Modal Interfaces:
Unifying Interface Automata and Modal Specifications

J.-B. Raclet¹, E. Badouel², A. Benveniste¹, B. Caillaud¹, A. Legay³ and R. Passerone²

¹ INRIA, France. {name.surname@inria.fr} ² University of Trento, Italy. roberto.passerone@unitn.it

1 Semantic of Modal Interfaces

The modal interface theory is a unification of:

- **Modal specifications**: transitions are given a modality may or/and
  must with consistency must ⊆ may;
- **Interface automata**: actions are originated either from the component
  (output: a) or from the environment (input: a?); input-enabledness is
  not required.

![Interface Diagram]

Refinement ≤: have more must and less may.

![Refinement Diagram]

Implementation |=: keep all must and some may.

![Implementation Diagram]

Refinement entails substitutability as it is sound and complete:

\[ S_1 \leq S_2 \Leftrightarrow \text{Impl}(S_1) \subseteq \text{Impl}(S_2) \]

2 Conjunction

Conjunction is needed for:

- Composition of different viewpoints for a same component;
- Shared refinement: use an implementation for several interfaces.

Product of underlying automata; \( \cap \) may, \( \cup \) must:

\[
\begin{array}{c|c|c|c|c}
\cap & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
\hline
\rightarrow & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
\rightarrow & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
\rightarrow & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
\end{array}
\]

and backward pruning of inconsistencies \( j \).

\( S_1 \cap S_2 \) is the GLB of \( S_1 \) and \( S_2 \); moreover:

\[
\text{Impl}(S_1 \cap S_2) = \text{Impl}(S_1) \cap \text{Impl}(S_2)
\]

3 Product

The product reflects the standard composition of implementations.

Product of underlying automata; \( \cap \) may, \( \cup \) must:

\[
\begin{array}{c|c|c|c|c}
\cap & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
\hline
\rightarrow & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
\rightarrow & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
\rightarrow & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
\end{array}
\]

The product is associative. Moreover it ensures independent design:

- \( S'_1 \leq S_1 \) and \( S'_2 \leq S_2 \) \( \Rightarrow \) \( S'_1 \cap S'_2 \leq S_1 \cap S_2 \);
- \( I_1 \models S_1 \) and \( I_2 \models S_2 \) \( \Rightarrow \) \( I_1 \cap I_2 \models S_1 \cap S_2 \).

4 Quotient

Quotient is needed for:

- Solving equations, \( \max : S_1 \times X \leq S_2 \);
- Implication for Assume/Guarantee contracts: \( G/A \).

Product of underlying automata; may/must as follows:

\[
\begin{array}{c|c|c|c|c}
\cap & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
\hline
\rightarrow & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
\rightarrow & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
\rightarrow & \rightarrow & \rightarrow & \rightarrow & \rightarrow \\
\end{array}
\]

and backward pruning of inconsistencies \( j \).

- \( S_1 \cap S_2 \leq S \quad \Rightarrow \quad S_2 \leq S/S_1 \);
- \( I_2 \models S/S_1 \quad \Rightarrow \quad \forall I_1 \models S, I_1 \cap I_2 \models S \).

5 Dealing with dissimilar alphabets

Needed as implementations can have a larger alphabet and interfaces may have different alphabets.

Alphabet equalization should be neutral: it should not constrain what other interfaces may want to require regarding these extra actions:

- For \( S_1 \land S_2 \) equalize by adding may self-loops onto \( S_1 \) and \( S_2 \);
- For \( S_1 \lor S_2 \) equalize by adding must self-loops onto \( S_1 \) and \( S_2 \);
- For \( S_1/S_2 \) equalize by adding may self-loops onto \( S_1 \) and must self-loops onto \( S_2 \).

6 Compatibility

Two interfaces are compatible if there exists an environment where they can avoid illegal states (i.e., states where an interface wishes to produce an output that is not accepted as input by the other interface).

Composition \( S_1 \parallel S_2 \) is obtained by:

- computing illegal states \( \text{Illegal}(S_1, S_2) \);
- computing exception states \( \hat{E} = \text{pre}(\text{Illegal}(S_1, S_2)) \);
- replacing transitions leading to \( \hat{E} \) by transitions to a new universal state.

Compatibility is preserved by refinement and \( \parallel \) is associative and monotonic for the refinement preorder.