Modal Interfaces: Unifying Interface Automata and Modal Specifications

Authors: Jean-Baptiste Raclet et al.

Presenters: Ilge Akkaya
Forrest Iandola

Introduction

- Modal Specifications: describe *composition* of functions in a system
- Interface Automata: describe *interfaces* among functions in a system
- Modal **Interfaces**: Modal Specifications + Interface Automata + some glue

Modal Specifications

Key Concepts

- Modal Specifications: describe *composition* of functions in a system
- *must(u)* = set of functions that *must* execute after function *u*
- *may(u)* = set of functions that *may* execute after function *u*

Modal Specifications

Key Concepts

- Modal Automata: similar to Nondeterministic Finite Automata, but with *must* and *may* properties to the transitions
- Pseudo-Modal Automata: *must* is not necessarily a subset of *may*
  - A transition can be both required and disallowed
  - This property is useful in derivations

Modal Specifications

Notation

- S = modal specification
- pS = pseudo-modal specification
- I = implementation
- L = language
- A = alphabet

"I implements S" in Modal Specifications

- If I *implements* system pS, then may( ) and must( ) need to be the same for I and pS
  - Assuming that I and pS have the same notation ("language")
- If I *strongly implements* pS, then I and pS have the same may( ) and must( ), except where the languages of I and pS differ
- If I *weakly implements* pS, then I and pS have the same may( ), except where the languages of I and pS differ
  - But I and pS might not share the same must( ).

"S2 refines S1" in Modal Specifications

- When we *refine* a system or implementation, all pre-existing may( ) and must( ) requirements need to be met
- "S2 *refines* S1," "S2 *strongly refines* S1," "S2 *weakly refines* S1" follow roughly the same logic that we've already seen
  - Weak and strong are related to whether must( ) needs to hold

"Language Extensions" in Modal Specifications

- Motivation: Each module of a system may have its own language and alphabet

Example

- Given alphabets A and C. A is a subset of C
- L1 is a language. L1 is a subset of C*
- *Extension of L1 to A* is the subset of L1 that can be expressed using the alphabet (A - C).
- Shorthand for *extension of L1 to A:* $(\mathcal{L}_1)_{\uparrow A}$

Operators in Modal Specifications

- Consider languages L1 in A1* and L2 in A2*
- Shuffle product (L1 x L2)

$$\mathcal{L}_1 \times \mathcal{L}_2 = (\mathcal{L}_1)_{\uparrow A} \cap (\mathcal{L}_2)_{\uparrow A}, \ \ where \ A = A_1 \cup A_2$$

Operators in Modal Specifications

- Conjunction (S1 ^ S2)
  - Intersection of the may( ) sets, and union of the must ( ) sets
  - Keep all musts, remove mays that aren't shared in S1 and S2
- Parallel Product ($S1 \otimes S2$)
  - Intersection of may( ) and must( ) sets for S1 and S2
- Quotient (S1 / S2)
  - Keep the both may( ) sets but remove both must( ) sets
  - This is a rough description, there are other details

Interface Automata: Overview

- Game semantics based variation of I/O automata
- Two player game:
  - *Input*: environment
  - *Output*: component itself
- *Optimistic* composition: two interfaces can be composed if there exists at least one environment that supports both (for all possible behavior of the Output player)

Definition: Interface Automaton

- An interface automaton is a tuple $P = (X, x_0, A, \rightarrow)$
  - X: set of states
  - Initial state: $x_0 \in X$
  - A: alphabet of actions, $A = A? \cup A!$
    - A?: set of inputs
    - A!: set of outputs
  - Transition relation: $\rightarrow \subseteq X \times A \times X$

Game-Based Model

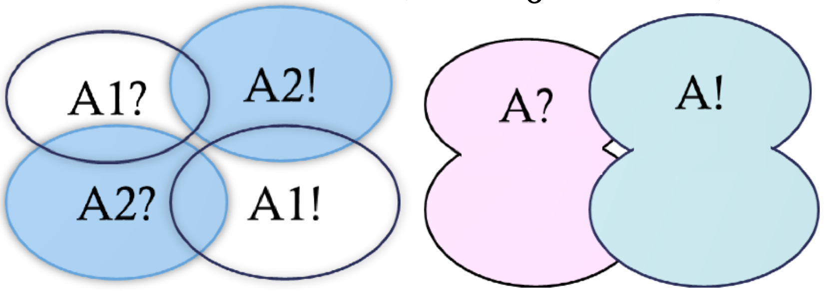
- Input player: Environment
  - Moves represent input actions
- Output player: Component
  - Moves represent output actions
- Interface automata are operational modes
  - No notion of model
  - Satisfiability or consistency not defined
- Refinement between interface automata
  - An interface I refines an interface J, if I's environment is more permissive whereas its component is more restrictive.

