Towards an I/O Conformance Testing Theory for Software Product Lines based on Modal Interface Automata

Lars Luthmann* Stephan Mennicke* Malte Lochau†

Institute for Programming and Reactive Systems
TU Braunschweig, Germany

Realtime Systems Lab
TU Darmstadt, Germany

l.luthmann@tu-bs.de mennicke@ips.cs.tu-bs.de malte.lochau@es.tu-darmstadt.de

We present an adaptation of input/output conformance (ioco) testing principles to families of similar implementation variants as appearing in product line engineering. Our proposed product line testing theory relies on Modal Interface Automata (MIA) as behavioral specification formalism. MIA enrich I/O-labeled transition systems with may/must modalities to distinguish mandatory from optional behavior, thus providing a semantic notion of intrinsic behavioral variability. In particular, MIA constitute a restricted, yet fully expressive subclass of I/O-labeled modal transition systems, guaranteeing desirable refinement and compositionality properties. The resulting modal-ioco relation defined on MIA is preserved under MIA refinement, which serves as variant derivation mechanism in our product line testing theory. As a result, modal-ioco is proven correct in the sense that it coincides with traditional ioco to hold for every derivable implementation variant. Based on this result, a family-based product line conformance testing framework can be established.

1 Introduction

Modal transition systems (MTS) constitute an extension to (labeled) transition systems (LTS) by enriching the transition relation with a *may*/*must* dichotomy [13, 11]. This way, behavioral system specifications based on MTS leave open implementation freedom by distinguishing *mandatory* from *optional* behaviors, thus imposing a rigorous notion of (semantic) *refinement* [2]. Considering Input/Output-labeled MTS in particular, they provide a suitable foundation for interface specifications of component-based systems [17]. MTS incorporate a natural notion of interface *compatibility* and, thereupon, enjoy desirable compositionality properties, being imperative for a comprehensive *interface theory* [18, 4].

Based on the work of Fischbein et al. in [10], Larsen et al. propose in [12] to use modal specifications as a basis for a *behavioral variability theory* for software product lines [9]. A product line, therefore, comprises a family of well-defined *implementation variants* derivable from modal specifications using modal refinement, where the validity of a variant is further restricted due to its compatibility with other components and/or a given environmental specification. Based on this compact representation of families of implementation variants, a verification theory for product lines has been developed in [3], combining MTS with deontic logics to further restrict variable behaviors. Those approaches allow for model-checking temporal properties on entire families of implementation variants without explicitly considering every particular variant, which is referred to as *family-based* product line analysis [19].

However, besides those appealing family-based product line verification approaches, the applicability of modal specifications as a formal foundation for a family-based product line testing theory has not been intensively considered so far. In particular, the input/output conformance testing theory, initially

*This work was partially supported by the DFG (German Research Foundation), grant GO-671/6-2.

†This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution.

J. Atlee and S. Gnesi (Eds.): FMSPLE 2015
EPTCS 182, 2015, pp. 1–13, doi:10.4204/EPTCS.182.1

introduced by Tretmans in [20], is one of the most established formal frameworks to reason about fundamental properties of (model-based) functional testing approaches. For this purpose, the **ioco** relation imposes a notion of observational equivalence [16] between a test model specification, given as an I/O-labeled transition system, and a (black box) implementation under test, requiring the implementation behaviors to conform to the specified behaviors.

To the best of our knowledge, there currently exist two approaches adapting I/O-conformance testing principles to product lines. In Beohar and Mousavi [6], featured transition systems (FTS), initially proposed by Classen et al. [8], are equipped with I/O labels to enrich the **ioco** relation with explicit feature constraints. This way, a family-based I/O-conformance testing framework can be established, based on constraint-solving capabilities as used in [7] for product line model checking. In contrast, the approach proposed in [14], called **mioco**, adapts the key concepts of **ioco** to modal product line specifications where I/O-labeled MTS (IOMTS) are used as specifications of product lines under test. A corresponding family-based I/O-conformance testing theory can be built upon the notions of modal refinement and composition [12]. In this paper, we present an improved elaboration of this initial approach to serve as a sound basis for family-based product line conformance testing. In particular, we make the following contributions.

- We consider a novel class of I/O-labeled modal transition systems, i. e., Modal Interface Automata (MIA) [15], instead of IOMTS. MIAs slightly restrict IOMTS to guarantee desirable refinement and compositionality properties.
- We define the conformance relation $\mathbf{mioco}_{\mathsf{MIA}}$ to relate product line implementations to product line specifications both given as MIAs. Thereupon, we clarify the assumptions to hold for specification and implementation in the spirit of classical **ioco**, e. g., concerning input-enabledness and different concepts for input completions. One major challenge is to guarantee a proper treatment of the two kinds of implementation freedom apparent in product line specifications, namely *variable* and *unspecified* behaviors.
- In addition to the basic result in [14] ensuring preservation of **mioco** on IOMTS under refinement, we obtain strong results for our novel conformance relation $\mathbf{mioco}_{\mathsf{MIA}}$, concerning soundness and completeness. Therefore, $\mathbf{mioco}_{\mathsf{MIA}}$ reflects the essence of family-based product line analysis by means of I/O-conformance testing [19].

Here, we focus our considerations to testing single components, and, therefore, also omitting τ transitions within modal specifications. However, the results obtained in this paper pave the way to a compositional and family-based product line testing theory.

The remainder of this paper is structured as follows. Sect. 2 provides a brief repetition of input/output conformance defined on I/O-labeled transition systems, as well as the foundations of modal transition systems. In Sect. 3, we introduce MIA as a new model for product line specifications and describe variant derivation semantics in terms of MIA refinement. In Sect. 4, we propose an adaptation of input/output conformance notions to MIA and define approaches for achieving input-enabledness via completions. Our main result concerning the correctness of modal-ioco on MIA are formulated and proven in Sect. 5. Sect. 6 concludes with an outlook on our ongoing and future research directions.

2 Preliminaries

We start with an overview on I/O-labeled transition systems and I/O conformance testing. Furthermore, we present modal transition systems (MTS) laying the foundation for modal interface automata.

Labeled transition systems (LTS) constitute a well-established model for discrete-state reactive system behaviors. The behavior of an LTS is specified by means of a labeled transition relation $\longrightarrow \subseteq Q \times act \times Q$ on a set of states Q and an alphabet of actions act. To serve as a test model specification for input/output conformance testing, the subclass of *I/O labeled transition systems* is considered, dividing the set act into disjoint subsets of controllable *input* actions I and observable *output* actions O. In addition, *internal* actions are usually summarized under the special symbol $\tau \notin (I \cup O)$. However, we do not consider τ transitions in this paper.

Definition 1 (I/O Labeled Transition System)**.** *An* I/O labeled transition system *(IOLTS) is a tuple* $(Q, I, O, \longrightarrow)$*, where*

- Q *is a countable set of states,*
- I *and* O *are disjoint sets of input actions and output actions, respectively, and*
- $\longrightarrow \subseteq Q \times (I \cup O) \times Q$ *is a labeled transition relation.*

Note that (IO)LTS usually do not comprise a predefined initial state, as it is either identified with its entire set of states Q, or some state $q \in Q$ denoting the initial state. By $q \xrightarrow{a} q'$ we mean that $(q, a, q') \in \longrightarrow$ holds, and we write $q \xrightarrow{a}$ as a short hand for $\exists q' \in Q : q \xrightarrow{a} q'$. We further denote a *path* $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n$ by $q_0 \xrightarrow{\sigma} q_n$, where $\sigma = a_1 a_2 \ldots a_n \in (I \cup O)^*$ is called a *trace*.

