Definition 2.5, \( u \) is also in \( L_{C_1, C_2, C_3} \). By Definition 2.5, \( u' = \text{pr}_{A_1 \cup A_3}(u) \) belongs to \( L_{C_1, C_3} \).

Since it is assumed that \( A_2 \cap A_3 \subseteq A_1 \cap A_3 \), action \( a \) belongs to \( A_1 \cap A_3 \). By Definition 2.4, either \( a \) is an output of \( C_1 \) and an input of \( C_3 \), such that \( a \in \text{may}_1(\text{pr}_{A_1}(u')) \) and \( a \notin \text{must}_3(\text{pr}_{A_3}(u')) \), or
(ii) \(a\) is an input of \(C_1\) and an output of \(C_3\), such that \(a \notin \text{must}_1(\text{pr}_{A_1}(u'))\) and \(a \in \text{may}_3(\text{pr}_{A_3}(u'))\). Meaning that \(u' \notin \text{Illegal}(C_1, C_3)\), which proves Lemma D.1.

We can now prove Lemma 4.1. Recall that \(A_1! \cup A_3!\) is included in \(A_2! \cup A_3!\). Hence, the set of actions \(\text{pr}_{A_1 \cup A_3}(\text{pre}_1^{\star}(\text{Illegal}(C_2, C_3)))\) is included in \(\text{pre}_1^{\star}(\text{pr}_{A_1 \cup A_3}(\text{Illegal}(C_2, C_3)))\), which in turn included in \(\text{pre}_1^{\star}(\text{Illegal}(C_1, C_3))\), thanks to Lemma D.1.

Corollary 4.1 is an immediate consequence of Lemma 4.1. Assume \(C_2\) and \(C_3\) incompatible, meaning that \(\epsilon \in E_{C_2 \parallel C_3}\). By Lemma 4.1, \(\epsilon = \text{pr}_{A_1 \cup A_3}(\epsilon) \in E_{C_1 \parallel C_3}\). Hence \(C_1\) and \(C_3\) are also incompatible.

E. Proof of Lemma 4.2

**Lemma E.1.** Given two modal interfaces \(C_1\) and \(C_2\):

\[
C_1 \otimes C_2 \leq C_1 \parallel C_2
\]

Two cases are possible:

- if \(u \in L_{C_1 \otimes C_2} \setminus E_{C_1 \parallel C_2}\) then \(\text{must}_{C_1 \otimes C_2}(u) = \text{must}_{C_1 \parallel C_2}(u)\) and \(\text{may}_{C_1 \otimes C_2}(u) = \text{may}_{C_1 \parallel C_2}(u)\);

- if \(u \in E_{C_1 \parallel C_2}\) then \(u \in L_{C_1 \parallel C_2}\) and \(\text{must}_{C_1 \parallel C_2}(u) = \emptyset\) and \(\text{may}_{C_1 \parallel C_2}(u) = A\).

Thus, \(\text{must}_{C_1 \otimes C_2}(u) \supseteq \text{must}_{C_1 \parallel C_2}(u)\) and \(\text{may}_{C_1 \otimes C_2}(u) \subseteq \text{may}_{C_1 \parallel C_2}(u)\).

F. Proof of Theorem 4.3

If \((\mathcal{I}_1, \pi_1) \models_s C_1\) and \((\mathcal{I}_2, \pi_2) \models_s C_2\), then, by Theorem 4.1, \((\mathcal{I}_1, \pi_1) \times (\mathcal{I}_2, \pi_2) \models_s C_1 \otimes C_2\).

By the previous lemma and by the generalization of Theorem 1 in Theorem 4.1: \((\mathcal{I}_1, \pi_1) \times (\mathcal{I}_2, \pi_2) \models_s C_1 \parallel C_2\).