Product of Interface Automata

- Two interface automata

$$P_1 = (X_1, x_{01}, A_1, \rightarrow_1) \quad P_2 = (X_2, x_{02}, A_2, \rightarrow_2)$$

- P_1 x P_2 is also an interface automaton $P = (X, x_0, A, \rightarrow)$
  - $X = X_1 \times X_2$
  - $x_0 = x_{01} \times x_{02}$
  - $A = A_1 \cup A_2$



$$A? = (A_1? \cup A_2?) \setminus ((A_1? \cap A_2!) \cup (A_2? \cap A_1!))$$

$$A! = A_1! \cup A_2!$$

Product of Interface Automata

Transition relation is defined as

- For each action $a \in A$ such that $a \notin A_1 \cap A_2$

1- There exists a transition $(x_1, y_1) \xrightarrow{a} (x_2, y_2)$ iff there exists a transition from x_1 to x_2 in P1 and $y_1 = y_2$, or there exists a transition from y_1 to y_2 in P2, and $x_1 = x_2$

(in P1, there exists a transition from x1 to x2 under a, and y remains unchanged, or in P2, there exists a transition from y1 to y2 under a, and x remains unchanged)

Product of Interface Automata

2 – For each action $a \in A_1? \cap A_2?$

And for each action $a \in (A_1? \cap A_2!) \cup (A_2? \cap A_1!)$

A transition exists in P iff there exist the respective transitions from x1 to x2 and y1 to y2 in P1 and P2, respectively.

Optimistic Semantics

- There may be illegal states if
  - One of the automata produces an output action that is in the input alphabet of the other automaton, but is not accepted at that state.
- This situation is not handled as an **incompatibility** in this framework
- If they can avoid the illegal states, they are still compatible. (existence of one illegal state does not violate compatibility) => **Optimistic**

Optimistic Semantics

- Deciding if there exists such environment is equivalent to
- **Checking** whether the environment always has a strategy to avoid illegal states.

Computing Safe States

- Illegal(P1,P2) is the subset of pairs $(x_1, x_2) \in X_1 \times X_2$

s.t. there exists either an action that is an output of P1 and an input of P2 that has a valid transition in P1 but not accepted in x2 by P2

Or an action that is an output of P2 and input of P1 with a valid transition in P2 but not accepted in x1 by P1.

Composition

- There can still exist refinements of P1 x P2 that ensures such illegal states cannot be reached. Such a refinement $Y \subseteq X_1 \times X_2$ can be found as follows:
  - $Pre_!(Y)$ is the subset Z such that a transition z→ y exists from all z in Z, to a state in Y (called exception states)
  - Iteratively remove $pre_!(\mathit{Illegal}(P_1,P_2))$ from X
  - Remove transitions to states in $pre_!(\mathit{Illegal}(P_1,P_2))$
  - Remove unreachable states
- Result of the pruning denoted by $P_1 \| P_2$ => called the composition

- S1 || S2 obtained by
  - Computing illegal states : Illegal(S1,S2)
  - Computing exception states: $pre_!$(Illegal(S1,S2)): states from which the illegal states can be reached
  - Replacing transitions leading to exception states by transitions to a new universal state.
  - || is associative and monotonic for the refinement preorder.

Modal Interfaces

- Extension of modal specifications where
- Actions are also typed as input or output.
- This allows to propose notions of composition and compatibility
- Use profiles to type actions of model specifications with Input/Output

Profiles

- For an alphabet of actions A, a profile is a function

$$\pi : A \mapsto \{?, !\}$$

  - where
  - π(a) = ? denotes a is an input action and
  - π(a) = ! denotes a is an output action.