In the input/output conformance relation (cf. Sect. 2.1), an implementation, represented as an I/O-labeled transition system, is assumed to be *input-enabled*, i. e., to never reject any inputs. This yields the subclass of *I/O transition systems*.

Definition 2 (I/O Transition System)**.** *A state* $q \in Q$ *of an IOLTS* Q *is* input-enabled *iff for all* $i \in I$*, there exists a state* $q' \in Q$ *such that* $q \xrightarrow{i} q'$*.* Q *is an* I/O Transition System *(IOTS) iff all* $q \in Q$ *are input-enabled.*

In deviation from Tretmans [20], we employ strong input-enabledness, as we do not consider internal behavior. Figure 1 shows three sample LTS, modeling different variants of a vending machine. All of these vending machines have in common that they accept money in the initial state (the topmost state) and are capable of dispensing tea. However, some of them may also dispense coffee and notify the user about errors. Transitions are labeled with either input labels (prefix ?), or output labels (prefix !). The LTS in Figure 1b accepts 1€ or 2€ coins from customers. After inserting 2€, change is returned and the customer is allowed to choose coffee or tea, or to refill cups. Next, the vending machine dispenses the selected beverage. The LTS in Figure 1a is an IOTS, as every state accepts all possible inputs, i. e., *1€*, *2€*, *cups*, and *tea*. Label I denotes that a transition exists for each input symbol, unless a state already accepts an input.

2.1 Input/Output Conformance

An implementation i, given as an IOTS, I/O-conforms to a specification s, given as an IOLTS, if all observable output symbols of i after any possible input sequence σ of s are permitted by s. That means that a system specification states the allowed output behavior. For this to hold, the set $Out(P)$ of output actions enabled in any possible state $p \in P$ of i reachable via a sequence σ, denoted by $P = i \,\mathbf{after}\, \sigma$, must be included in the corresponding set $Out(Q)$ with $Q = s \,\mathbf{after}\, \sigma$. To rule out trivial implementations never showing any outputs, the concept of *quiescence* is introduced by means of an observable action δ to explicitly permit the absence (suspension) of any output in a state. The definitions of this section follow [20].

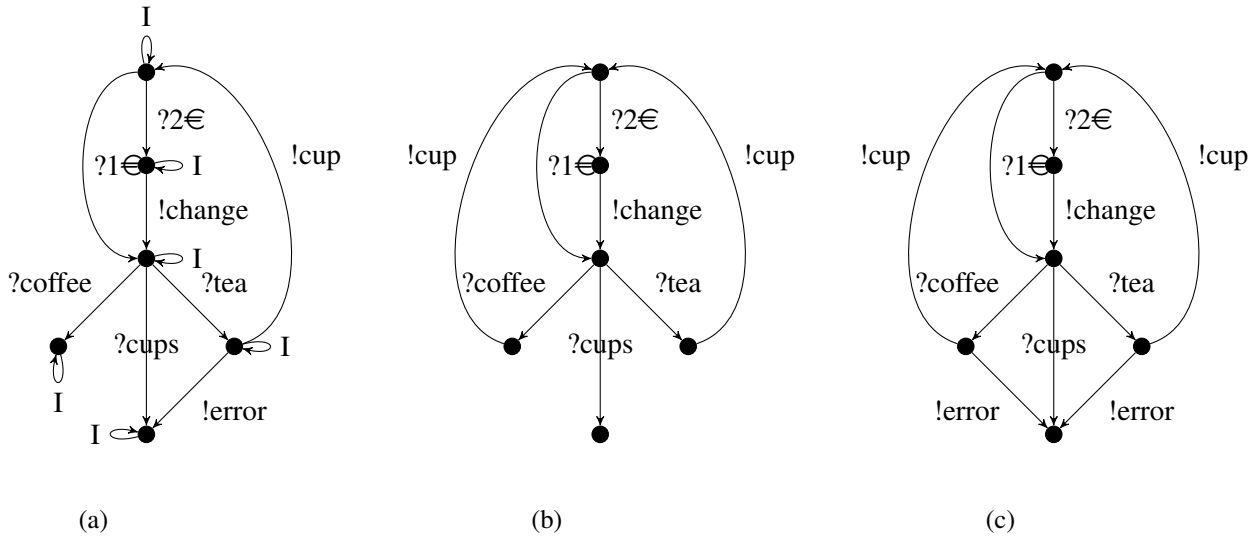


Figure 1: Sample LTS of a simple vending machine, adapted from [14].

Definition 3. *Let Q be an IOLTS, $p \in Q$, $P \subseteq Q$, and $\sigma \in (I \cup O \cup \{\delta\})^*$.*

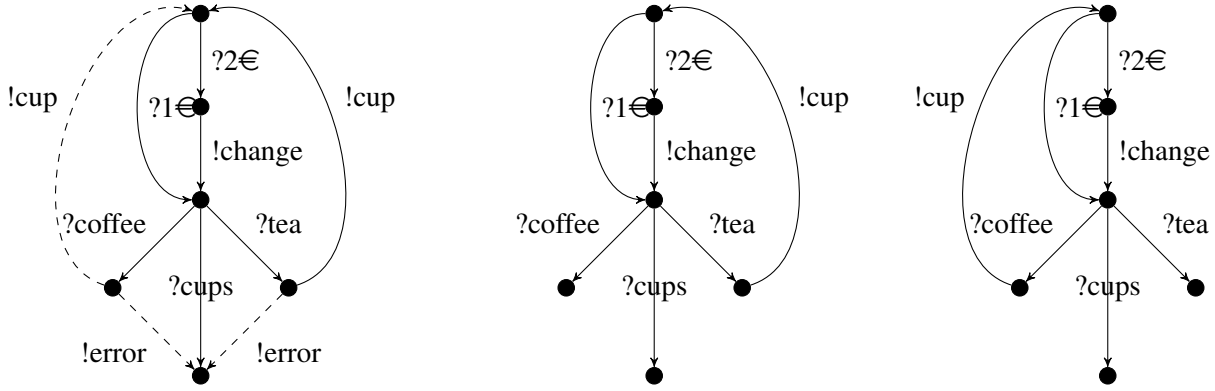
- $init(p) := \{\mu \in (I \cup O) \mid p \xrightarrow{\mu}\}$,
- *p is* quiescent*, denoted by $\delta(p)$, iff $init(p) \subseteq I$,*
- p **after** $\sigma := \{q \in Q \mid p \xrightarrow{\sigma} q\}$,
- $Out(P) := \{\mu \in O \mid \exists p \in P : p \xrightarrow{\mu}\} \cup \{\delta \mid \exists p \in P : \delta(p)\}$, *and*
- $Straces(p) := \{\sigma \in (I \cup O \cup \{\delta\})^* \mid p \xrightarrow{\sigma}\}$, *where $q \xrightarrow{\delta} q$ iff $\delta(q)$.*

I/O conformance requires any reaction of an implementation i to every possible environmental behavior σ to be checked against those of its specification s, even if no proper reaction for σ is actually specified by s. Hence, conformance testing is usually limited to positive testing, i. e., only considering behaviors being explicitly specified in s, i. e., contained in the *suspension traces* ($Straces(s)$) of s.

