Compositional specifications for IOCO testing

Przemyslaw Daca, Thomas A Henzinger, Willibald Krenn, Dejan Nickovic

https://doi.org/10.15479/AT:IST-2014-148-v2-1

Deposited at: 12 Dec 2018 11:54  ISSN: 2664-1690
Copyright © 2014, by the author(s).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.
Compositional Specifications for IOCO Testing; Technical Report

Przemysław Daca and Thomas A Henzinger and Willibald Kreum and Dejan Nićkovic
Compositional Specifications for \textbf{ioco} Testing: Technical Report

Przemysław Daca  
and Thomas A. Henzinger  
IST Austria  
Klosterneuburg, Austria

Willibald Krenn  
and Dejan Ničković  
AIT Austrian Institute of Technology GmbH.  
Vienna, Austria

Abstract—Model-based testing is a promising technology for black-box software and hardware testing, in which test cases are generated automatically from high-level specifications. Nowadays, systems typically consist of multiple interacting components and, due to their complexity, testing presents a considerable portion of the effort and cost in the design process. Exploiting the compositional structure of system specifications can considerably reduce the effort in model-based testing. Moreover, inferring properties about the system from testing its individual components allows the designer to reduce the amount of integration testing.

In this paper, we study compositional properties of the ioco-testing theory. We propose a new approach to composition and hiding operations, inspired by contract-based design and interface theories. These operations preserve behaviors that are compatible under composition and hiding, and prune away incompatible ones. The resulting specification characterizes the input sequences for which the unit testing of components is sufficient to infer the correctness of component integration without the need for further tests. We provide a methodology that uses these results to minimize integration testing effort, but also to detect potential weaknesses in specifications. While we focus on asynchronous models and the ioco conformance relation, the resulting methodology can be applied to a broader class of systems.

I. INTRODUCTION

Modern software and hardware system design usually involves the integration of interacting components that work together in order to realize some requested behavior. Figure 1 illustrates two components $I_1$ and $I_2$ that are composed together to form the system $I$. The complexity of the individual components together with their elaborate cooperation protocols often results in behavioral faults. Therefore, verification and validation methods are applied to ensure that the embedded system satisfies its specification. This is in particular true for safety-critical designs, for which correctness evidence is imposed by the regulation bodies (see for example the automotive standard ISO 26262 [1]).

Up to date, design simulation combined with testing remains the preferred technique in industry to demonstrate the correctness of software and hardware systems. Typically, verification engineers need to test individual system components (\textit{unit testing} of $I_1$ and $I_2$), and then test the complete system (\textit{integration testing} of $I$). This process relies on verification engineers manually generating test vectors from specifications given as informal (natural language) requirements. This process is inherently time consuming, ad-hoc and prone to human errors. As a result, testing represents the main bottleneck in the design of complex systems today.

Fig. 1. A system consisting of two interacting components.

Model-based testing is a technology that provides formalization and automation to the test case generation and execution process, thus reducing time and cost of systems design. In model-based testing a system-under-test (SUT), denoted by $I$, is tested for conformance to its formal specification model $S$, derived from informal requirements. While $S$ is a formal object, $I$ is a “black-box”, a physical implementation with unknown internal structure. The SUT $I$ can be accessed by the tester only through its external interface. In order to reason about the compliance of $I$ to $S$, one needs to use the testing assumption (see [2]), stating that $I$ can be modeled in the same formalism as $S$ and that $I$ is receptive (input-enabled), i.e. it accepts all inputs at any point in time. In contrast to $I$, $S$ does not need to be receptive. Lack of input \textit{(under-specification)} in a given state of the specification models the assumption that the external environment for $I$ does not provide that input. If the environment nevertheless emits this input action, the specification allows $I$ to freely choose its response.

In this paper, we focus on the \textbf{ioco}-testing theory [2], a model-based testing framework for input/output labeled transition systems (IOLTs). The \textbf{ioco}-testing theory is centered around the input/output conformance relation \textbf{ioco}. Informally, we say that an implementation $I$ \textbf{ioco}-conforms to its specification $S$ if any experiment derived from $S$ and executed on $I$ leads to an output in $I$ that is foreseen by $S$. In the \textbf{ioco}-testing theory, the lack of outputs or internal transitions is observable via a quiescence action.