Maps each action in the alphabet to either the input or the output set

Profiles: Properties

- Product between profiles: composition

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

- Refinement between profiles:

$$\pi_2 \leq \pi_1 \iff A_2 \supseteq A_1$$

  - And if both profiles coincide on A_1

Profiles: Properties

- Conjunction: $\pi_1 \wedge \pi_2$
  - GLB of the profiles, if exists (iff both profiles coincide on the common alphabet)
  - Whenever defined, the conjunction coincides with π_1 for every letter in A_1 and with π_2 on A_2.
- Quotient: π_1/π_2 is defined as the adjoint:

$$\pi_1/\pi_2 = max\{\pi | \pi \otimes \pi_2 \leq \pi_1\}$$

Modal Interfaces

- DEFINITION: A modal interface is a pair C=(S, π),
  - S: modal specification on alphabet A_S
  - π: $A_s \rightarrow \{?,!\}$ is a profile.

- Model for a modal interface is a tuple (I, π'), I: prefix closed language, π': profile for I.

(I, π') strongly implements (S, π) if $I \models_S S \quad and \quad \pi' \leq \pi$

Weak implementation $I \models_w S \quad and \quad \pi' \leq_w \pi$

Operations on Modal Interfaces

- Conjunction, product and quotient on C1, C2 defined as:

$$C_1 \star C_2 = (S_1 \star S_2, \pi_1 \star \pi_2), \star \in \{\wedge, \otimes, /\}$$

All the properties of modal specifications directly extend to modal interfaces, since operation distributes over the modal specification and the profile separately.

Interface Automata → Modal Interfaces

- The supporting language allows the environment to violate the constraints set on it by P.
- This can be interpreted as an exception
- Once this happens, P has no promises and can perform anything.
- Exception handling needs to consider refining this modal interface.

Interface Automata → Modal Interfaces

Refinement :

Consider an interface automaton $P = (X, x_0, A, \rightarrow)$

Assume determinacy. L_P: language defined by P.

Alphabet of S_p: A_{sp} and modalities defined for all u in A_p^* :

$$
\begin{array}{lll}
a? \in must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u.a? \in \mathcal{L}_\mathcal{P} \\
a! \in may_{\mathcal{S}_\mathcal{P}}(u) \setminus must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u.a! \in \mathcal{L}_\mathcal{P} \\
a? \in may_{\mathcal{S}_\mathcal{P}}(u) \setminus must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u \in \mathcal{L}_\mathcal{P} \\
 & \text{and} & u.a? \notin \mathcal{L}_\mathcal{P} \\
a! \notin may_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u \in \mathcal{L}_\mathcal{P} \\
 & \text{and} & u.a! \notin \mathcal{L}_\mathcal{P} \\
a \in may_{\mathcal{S}_\mathcal{P}}(u) \setminus must_{\mathcal{S}_\mathcal{P}}(u) & \text{if} & u \notin \mathcal{L}_\mathcal{P}.
\end{array}
$$

$$
\begin{array}{lll}
a? \in must_{\mathcal{S}_{\mathcal{P}}}(u) & \text{if} & u.a? \in \mathcal{L}_{\mathcal{P}} \\
a! \in may_{\mathcal{S}_{\mathcal{P}}}(u) \setminus must_{\mathcal{S}_{\mathcal{P}}}(u) & \text{if} & u.a! \in \mathcal{L}_{\mathcal{P}} \\
a? \in may_{\mathcal{S}_{\mathcal{P}}}(u) \setminus must_{\mathcal{S}_{\mathcal{P}}}(u) & \text{if} & u \in \mathcal{L}_{\mathcal{P}} \\
 & \text{and} & u.a? \notin \mathcal{L}_{\mathcal{P}} \\
a! \notin may_{\mathcal{S}_{\mathcal{P}}}(u) & \text{if} & u \in \mathcal{L}_{\mathcal{P}} \\
 & \text{and} & u.a! \notin \mathcal{L}_{\mathcal{P}} \\
a \in may_{\mathcal{S}_{\mathcal{P}}}(u) \setminus must_{\mathcal{S}_{\mathcal{P}}}(u) & \text{if} & u \notin \mathcal{L}_{\mathcal{P}}.
\end{array}
$$

- Case1: Components must accept an input within assumptions
- Case 2: component behaves according to best effort regarding its output actions
- Cases 3,4: violation of the obligations by the environment are seen as an exception and exception handling is not specified.

The composition by Larsen et al.