Definition 4 (Input/Output Conformance)**.** *Let s be an IOLTS and i an IOTS with the same sets of inputs and outputs. i* **ioco** *$s :\Leftrightarrow \forall \sigma \in Straces(s) : Out(i$* **after** *$\sigma) \subseteq Out(s$* **after** *$\sigma)$.*

Assuming the IOLTS of Figure 1c to be a specification s and the IOTS of Figure 1a to be an implementation i, then i **ioco** s holds, as i does not show any unspecified output behavior. However, considering the IOLTS of Figure 1b as a specification s for i, then i **ioco** s does not hold as i exhibits output behavior *error*, violating conformance of i to s.

The **ioco** relation permits implementation freedom as only one specified output behavior must be implemented. In addition, if there are unspecified inputs for state q in the specification s, then an implementation may react with arbitrary outputs to those unspecified inputs, as those behaviors do not occur in the suspension traces of s and are, therefore, never tested. However, for product lines, we further need the possibility to (1) explicitly express variability within specifications and, therefore, to (2) distinguish mandatory from optional behavior, which leads us to Modal Transition Systems (MTS).



(a) Specification (b) Correct implementation (c) Incorrect implementation

Figure 2: Figure 2a shows an MTS combining all systems from Figure 1. Figures 2b and 2c show a correct and an incorrent implementation regarding $\mathbf{mioco}_{\mathsf{MIA}}$ (see Section 4).

2.2 Modal Transition Systems

To specify behavioral variability of product lines, we use Modal Transition Systems (MTS) according to Larsen [11, 12] as a basis. MTS are based on LTS but distinguish between so called must and may transitions, specifying mandatory behavior as well as allowed behavior, respectively. By definition any must transition is a may transition as any mandatory behavior must also be allowed. Therefore, only may transitions not underlying must transitions denote optional behavior. Accordingly, we call may transitions that are not must transitions *optional* and must transitions *mandatory*. Additionally, absent transitions denote forbidden behavior.

Definition 5 (Modal Transition System)**.** *A tuple* $Q = (Q, A, \longrightarrow_{\Box}, \longrightarrow_{\Diamond})$ *is a* Modal Transition System *(MTS) iff*

- Q *is a finite set of states,*
- A *is a set of actions,*
- $\longrightarrow_{\Box} \subseteq Q \times A \times Q$ *is the labeled* must transition *relation,*
- $\longrightarrow_{\Diamond} \subseteq Q \times A \times Q$ *is the labeled* may transition *relation, and*

Q *is* syntactically consistent*, i. e.,* $q \xrightarrow{a}_{\Box} q'$ *implies* $q \xrightarrow{a}_{\Diamond} q'$.

Note that, in our setting, we assign $I \cup O$ to A which defines *I/O-labeled MTS*. MTS allow us to superimpose several systems into one larger system, from which the original systems are derivable via modal refinement. Therefore, modal *refinement* preserves mandatory and forbidden behavior, whereas optional behavior may turn into either mandatory or forbidden behavior. Figure 2a shows a sample MTS. Therein, solid edges depict mandatory behavior and dashed edges depict optional behavior. The MTS in Figure 2a combines all IOLTS from Figure 1 into one. This is achieved by making the behaviors being common to all IOLTS variants mandatory behaviors, whereas the variable behaviors become optional in the MTS. Larsen et al. [12] also defined input-enabledness for MTS and an according input-enabled MTS version to be called *I/O modal transition systems* (IOMTS). However, there are two flavors of input-enabledness: *must-input-enabledness* and *may-input-enabledness*, both being defined as canonical

extensions of Def. 2. A may-input-enabled MTS is called *I/O-labeled MTS* (IOMTS). For an overview on IOMTS and the according I/O-conformance relation, we refer to [14]. Another option for adding modalities to system specifications are *Modal Interface Automata*, being a restricted subclass of IOMTS.

3 Modal Interface Automata

The model we employ in this paper is called *Modal Interface Automata* (MIA) which are basically input-deterministic I/O-labeled MTS. Input-determinism is a desirable property in model-based testing, as it makes testing procedures manageable by ensuring that some inputs are not infinitely often neglected during test scenarios, as imposed by non-deterministic inputs. Furthermore, specified inputs are always mandatory, but unspecified inputs are implicitly allowed. This restriction yields refinement and composition properties beneficial for both modeling product line specifications and implementations with behavioral variability, as well as modal I/O-conformance testing as described in Sect. 4. The MTS depicted in Figure 2 are in fact MIAs, as they exhibit input-determinism and every input transition is mandatory.

Definition 6 (Modal Interface Automaton)**.** *A tuple $Q = (Q, I, O, \longrightarrow_\Box, \longrightarrow_\Diamond)$ is a* Modal Interface Automaton *(MIA), where $(Q, I \cup O, \longrightarrow_\Box, \longrightarrow_\Diamond)$ is an MTS with disjoint alphabets* I*,* O *and for all $i \in I$:*

- *$q \xrightarrow{i}_\Box q'$ and $q \xrightarrow{i}_\Box q''$ implies $q' = q''$ (i. e., we require input-determinism),*
- *$q \xrightarrow{i}_\Diamond q'$ implies $q \xrightarrow{i}_\Box q'$ (i. e., all inputs are mandatory behavior).*

In deviation to Lüttgen and Vogler [15], we do not employ *disjunctive* MTS, as they are not needed for our purposes. Furthermore, we limit our considerations to MIAs without internal behaviors, i. e., τ transitions, which is no limitation, as all our results remain valid for MIAs with internal behavior. For future work, we plan to investigate our testing theory under parallel composition for which a treatment of internal transitions is inevitable. Lüttgen and Vogler define an operator for parallel composition of MIA similar to interface automata [1]. They identify *error states* arising from the composition of incompatible states, and remove them, as well as all states from which reaching some error state is no more preventable by environmental inputs. This is similar to the operator by Larsen et al. [12], but, in contrast to IOMTS, composability of MIA is based on the compatible component semantics rather than syntactic criteria.

Each input transition of a MIA is, by definition, mandatory. However, this does not limit the expressiveness of MIA compared to IOMTS, as input transitions are always implicitly allowed by *modal refinement*. Modal refinement is a crucial notion in modal theories, as they constitute an implementation relation that preserves mandatory behaviors, but also leaves implementation freedom concerning optional and unspecified behaviors. Intuitively, a MIA p refines q if (1) the optional output behavior of p is simulated by q, and (2) all mandatory behavior of q is simulated by p, thus imposing an *alternating simulation* [15].