In its original form, \textbf{ioco}-testing does not take the compositional aspects of systems into account. For example, in Figure 1 $I$ is the result of composing the components $I_1$ and $I_2$. Typically, the actions over which the two components synchronize ($c$ and $d$ in the example) are hidden and become
unobservable to the tester after integration. In order to cope with costly testing of large systems, results of unit testing individual components need to be used to infer properties about the composed system, to avoid or at least minimize expensive integration tests. The compositional \textit{ioco}-testing problem can be formulated as follows: if $I_1$ \textit{ioco}-conforms to its specification model $S_1$ and $I_2$ \textit{ioco}-conforms to its specification model $S_2$, can we infer that $I$ also \textit{ioco}-conforms to $S$, where $I$ and $S$ denote the composed implementation and specification after hiding synchronization actions?

This question was addressed in [3]. The authors show that in the general case neither parallel composition of specifications nor hiding of actions are compositional in the \textit{ioco}-testing theory. This result is not surprising. Parallel composition is an operation tailored to receptive models. We argue that it is not an appropriate operator for composing under-specified models where one component can generate an output which is not expected as an input by the other. In addition, the hiding operation introduces partial observation over the actual state of the SUT and can result in confusion regarding the under-specified parts of the system. The authors of [3] propose two alternative restrictions to the specification models in order to preserve compositionality of \textit{ioco}. The first option is to disallow under-specification of inputs. This is a very strong requirement in practice, since components are usually designed to operate in constrained environments. The second option allows starting with under-specification, but requires \textit{demonic completion} of specification models — an operation which makes the assumptions about the component’s environment explicit and thus makes the specification model input-enabled. We claim that demonic completion can hide important information from the tester about the poor quality of a specification and thus obscure its original intent.

We propose a different approach to compositional \textit{ioco}-testing which, in contrast to [3], does not restrict the specification models. We define two operations — \textit{friendly composition} and \textit{hiding} — that are tailored to the integration of non-receptive specifications. They are based on a game-theoretic \textit{optimistic} approach, inspired by interface theories [4], [5]. The result of the friendly composition is the overall specification that integrates the component specifications while pruning away any inputs that lead to incompatible interactions between the components. The friendly hiding operation prunes away inputs that lead to states which are ambiguous with respect to under-specification after hiding. After composing the component specifications followed by hiding of synchronizing actions, the resulting specification defines all input sequences for which no integration testing is needed — the correct integration follows from the conformance of the individual components to their specification. In addition to these technical results, this paper provides guidelines to identify specifications that are poorly modeled for compositional testing. We argue that pruned input sequences often indicate weaknesses in the specification and can be addressed by: (1) strengthening the specifications; (2) making more outputs observable; and (3) integration testing. Indeed, the proper formalization of requirements resulting in high-quality component specifications is crucial for exploiting the compositional nature of systems in testing. Investing efforts in improving models can considerably minimize expensive integration tests. We discuss methodological aspects of using our technical results to improve component models and to tailor them to compositional testing.

In Section II, we further motivate the problem of compositional testing with \textit{ioco} and provide an informal overview of our approach, illustrating it with a vending machine example. We also identify modeling issues and discuss problems related to compositional \textit{ioco}-testing and sketch possible solutions. Section III recalls the basics of the \textit{ioco}-testing theory including the known results about compositional \textit{ioco}-testing. We provide the formal presentation of our approach in Section IV and evaluate it in Section V. We present related work in more detail in Section VI and finally conclude the paper in Section VII  giving future perspectives for our work. The proofs are presented in Appendix A.

II. OVERVIEW AND MOTIVATING EXAMPLE

In this section, we develop the example of a \textit{drink vending machine}, which we use to (1) further motivate the problem; (2) highlight the difficulties of compositional testing within the \textit{ioco}-theory; and (3) provide an informal overview of our proposed approach to tackle the problem.

The drink vending machine consists of the \textit{user interface} and the \textit{drink maker} components. The user interface specification $S_1$ is shown in Figure 2(a). $S_1$ requires that the user first inserts a coin (\textit{coin}?) and then selects either a tee (\textit{utee}?) or a coffee (\textit{ucoffee}?). After choosing the coffee, the user can also request milk (\textit{umilk}?). The drink request (\textit{mtee}, \textit{mcoffee}! or \textit{mcoffee}!\textit{milk}!) is forwarded to the drink maker, and the user interface waits for an acknowledgment (\textit{done}?) that the drink was delivered to the user. When the drink is ready, the user interface emits a message (\textit{msg}?) to the user and returns to its initial state.