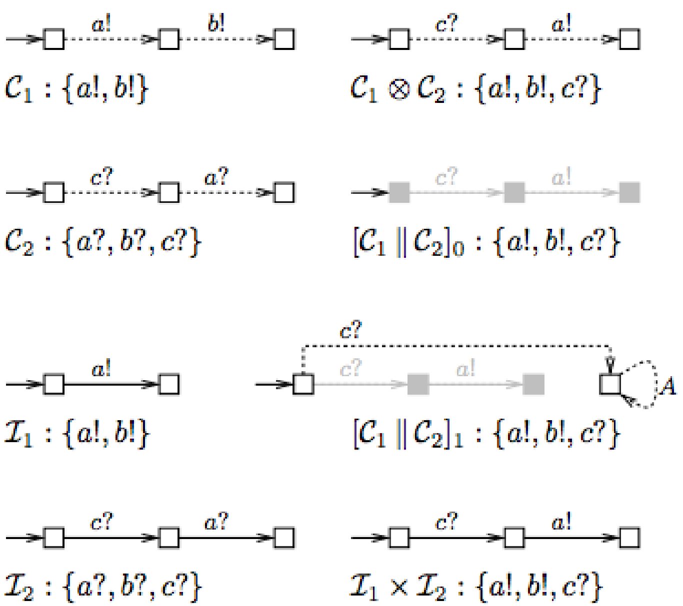
- Compatibility for two modal interfaces, C_1 and C_2.
- Compute the product between C1, C2 by the previous formula
- Define Illegal(C1, C2) to be the subset of words u s.t. there exists either
  - An action that is an output of P1 and an input of P2 with $a \in may_1(u_1) \setminus must_2(u_2)$
  - Or an action that is an output of P2 and an input of P1 with $a \in may_2(u_2) \setminus must_1(u_1)$

The composition by Larsen et al.

- Follow backward pruning defined for interface automata to remove illegal states.
- Two interfaces C1 and C2 are compatible, denoted C1||C2,
  - if the pruning does not remove the empty word.

Counterexample to Thm. 10 by Larsen et al.

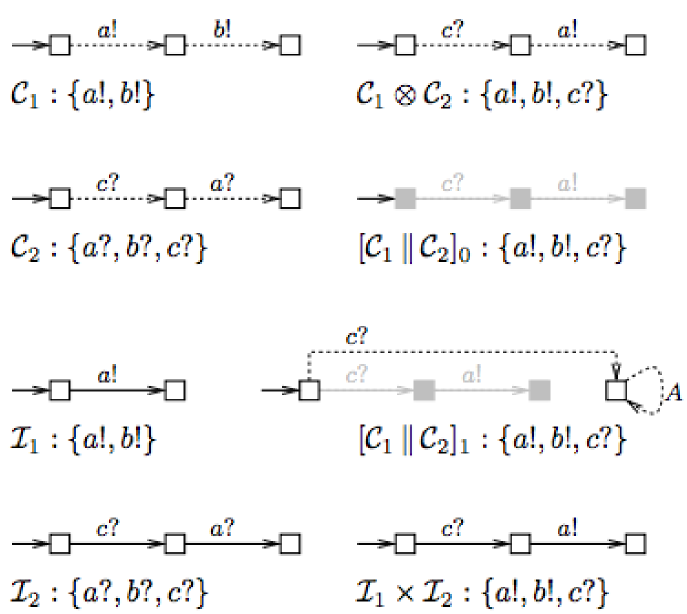
"*(Independent Implementability). For any two composable modal interfaces* $\mathcal{C}_1$, $\mathcal{C}_2$ *and two implementations* $(\mathcal{I}_1, \pi_1)$ *and* $(\mathcal{I}_2, \pi_2)$. *If* $(\mathcal{I}_1, \pi_1) \leq \mathcal{C}_1$ *and* $(\mathcal{I}_2, \pi_2) \leq \mathcal{C}_2$, *then it holds that* $(\mathcal{I}_1, \pi_1) \times (\mathcal{I}_2, \pi_2) \leq \mathcal{C}_1 \,||\, \mathcal{C}_2$."



Word c?.a! is illegal in the composition, because for a!, C1 may offer b!, but C2 does not accept it. c?.a! is, however in the product of the two implementations.

I1xI2 does not refine C1||C2. Thm 10 is wrong.