Definition 7 (MIA Refinement)**.** *Let P, Q be MIAs over I and O. A relation $\mathscr{R} \subseteq P \times Q$ is a* MIA-refinement *iff for all $(p,q) \in \mathscr{R}$:*

1. *$q \xrightarrow{a}_\Box q'$ where $a \in I \cup O$ implies $\exists p' : p \xrightarrow{a}_\Box p'$ and $(p', q') \in \mathscr{R}$,*
2. *$p \xrightarrow{\alpha}_\Diamond p'$ where $\alpha \in O$ implies $\exists q' : q \xrightarrow{\alpha}_\Diamond q'$ and $(p', q') \in \mathscr{R}$.*

If there is a MIA-refinement $\mathscr{R}$ such that $(p,q) \in \mathscr{R}$, then p MIA-refines *q, denoted by $p \sqsubseteq_{\mathsf{MIA}} q$.*

The most desirable property for composition operators in modal system theory is their preservation of modal refinement. Lüttgen and Vogler show this to hold for parallel MIA composition, and also for

conjunction and *disjunction* of MIAs [15]. In contrast to MTS, not all unspecified transitions of MIAs refer to forbidden behavior, but only those being outputs. Input transitions are always implicitly allowed and, therefore, in Def. 7, only optional outputs of the refined MIA must be simulated by the unrefined one.

In this paper, we interpret MIAs with mandatory and optional behaviors as families of similar system variants. In this regard, the refinement notion serves *variant derivation* such that in $p \sqsubseteq_{\mathsf{MIA}} q$, p represents a variant of q, where a (partially) refined p may still contain optional behavior. Furthermore, as unspecified inputs are implicitly allowed under MIA-refinement, p may also contain additional behaviors, which is not feasible for a product line specification. In order to obtain a sound variant derivation mechanism, we require it to finally yield an IOLTS, which does not incorporate optional behavior. Hence, this IOLTS variant p refines a MIA q, but is restricted to behaviors allowed in q.

Definition 8 (Variant Derivation)**.** *Let $(P,I,O,\longrightarrow)$ be an IOLTS and Q be a MIA over I and O. $p \in P$ is a* variant of *$q \in Q$, denoted by $p \sqsubseteq_{\mathsf{var}} q$, iff (1) $p \sqsubseteq_{\mathsf{MIA}} q$, i. e., there is a MIA-refinement $\mathscr{R}$ between $(P,I,O,\longrightarrow,\longrightarrow)$ and Q such that $(p,q) \in \mathscr{R}$ and (2) $\longrightarrow \subseteq \longrightarrow_{\Diamond}$.*

Thus, a variant derivation is a special kind of MIA-refinement, ensuring that every optional transition of the specification is either removed from, or definitely included in the variant. There is a close relationship between traces of variants $p \sqsubseteq_{\mathsf{var}} q$ and may-traces of q.

Lemma 1. *Let P be an IOLTS with $p \in P$ and Q a MIA with $q \in Q$. If $p \sqsubseteq_{\mathsf{var}} q$, then for each $w \in (I \cup O)^*$ with $p \xrightarrow{w}$ it holds that $q \xrightarrow{w}_{\Diamond}$.*

Based on MIA refinement and MIA variant derivation mechanism, we define a modal version of **ioco**.

4 I/O Conformance Testing for MIAs

In Sect. 2.1, we have already discussed I/O conformance on IOLTS, allowing a certain degree of variability in implementations. However, we propose to apply modal specifications to explicitly capture variability also within specifications, as being inherent to SPLs. We now define the foundations of modal input/output conformance [14] and introduce completion strategies for constructing input-enabled modal interface automata.

4.1 Modal Input/Output Conformance

Intuitively, a modal implementation conforms to a modal specification if it does not exceed the allowed outputs (may) and preserves all mandatory outputs (must). We adapt the notion on I/O conformance to the MIA-framework, accordingly.

Definition 9. *Let Q be a MIA over I and O, $p \in Q$, $P \subseteq Q$, $\sigma \in (I \cup O \cup \{\delta_{\Box}, \delta_{\Diamond}\})^*$, and $\gamma \in \{\Box, \Diamond\}$.*

- $init_{\gamma}(p) := \{\mu \in (I \cup O) \mid p \xrightarrow{\mu}_{\gamma}\}$,
- *p is* may-quiescent*, denoted by $\delta_{\Diamond}(p)$, iff $init_{\Box}(p) \subseteq I$ and p is* must-quiescent*, denoted by $\delta_{\Box}(p)$, iff $init_{\Diamond}(p) \subseteq I$,*
- p **after**$_{\gamma}\,\sigma := \{q \in Q \mid p \xrightarrow{\sigma}_{\gamma} q\}$,
- $Out_{\gamma}(P) := \{\mu \in O \mid \exists p \in P : p \xrightarrow{\mu}_{\gamma}\} \cup \{\delta_{\gamma} \mid \exists p \in P : \delta_{\gamma}(p)\}$, *and*
- $Straces_{\gamma}(p) := \{\sigma \in (I \cup O \cup \{\delta\})^* \mid p \xrightarrow{\sigma}_{\Diamond}\}$, *where $q \xrightarrow{\delta} q$ iff $\delta_{\gamma}(q)$.*

Due to the property of syntactic consistency of MIAs, similar properties are induced for the notions of Def. 9. For instance, $init_\Diamond(p) \subseteq I$ implies $init_\Box(p) \subseteq I$ as $a \in I$, $q \xrightarrow{a}_\Diamond q'$ implies $q \xrightarrow{a}_\Box q'$ and, therefore, must-quiescence of any state p implies may-quiescence of p.

Proposition 1. *Let Q be a MIA over I and O, $p \in Q$, $P \subseteq Q$, and $\sigma \in (I \cup O \cup \{\delta_\Diamond, \delta_\Box\})^*$. Then the following statements hold.*

1. $init_\Box(p) \subseteq init_\Diamond(p)$,
2. *if* $\delta_\Box(p)$, *then* $\delta_\Diamond(p)$,
3. p **after**$_\Box$ $\sigma \subseteq p$ **after**$_\Diamond$ σ,
4. $Out_\Box(P) \subseteq Out_\Diamond(P)$, *and*
5. $Straces_\Box(p) \subseteq Straces_\Diamond(p)$.

Proofs follow from Def. 6 and Def. 9. Due to the required input-determinism, MIAs may only show non-determinstic behavior due to conflicting output transitions. Thus, when considering a trace w of a state q, it holds that $|q$ **after**$_\Diamond$ $w| \geq 1$. Considering MIA-refinement $p \sqsubseteq_{\mathsf{MIA}} q$, each state in p **after**$_\Diamond$ w relates to some state in q **after**$_\Diamond$ w. Conversely, the same holds for states of p **after**$_\Box$ w and q **after**$_\Box$ w.

Lemma 2. *Let p, q be MIAs such that $p \sqsubseteq_{\mathsf{MIA}} q$.*

1. $\forall \sigma \in (I \cup O \cup \{\delta_\Diamond\})^* : q$ **after**$_\Diamond$ $\sigma \neq \emptyset \Rightarrow \forall p' \in p$ **after**$_\Diamond$ $\sigma : \exists q' \in q$ **after**$_\Diamond$ $\sigma : p' \sqsubseteq_{\mathsf{MIA}} q'$
2. $\forall \sigma \in (I \cup O \cup \{\delta_\Box\})^* : p$ **after**$_\Box$ $\sigma \neq \emptyset \Rightarrow \forall q' \in q$ **after**$_\Box$ $\sigma : \exists p' \in p$ **after**$_\Box$ $\sigma : p' \sqsubseteq_{\mathsf{MIA}} q'$

This property is very useful when arguing on paths of MIAs related under MIA-refinement (cf. Theorem 2). While for MTS we distinguished may from must input-enabledness, there is no difference between both in case of MIA, as inputs in MIAs are mandatory.