The drink maker specification $S_2$, depicted in Figure 2(b), waits for a drink request (\textit{mcoffee}? or \textit{mcoffee}!\textit{milk}!) from the user interface. Upon receiving the drink request, $S_2$ signals the delivery of the drink to the user (actions \textit{coffee}! and \textit{coffee}!\textit{milk}!) and finally an acknowledgment (\textit{done}!) is sent to the user interface. Note that $S_2$ is under-specified in its initial state $A$ — it omits the action \textit{mtee}? and thus makes an assumption that the user interface never requests a tee.

We note that in states 1 and 2 of $S_1$, and state A of $S_2$ only inputs are allowed. In the \textit{ioco}-testing theory, such states are called \textit{quiescent}, where the quiescence denotes the absence of observable outputs and internal actions. The absence of outputs is considered to be observable, and is marked as a special $\delta$ action (green self-loops in Figure 2). Since quiescence is usually not explicitly modeled by the designer, we omit marking quiescent transitions in the rest of the section.

Specifications $S_1$ and $S_2$ give a certain freedom in implementing user interface and drink machine components. In particular, implementations can choose how to treat under-specified (unexpected) inputs. For instance, $S_1$ assumes that the user inserts a coin before choosing the drink. If the user swaps the order of actions, and first orders a drink, $S_1$ allows the implementation to react to this input in an arbitrary way. Figure 3 depicts possible implementations $I_1$ and $I_2$ of their respective specifications $S_1$ and $S_2$. The user

\footnote{We consistently use the symbol ? to denote an input, and the symbol ! to denote an output action.}
interface implementation $I_1$ closely follows its specification and silently ignores all unexpected inputs (marked in blue). For example, if the user requests a coffee before inserting a coin (in state 1), this request is silently consumed by $I_1$. This implementation is ioco-conformant to its specification because it never generates an output which is not foreseen by $S_1$. Similarly to $I_1$, the drink maker implementation $I_2$ also silently consumes unexpected inputs in all states, except in the state $A$. In the initial state, $I_2$ reacts to a tee request (mtee?) by moving to the state $B$, from which a coffee (coffee!) is delivered to the user. Although preparing a coffee upon a tee request may not be a logical behavior, it is ioco-conformant to $S_2$ — the specification does not impose any particular reaction to the tee request in its initial state; giving to correct implementations complete freedom of handling such input.

In the classic ioco-theory [3], specifications are combined with the parallel composition operation for input/output transitions systems [6]. Informally, parallel composition of two specifications is their Cartesian product, where each specification is allowed to take local actions independently, but the two specifications must synchronize on shared actions. Figure 4 (a) depicts the parallel composition of $S_1$ and $S_2$, where mtee, mcoffee, mcoffeemilk and done are the shared actions. In addition to parallel composition, we may wish to hide shared actions, which are often only used to synchronize components, but are not observable by the external user. In the vending machine example, the user can observe inputs to the vending machine (inserting a coin and choosing the drink) and the outputs from the machine (the actual drink and the acknowledgment message). The actions mtee, mcoffee, mcoffeemilk and done are used for proper synchronization between the user interface and the drink maker components and are not visible to the user. Figure 4 (b) depicts the parallel composition of specifications $S_1$ and $S_2$ in which the shared actions are hidden (denoted by the special action $\tau$). Figures 5 (a) and (b) show the parallel composition of the component implementations without and with hiding of the synchronization actions, respectively.

While $I_1$ is ioco-conformant to $S_1$ and $I_2$ is ioco-conformant to $S_2$, the composition of $I_1$ and $I_2$ is not ioco-conformant to the composition of $S_1$ and $S_2$ (with or without hiding of shared actions), as shown in [3]. We now explain the reasons for non-compositionality of ioco in the vending machine example.

Assuming that shared actions are not hidden, consider a test case starting with the input sequence coin? · utee?. According to the specification (Figure 4(a)), the only allowed observation
after reading this sequence is quiescence, since state 3A does not have any outgoing transitions labeled by an output action. However, the composed implementation (Figure 5 (a)) emits mtee!, an output action not allowed by the specification after reading the same sequence. It follows that loco-conformance is not preserved by parallel composition.

Parallel composition is an operation tailored to combining receptive components — whenever a component outputs a shared action the other component is by definition able to consume it from any of its local states. This is not the case for non-receptive models. Indeed, S1 emits the shared action mtee! in its local state 3 while the drink maker specification S2 is not ready to consume it in its local state A; i.e. the assumption of S2 is not fulfilled by S1. This results in a “deadlock” state 3A in the parallel composition of S1 and S2. However, the intended meaning of under-specifying the action mtee? in the state A of S2 is that S2 is free to choose any reaction to this unexpected action. This is in contrast to what happens in state 3A of the composed specification.