Correction



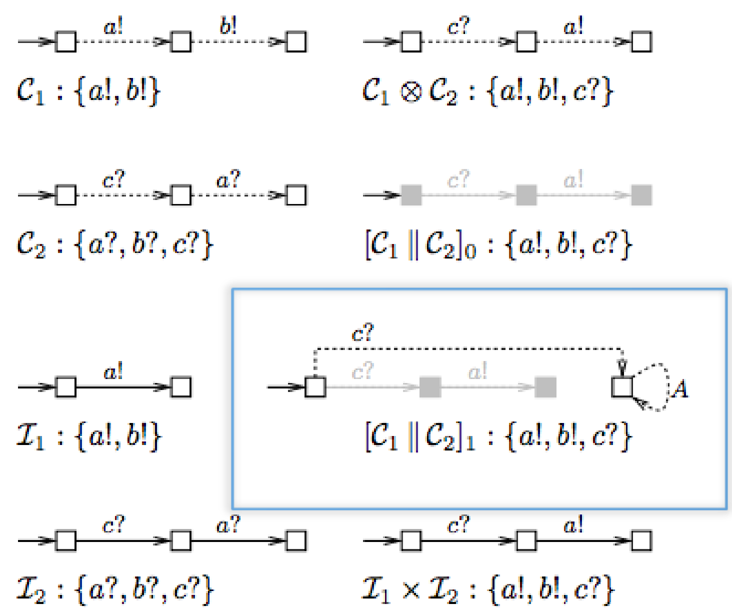
If the environment has no strategy to prevent the occurrence of an illegal word, call this an **exception**.

- Exception language of modal interfaces C1 and C2 is:
  - $pre_!^*$(Illegal(C1, C2))
- C1||C2 iff the empty word is not an exception.
- || is commutative and associative

Parallel Composition

DEFINITION 12 (PARALLEL COMPOSITION). *Given two modal interfaces $\mathcal{C}_1$ and $\mathcal{C}_2$, the* relaxation *of $\mathcal{C}_1 \otimes \mathcal{C}_2$ is obtained by applying the following pseudo-algorithm to $\mathcal{C}_1 \otimes \mathcal{C}_2$:*

for all v **in** $\mathcal{L}_{\mathcal{C}_1 \otimes \mathcal{C}_2}$ **do**
    for all a **in** A **do**
        if $v \notin \mathcal{E}_{\mathcal{C}_1 \parallel \mathcal{C}_2}$ **and** $v.a \in \mathcal{E}_{\mathcal{C}_1 \parallel \mathcal{C}_2}$ **then**
            for all w **in** A^* **do**
                $must(v.a.w) := \emptyset$
                $may(v.a.w) := A$
            end for
        end if
    end for
end for



If illegal words exists for certain pairs of implementations, the system is taken to a universal state: nothing is forbidden, nothing is mandatory (for all actions)

Independent Implementability

THEOREM 15 (INDEPENDENT IMPLEMENTABILITY). *For any two modal interfaces $\mathcal{C}_1$, $\mathcal{C}_2$ and two implementations $(\mathcal{I}_1, \pi_1)$, $(\mathcal{I}_2, \pi_2)$ such that $(\mathcal{I}_1, \pi_1) \models_s \mathcal{C}_1$ and $(\mathcal{I}_2, \pi_2) \models_s \mathcal{C}_2$, it holds that $(\mathcal{I}_1, \pi_1) \times (\mathcal{I}_2, \pi_2) \models_s \mathcal{C}_1 \| \mathcal{C}_2$.*

Proof: *If $(\mathcal{I}_1, \pi_1) \models_s \mathcal{C}_1$ and $(\mathcal{I}_2, \pi_2) \models_s \mathcal{C}_2$, then, by Theorem 10, $(\mathcal{I}_1, \pi_1) \times (\mathcal{I}_2, \pi_2) \models_s \mathcal{C}_1 \otimes \mathcal{C}_2$.*
By the previous lemma and by the generalization of Theorem 1 in Theorem 10: $(\mathcal{I}_1, \pi_1) \times (\mathcal{I}_2, \pi_2) \models_s \mathcal{C}_1 \| \mathcal{C}_2$. □

Conclusions

- Modal interface: unification of interface automata and modal specifications
- Core contribution: || operator that is an optimistic composition rule for interfaces
- Vague use cases and applications
- Missing empirical comparison
- Future work
  - Implementation
  - Timed extension of modal interfaces