Definition 10 (Input-Enabledness for MIA)**.** *A MIA Q is* input-enabled *iff for all $q \in Q$ and for all $i \in I$, it holds that $q \xrightarrow{i}_\Box$.*

Henceforth, we require product line implementations i to be given as input-enabled MIAs in order to meet the assumptions originally made by **ioco** that implementations do not deadlock while processing inputs not being serviced by the implementation. Input-enabledness of MIA is preserved under MIA-refinement.

Lemma 3. *Let q, r be MIAs over I and O such that $r \sqsubseteq_{\mathsf{MIA}} q$. If q is input-enabled, then r is input-enabled.*

This also holds for variant p derived from q. In **ioco**, no distinction is made between specified mandatory and optional behavior. A first conformance relation supporting optional behaviors is **mior** [14]. It holds that i **mior** s in case of trace inclusion of may-suspension-traces as well as must-suspension-traces, respectively. However, if we interpret the set of must-behaviors specified by s as the product line core behavior incorporated by all variants, this notion of conformance fails to fully capture this intuition. Suspension trace inclusion solely ensures *some* behaviors of the specified behaviors to be actually implemented (if any), but it does not differentiate within the set of allowed behaviors between mandatory and optional ones.

Figure 2 illustrates the weakness of **mior**. Assuming Figure 2a as specification and the other two Figures to be implementations, then both implementations are correct under **mior**. For Figure 2b, this is obvious as only optional behavior is left out. The problem of **mior** is depicted in the implementation of Figure 2c. Therein, the mandatory behavior outputting the *cup* after the input *tea* is left out but the **mior** relation still holds as no behavior is added. This contradicts the intention of mandatory behaviors as core

behaviors of all product line variants. To overcome this drawback, consider an alternative definition for I/O conformance, denoted by $\mathbf{mior}_{\leq}$ [14], being closer to the very essence of modal refinement requiring alternating suspension trace inclusions. The $\mathbf{mior}_{\leq}$ relation requires implementation i to show at least all mandatory behaviors and at most the allowed behaviors of a specification s. Following this idea, the respective modal version of **ioco** is defined as follows.

Definition 11 (Modal Input/Output Conformance)**.** *Let s, i be MIAs over I, O and i being input-enabled. $i\,\mathbf{mioco}_{\mathsf{MIA}}\,s$ iff*

1. $\forall \sigma \in Straces_{\Diamond}(s) : Out_{\Diamond}(i\ \mathbf{after}_{\Diamond}\ \sigma) \subseteq Out_{\Diamond}(s\ \mathbf{after}_{\Diamond}\ \sigma)$*, and*
2. $\forall \sigma \in Straces_{\Box}(i) : Out_{\Box}(s\ \mathbf{after}_{\Box}\ \sigma) \subseteq Out_{\Box}(i\ \mathbf{after}_{\Box}\ \sigma)$.

In the first part of checking $i\,\mathbf{mioco}_{\mathsf{MIA}}\,s$, we consider all specified suspension-traces and essentially check $i\,\mathbf{ioco}\,s$. In the second part, we only consider must-suspension-traces of i and essentially check $s\,\mathbf{ioco}\,i$. That way, we make sure that the implementation does not add forbidden behavior or ignores mandatory behavior. Requiring input-enabledness for specifications of i is infeasible for realistic test modeling approaches. However, an artificial input-enabledness for incomplete specifications of i may always be achieved by *completions* of i (cf. Sect. 4.2).

Let us reconcile Figure 2 with the $\mathbf{mioco}_{\mathsf{MIA}}$ relation instead of **mior**. Again, Figure 2a depicts the specification and the other two figures represent the implementations. The implementation in Figure 2b is still correct, whereas that in Figure 2c discards the mandatory action *!cup* after the action *?tea*, thus being incorrect regarding $\mathbf{mioco}_{\mathsf{MIA}}$.

MIA-refinement is considered in two ways, first as an implementation relation ($\sqsubseteq_{\mathsf{MIA}}$) and second as a relation for variant derivation ($\sqsubseteq_{\mathsf{var}}$). For $\mathbf{mioco}_{\mathsf{MIA}}$ to yield a family-based conformance testing relation, it should be preserved by $\sqsubseteq_{\mathsf{var}}$, i. e., if $i\,\mathbf{mioco}_{\mathsf{MIA}}\,s$ is checked for a product line implementation i and its specification s, then this check can be neglected for the variants derivable from i. Due to the fact that implementations are input-enabled, $\mathbf{mioco}_{\mathsf{MIA}}$ is also preserved by $\sqsubseteq_{\mathsf{MIA}}$.

Proposition 2 (MIA-Refinement preserves $\mathbf{mioco}_{\mathsf{MIA}}$)**.** *Let s, i be MIAs over I and O such that i is input-enabled. If $i\,\mathbf{mioco}_{\mathsf{MIA}}\,s$, then for all $i' \sqsubseteq_{\mathsf{MIA}} i$ it holds that $i'\,\mathbf{mioco}_{\mathsf{MIA}}\,s$.*

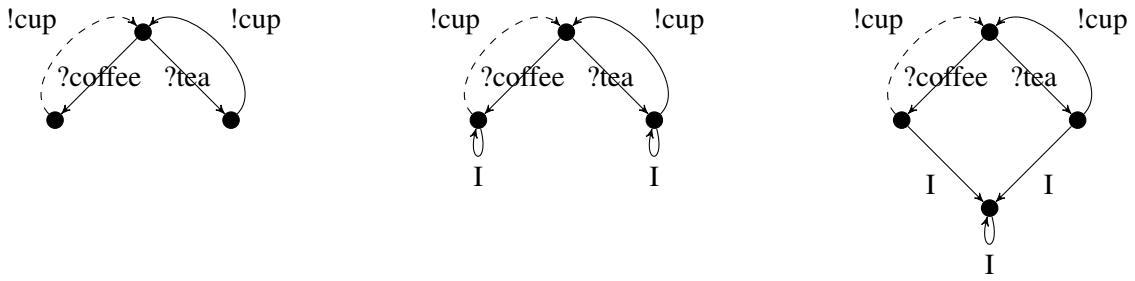
Proof. The fact $i\,\mathbf{mioco}_{\mathsf{MIA}}\,s$ implies that $Out_{\Diamond}(i\ \mathbf{after}_{\Diamond}\ \sigma) \subseteq Out_{\Diamond}(s\ \mathbf{after}_{\Diamond}\ \sigma)$ for all $\sigma \in Straces_{\Diamond}(s)$ holds and $Out_{\Box}(s\ \mathbf{after}_{\Box}\ \sigma) \subseteq Out_{\Box}(i\ \mathbf{after}_{\Box}\ \sigma)$ for all $\sigma \in Straces_{\Box}(i)$ holds as well. Let i' be a MIA such that $i' \sqsubseteq_{\mathsf{MIA}} i$. Due to Lemma 2 and Lemma 3, i' is input-enabled and for all $\sigma \in (I \cup O \cup \{\delta_{\Diamond}\})^*$, it holds that $Out_{\Diamond}(i'\ \mathbf{after}_{\Diamond}\ \sigma) \subseteq Out_{\Diamond}(i\ \mathbf{after}_{\Diamond}\ \sigma)$. By transitivity of $\subseteq$, $Out_{\Diamond}(i'\ \mathbf{after}_{\Diamond}\ \sigma) \subseteq Out_{\Diamond}(s\ \mathbf{after}_{\Diamond}\ \sigma)$ holds for all $\sigma \in Straces_{\Diamond}(s) \subseteq (I \cup O \cup \{\delta_{\Diamond}\})^*$.

We now prove that also $Out_{\Box}(s\ \mathbf{after}_{\Box}\ \sigma) \subseteq Out_{\Box}(i'\ \mathbf{after}_{\Box}\ \sigma)$ for all $\sigma \in Straces_{\Box}(i')$. As i is input-enabled, it holds that $Out_{\Box}(i\ \mathbf{after}_{\Box}\ \sigma) \subseteq Out_{\Box}(i'\ \mathbf{after}_{\Box}\ \sigma)$ for all $\sigma \in (I \cup O \cup \{\delta_{\Box}\})^*$. Therefore, $Out_{\Box}(s\ \mathbf{after}_{\Box}\ \sigma) \subseteq Out_{\Box}(i'\ \mathbf{after}_{\Box}\ \sigma)$ holds for all $\sigma \in Straces_{\Box}(i') \subseteq (I \cup O \cup \{\delta_{\Box}\})$ by transitivity of $\subseteq$. Thus, $i'\,\mathbf{mioco}_{\mathsf{MIA}}\,s$. □

Next we show how to achieve input-enabledness by so-called *completions*.

4.2 Completions for MIA

In $\mathbf{mioco}_{\mathsf{MIA}}$, we permit states to be underspecified, i. e., we may leave open how a state $q \in Q$ of an implementation behaves in case of action $a \in (I \cup O)$ if $q \not\xrightarrow{a}_{\Diamond}$. Underspecification comes in two flavors: underspecification of input actions and underspecification of output actions. Underspecification of output actions is explicit, i. e., a state can only perform outputs attached to one of its transitions. In contrast, underspecification of input actions is implicit, i. e., a state accepts every possible input of the



(a) Specification (b) Angelic completion (c) Chaotic completion

Figure 3: Specification of a simplified vending machine and two completion strategies, where I denotes one transition for both *?coffee* and *?tea*.

input set (even if there is no dedicated transition). In this section, we present two transformations from underspecified to specified MIA accepting every possible input action, namely *angelic completion* and *chaotic completion*. Both completions are described using the underspecified MIA depicted in Figure 3a. In this MIA with $I = \{coffee,\ tea\}$ the two lower states are underspecified. One possibility for completion, called *angelic* by Vaandrager [21], is to ignore unspecified inputs. An angelically completed automaton MIA_{AC} of a given *MIA* is obtained by adding self-loop transitions to every state $q \in Q$ for every input $i \in I$ not being accepted by the state. In Figure 3b, we added self-loops to the bottom states for input actions *coffee* and *tea*.

Definition 12 (Angelic Completion)**.** *Given a MIA* $(Q, I, O, \longrightarrow_{\square}, \longrightarrow_{\Diamond})$*, its angelic completion* MIA_{AC} *is defined as* $(Q, I, O, \longrightarrow'_{\square}, \longrightarrow'_{\Diamond})$*, where*

- $\longrightarrow'_{\square} = \longrightarrow_{\square} \cup \{(q,i,q) \mid q \in Q,\ i \in I,\ q \not\xrightarrow{i}_{\square}\}$*, and*
- $\longrightarrow'_{\Diamond} = \longrightarrow_{\Diamond} \cup \{(q,i,q) \mid q \in Q,\ i \in I,\ q \not\xrightarrow{i}_{\Diamond}\}$.

Chaotic completion is also based on the work of Vaandrager [21], where the automaton is no more able to do any outputs as soon as an unspecified input occurred. A chaotically completed automaton MIA_{CC} is obtained by adding a fresh error state which is entered whenever an unspecified input actions occurs. In Figure 3c, we added transitions from the states with underspecified input behavior to the error state. Note that the error state is a so-called *sink state*, because once reached, it will never be left.

Definition 13 (Chaotic Completion)**.** *Given a MIA* $(Q, I, O, \longrightarrow_{\square}, \longrightarrow_{\Diamond})$*, its chaotic completion* MIA_{CC} *is defined as* $(Q', I, O, \longrightarrow'_{\square}, \longrightarrow'_{\Diamond})$*, where*

- $Q' = Q \cup \{q_E\}$*, where* $q_E \notin Q$*,*
- $\longrightarrow'_{\square} = \longrightarrow_{\square} \cup \{(q,i,q_E) \mid q \in Q,\ i \in I,\ q \not\xrightarrow{i}_{\square}\} \cup \{(q_E, \lambda, q_E) | \lambda \in I\}$*, and*
- $\longrightarrow'_{\Diamond} = \longrightarrow_{\Diamond} \cup \{(q,i,q_E) \mid q \in Q,\ i \in I,\ q \not\xrightarrow{i}_{\Diamond}\} \cup \{(q_E, \lambda, q_E) | \lambda \in I\}$.

These results complete our discussions on modal testing theory based on MIA. We now consider soundness and completeness notions for $\mathbf{mioco}_{\mathsf{MIA}}$ and give corresponding proofs.

5 Correctness of $\mathbf{mioco}_{\mathsf{MIA}}$

In order to serve as a reliable basis for family-based product line conformance testing, it is necessary for $\mathbf{mioco}_{\mathsf{MIA}}$ to be (1) sound, i. e., whenever $i\,\mathbf{mioco}_{\mathsf{MIA}}\,s$ holds, then each implementation variant derivable

Figure 4: Each variant of i conforms to s (ioco), but i **mioco**$_{\mathsf{MIA}}$ s does not hold.

from i also conforms to a variant of s, and (2) complete, i. e., whenever all variants of i are correct w. r. t. **ioco** and to the product line specification s, then i **mioco**$_{\mathsf{MIA}}$ s holds. In this section, we prove soundness of **mioco**$_{\mathsf{MIA}}$ and discuss under which conditions completeness of **mioco**$_{\mathsf{MIA}}$ may be obtained. Whenever we use **mioco**$_{\mathsf{MIA}}$ to show that a modal implementation i conforms to the modal specification s, for each variant $i' \sqsubseteq_{\mathsf{var}} i$ there is a variant $s' \sqsubseteq_{\mathsf{var}} s$ such that i' **ioco** s' holds. By checking i **mioco**$_{\mathsf{MIA}}$ s once, checking each and every variant of i against some variant of s can be omitted. This is a remarkable improvement compared to variant-by-variant conformance testing, due to the exponentially growing number of variants i' in the number of optional transitions of i.

For soundness, we need to take into account that all considered i' are variants derived from i. By Def. 8, each i' contains at most those transitions being may transitions of i. Therefore, i restricts the set of possible transitions of i' and as i **mioco**$_{\mathsf{MIA}}$ s holds, also s restricts the set of possible transitions of s'. It is sufficient that the output behavior of i' is included in that of s', but not vice versa. We, therefore, choose a single s' for each $i' \sqsubseteq_{\mathsf{var}} i$, the *Family-LTS of s*, denoted by s_{fam}, consisting of all may transitions of s. If Q (q) is a MIA, then Q_{fam} (q_{fam}) is the LTS with $Q_{fam} = Q$ and $\longrightarrow_{fam} = \longrightarrow_{\Diamond}$. As $\longrightarrow_{fam} = \longrightarrow_{\Diamond}$ and, therefore, $\longrightarrow_{fam} \subseteq \longrightarrow_{\Diamond}$, it holds that $q_{fam} \sqsubseteq_{\mathsf{var}} q$ for every MIA q. Using $s' = s_{fam}$ enables us to prove soundness.

Theorem 1 (Soundness)**.** *Let s and i be MIAs such that i is input-enabled. If i* **mioco**$_{\mathsf{MIA}}$ *s, then for all $i' \sqsubseteq_{\mathsf{var}} i$, there exists some $s' \sqsubseteq_{\mathsf{var}} s$ such that i'* **ioco** *s' holds.*

Proof. We prove i **mioco**$_{\mathsf{MIA}}$ $s \Rightarrow \forall i' \sqsubseteq_{\mathsf{var}} i : \exists s' \sqsubseteq_{\mathsf{var}} s : i'$ **ioco** s' by contradiction. We choose s' to be s_{fam} for all $i' \sqsubseteq_{\mathsf{var}} i$. Assume that there is an $i' \sqsubseteq_{\mathsf{var}} i$ such that i' **ioco** s_{fam} does not hold, i. e., there exists a $\sigma \in Straces(s_{fam})$ such that $Out(i'$ **after** $\sigma) \not\subseteq Out(s'$ **after** $\sigma)$. By Lemma 1, $\sigma \in Straces_{\Diamond}(s)$ and by construction of s_{fam} it holds that $Out(s_{fam}$ **after** $\sigma) = Out_{\Diamond}(s$ **after**$_{\Diamond}$ $\sigma)$. It also holds that $Out(i'$ **after** $\sigma) \subseteq Out_{\Diamond}(i$ **after**$_{\Diamond}$ $\sigma)$, which implies that $Out_{\Diamond}(i$ **after**$_{\Diamond}$ $\sigma) \not\subseteq Out_{\Diamond}(s$ **after**$_{\Diamond}$ $\sigma)$ contradicting the assumption that i **mioco**$_{\mathsf{MIA}}$ s. Thus, there is no $i' \sqsubseteq_{\mathsf{var}} i$ such that i' **ioco** s_{fam} does not hold. □

The converse does not hold in general. Consider the MIAs of Figure 4, where $s_{fam} = s$ and each variant $i' \sqsubseteq_{\mathsf{var}} i$ exhibits i' **ioco** s. However, i **mioco**$_{\mathsf{MIA}}$ s does not hold, as s specifies an output b as mandatory while in i, the b-transition is optional. We observe that each **ioco** check does not cover the fact that mandatory behavior of the specification s must also be mandatory behavior of i. This is due to the fact that in **ioco** only allowed outputs may be implemented, but an obligation to implement any output, as imposed by must-modalities, is not covered. If we ensure that mandatory behavior of s is preserved by i, as e. g., under MIA-refinement, the completeness claim holds. Thus, we obtain the following completeness claim.

Theorem 2 (Completeness I)**.** *Let i, s be MIAs such that i is input-enabled and $i \sqsubseteq_{\mathsf{MIA}} s$. If for all $i' \sqsubseteq_{\mathsf{var}} i$ it holds that i'* **ioco** *s_{fam}, then i* **mioco**$_{\mathsf{MIA}}$ *s.*

Proof. Assume i **mioco**$_{\mathsf{MIA}}$ s does not hold, but for all $i' \sqsubseteq_{\mathsf{var}} i$ it holds that i'**ioco** s_{fam}. This means that (1) there exists a $\sigma \in Straces_{\Diamond}(s)$ such that $Out_{\Diamond}(i$ **after**$_{\Diamond}$ $\sigma) \not\subseteq Out_{\Diamond}(s$ **after**$_{\Diamond}$ $\sigma)$ or (2) there exists a $\sigma \in Straces_{\Box}(i)$ so that $Out_{\Box}(s$ **after**$_{\Box}$ $\sigma) \not\subseteq Out_{\Box}(i$ **after**$_{\Box}$ $\sigma)$.

Case (1): It holds that $\sigma \in Straces_{\Diamond}(i)$. We construct a variant of i respecting σ as follows. Let $i' \sqsubseteq_{\mathsf{MIA}} i$ the largest MIA (w. r. t. $\sqsubseteq_{\mathsf{MIA}}$) such that whenever $i \xrightarrow{\sigma}_{\Diamond} q \xrightarrow{a}_{\Diamond} q'$ then $i' \xrightarrow{\sigma}_{\Box} q \xrightarrow{a}_{\Box} q'$. Hence, $Out_{\Box}(i' \textbf{ after}_{\Box} \sigma) = Out_{\Diamond}(i \textbf{ after}_{\Diamond} \sigma)$. i_{σ} is the variant of i that includes all must transitions of i'. But then $Out(i_{\sigma} \textbf{ after } \sigma) = Out_{\Box}(i' \textbf{ after}_{\Box} \sigma) \not\subseteq Out_{\Box}(s \textbf{ after}_{\Box} \sigma)$ and thus $i_{\sigma} \textbf{ ioco } s_{fam}$ does not hold, which contradicts the assumption that all variants of i conform to s_{fam}.

Case (2): It holds that $\sigma \in Straces_{\Box}(s)$. As $Out_{\Box}(s \textbf{ after}_{\Box} \sigma) \not\subseteq Out_{\Box}(i \textbf{ after}_{\Box} \sigma)$, there is an $s' \in s \textbf{ after}_{\Box} \sigma$ such that $s' \xrightarrow{a}_{\Box}$ for some $a \in O$, but for all $i' \in i \textbf{ after}_{\Box} \sigma$, it holds that $i' \not\xrightarrow{a}_{\Box}$. But this contradicts the assumption that $i \sqsubseteq_{\mathsf{MIA}} s$, as by Lemma 2 there is an $i' \in i \textbf{ after}_{\Box} \sigma$ and $i' \sqsubseteq_{\mathsf{MIA}} s'$.

Thus, $i' \textbf{ ioco } s_{fam}$ for all $i' \sqsubseteq_{\mathsf{var}} i$ implies that $i \textbf{ mioco}_{\mathsf{MIA}} s$. □

Thus, our $\textbf{mioco}_{\mathsf{MIA}}$ framework is sound, and complete in case the implementation is a refined version of the specification. When dropping the requirement of $i \sqsubseteq_{\mathsf{MIA}} s$, it is possible to show that if there is a variant i' of i such that $i' \textbf{ ioco } s_{fam}$ does not hold, then $i \textbf{ mioco}_{\mathsf{MIA}} s$ does not hold, either.

Theorem 3 (Completeness II)**.** *Let i, s be MIAs such that i is input-enabled. If there is an $i' \sqsubseteq_{\mathsf{var}} i$ such that $i' \textbf{ioco } s_{fam}$ does not hold, then $i \textbf{ mioco}_{\mathsf{MIA}} s$ does not hold.*

Proof. Let $i' \sqsubseteq_{\mathsf{var}} i$ be an IOLTS such that $i' \textbf{ ioco } s_{fam}$ does not hold, i. e., there exists a $\sigma \in Straces(s_{fam})$ so that $Out(i' \textbf{ after } \sigma) \not\subseteq Out(s_{fam} \textbf{ after } \sigma)$. By Lemma 1, $\sigma \in Straces_{\Diamond}(s)$ and also $Out_{\Diamond}(i \textbf{ after}_{\Diamond} \sigma) \neq \emptyset$. From the construction of s_{fam}, $Out_{\Diamond}(i \textbf{ after}_{\Diamond} \sigma) \not\subseteq Out_{\Diamond}(s \textbf{ after}_{\Diamond} \sigma)$ implying $i \textbf{ mioco}_{\mathsf{MIA}} s$ does not hold. □

Theorem 1 ensures that whenever $\textbf{mioco}_{\mathsf{MIA}}$ is established between product line implementation i and product line specification s, then each variant i' derived from i I/O-conforms to s_{fam}. Correspondingly, Theorem 2 and Theorem 3 state that whenever $\textbf{mioco}_{\mathsf{MIA}}$ cannot be established between i and s, then there is at least one variant i' of i not I/O-conforming to s_{fam}. According to Theorem 2, this is only ensured if $i \sqsubseteq_{\mathsf{MIA}} s$ holds. Summarizing, our $\textbf{mioco}_{\mathsf{MIA}}$ reflects the essence of family-based product line analysis [19] by means of I/O-conformance testing.

6 Conclusion and Future Work

In this paper, we proposed a family-based I/O-conformance testing theory for product lines based on Modal Interface Automata, which is sound and complete w. r. t. variant-by-variant I/O-conformance testing based on IOLTS. As future work, we plan to exploit the MIA framework for its compositionality properties to obtain criteria for compositional I/O-conformance testing of product lines. Therefore, dealing with internal actions, excluded from this papers' considerations, is inevitable. However, the results we obtained throughout this paper canonically extend to the case of MIAs with internal actions. This way, we obtain a similar variability concept as Larsen et al. [12], which is based on modal refinement and the ability of composition with an environmental specification validating implementation variants. Furthermore, we plan to implement our theory, based on a $\textbf{mioco}_{\mathsf{MIA}}$-extended version of JTorX [5] to provide an applicable tool for efficient product line I/O conformance testing.

References

[1] Luca de Alfaro & Thomas A. Henzinger (2005): *Interface-Based Design*. In: *Engineering Theories of Software Intensive Systems*, *NATO Science Series* 195, Springer, pp. 83–104, doi:10.1007/1-4020-3532-2_3.

[2] Rajeev Alur, Thomas A. Henzinger, Orna Kupferman & Moshe Y. Vardi (1998): *Alternating Refinement Relations*. In: *Concur'98*, Springer, pp. 163–178, doi:10.1007/BFb0055622.

[3] Patrizia Asirelli, Maurice H. ter Beek, Alessandro Fantechi & Stefania Gnesi (2011): *A Model-Checking Tool for Families of Services*. In: *LNCS 6722*, pp. 44–58, doi:10.1007/978-3-642-21461-5_3.

[4] Sebastian S. Bauer, Rolf Hennicker & Stephan Janisch (2011): *Interface Theories for (A)synchronously Communicating Modal I/O-Transition Systems*. *EPTCS* 46, pp. 1–8, doi:10.4204/EPTCS.46.1.

[5] Axel Belinfante (2010): *JTorX: A Tool for On-Line Model-Driven Test Derivation and Execution.* In Javier Esparza & Rupak Majumdar, editors: *TACAS*, *LNCS* 6015, Springer, pp. 266–270.

[6] Harsh Beohar & Mohammad Reza Mousavi (2014): *Input-output Conformance Testing Based on Featured Transition Systems*. In: *Proc. of SAC'14*, ACM, New York, NY, USA, pp. 1272–1278, doi:10.1145/2554850.2554949.

[7] Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens & Axel Legay (2011): *Symbolic Model Checking of Software Product Lines*. In: *ICSE'11*, pp. 321–330, doi:10.1145/1985793.1985838.

[8] Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay & Jean-François Raskin (2010): *Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines*. In: *ICSE '10*, pp. 335–344, doi:10.1145/1806799.1806850.

[9] Paul Clements & Linda Northrop (2001): *Software Product Lines: Practices and Patterns*. Addison-Wesley Longman Publishing Co., Inc.

[10] Dario Fischbein, Sebastin Uchitel & Víctor A. Braberman (2006): *A Foundation for Behavioural Conformance in Software Product Line Architectures*. In Robert M. Hierons & Henry Muccini, editors: *ISSTA'06*, ACM, pp. 39–48, doi:10.1145/1147249.1147254.

[11] Kim G. Larsen (1990): *Modal Specifications*. In: *Automatic Verification Methods for Finite State Systems*, *LNCS* 407, Springer, pp. 232–246, doi:10.1007/3-540-52148-8_19.

[12] Kim G. Larsen, Ulrik Nyman & Andrzej Wasowski (2007): *Modal I/O Automata for Interface and Product Line Theories*. In: *Proc. of ESOP'07*, LNCS 4421, Springer, pp. 64–79, doi:10.1007/978-3-540-71316-6_6.

[13] Kim G. Larsen & Bent Thomsen (1988): *A Modal Process Logic*. In: *LICS*, pp. 203–210.

[14] Malte Lochau, Sven Peldszus, Matthias Kowal & Ina Schaefer (2014): *Model-Based Testing*. In: *Formal Methods for Executable Software Models*, *LNCS* 8483, Springer, pp. 310–342, doi:10.1007/978-3-319-07317-0_8.

[15] Gerald Lüttgen & Walter Vogler (2013): *Modal Interface Automata*. *LMCS* 9.

[16] Rocco de Nicola (1987): *Extensional equivalences for transition systems*. *Acta Informatica* 237, pp. 211–237, doi:10.1007/BF00264365.

[17] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, Axel Legay & Roberto Passerone (2009): *Modal Interfaces: Unifying Interface Automata and Modal Specifications*. In: *EMSOFT'09*, ACM, pp. 87–96, doi:10.1145/1629335.1629348.

[18] Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoît Caillaud, Axel Legay & Roberto Passerone (2011): *A Modal Interface Theory for Component-based Design*. *Fundam. Inf.* 108, pp. 119–149.

[19] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer & Gunter Saake (2014): *A Classification and Survey of Analysis Strategies for Software Product Lines*. *ACM Comput. Surv.* 47, pp. 6:1–6:45, doi:10.1145/2580950.

[20] Jan Tretmans (1996): *Test Generation with Inputs, Outputs and Repetitive Quiescence*. *Software – Concepts and Tools* 17(3), pp. 103–120.

[21] Frits W. Vaandrager (1991): *On the Relationship Between Process Algebra and Input/Output Automata*. In: *Proc. of LICS '91*, pp. 387–398, doi:10.1109/LICS.1991.151662.