We now consider the case when the shared actions are hidden. After reading the input sequence coin? · ucoffee?, the state of the composed system after hiding is not uniquely defined — it can be either 4A or 6B (Figure 5 (b)) since the user cannot observe whether the hidden action coffeemilk! has taken place. Note that the specification (Figure 4 (b)) leaves the input umilk? unspecified in 6B, but not in 4A, thus resulting in an ambiguity on what an external observer can expect as the reaction to this input. In fact, according to the loco-theory, the only allowed observable output after executing the sequence coin? · ucoffee? · umilk? is coffeemilk!, while the composed implementation can output both coffeemilk! (if in state 6C) or coffee! (if in state 6B).

In [3], the authors propose two solutions to the above anomalies. Both solutions guarantee that loco-conformance is preserved by parallel composition and hiding. The first solution requires specification models to be receptive. We claim that this restriction is too strong — under-specification of inputs is one of the most powerful modeling tools for specifying open systems. A component is almost always expected to work correctly only in constrained contexts, and under-specification of inputs allows to exactly define a valid operating environment. The second solution allows non-receptive models, but requires their demonic completion — an operation that makes the model effectively input-enabled. Demonic completion results in adding transitions labeled with the under-specified inputs from every state to a newly inserted “sink” portion of the graph. The sink portion essentially self-loops with all inputs and outputs. Demonic completion makes the intended meaning of input under-specification explicit: when a system receives an unexpected input, it has the full freedom to choose its reaction to this input. Figure 5 depicts parts of the demonically completed specifications S1 and S2.

While demonic completion preserves the intended meaning of specifications, we argue that it does not provide a fully satisfactory solution to compositional loco-testing. First, the resulting specification after demonic completion increases in size. Although linear, this increase is still important for extensively under-specified models. For instance, S1 has 7 states and 9 transitions, while its size increases to 10 states and 55 transitions after completion. Second, demonic completion obfuscates the distinction between foreseen and unspecified interactions between components. In Figure 5 (a) the information that the input action ucoffee? is not expected in state 6 is lost.

This lack of distinction between foreseen and unexpected interactions between components masks the fact that we often deal with component specifications of poor quality. This results in a composition which does not faithfully represent the intended behavior of the overall system. In particular, composition with hiding of demonically completed specifications may result in many vacuous behaviors. We illustrate this problem with the example from Figure 6. After composing demonically completed variants of S1 and S2, and hiding the synchronization action coffeemilk, the external observer cannot distinguish between states 4A and 6B, which in contrast to the original composition (see Figure 4 (b)), now both admit the input action umilk?. However, umilk? triggers a transition from 6B to the state D1B, from which the demonically completed specification S1 allows all possible behaviors, including the one in which the acknowledgment message is sent back to the user without any drink being served.

In adding transitions labeled with the under-specified inputs from every state to a newly inserted “sink” portion of the graph. The sink portion essentially self-loops with all inputs and outputs. Demonic completion makes the intended meaning of input under-specification explicit: when a system receives an unexpected input, it has the full freedom to choose its reaction to this input. Figure 5 depicts parts of the demonically completed specifications S1 and S2.

We propose an alternative approach to composition and hiding that provides additional information to the designer. It is inspired by interface theories [4, 5]. We provide a new composition operation, called the friendly composition,
which takes an optimistic approach to combining two specifications. Following the optimistic approach, two specifications are compatible for composition, if there exists some context in which they can interact while both satisfying their guarantees. We have seen in Figure 3 that the interaction of two specifications (user interface and drink maker) results in states in which one component is allowed to emit an output which is not expected by the other component (action mitee! in the state 3A). We declare such states as ambiguous states, and compute the maximal compound environment which avoids such states. The algorithm that computes the friendly composition of two specifications prunes all states from which the environment cannot prevent the composed system from reaching an ambiguous state. The resulting composite specification combines the compatible interactions of two component specifications. Figure 4(a) depicts the friendly composition of S1 and S2, where red dashed transitions and states are the ones pruned away from their parallel composition.

Similarly to friendly composition, we define the friendly hiding operation, which prunes away from the specification with hidden actions all the states that can become ambiguous when interacting with an external environment. The resulting specification is depicted in Figure 4(b), where red dashed transition and states are pruned away from the composite specification of S1 and S2 after hiding.

The main technical contribution of this paper is the definition of the friendly composition and friendly hiding operations, for which we show that they preserve ioco-conformance. In contrast to combining demonic completion with parallel composition and hiding, our approach results in composite specifications of smaller size. The resulting composite specification defines behaviors for which no integration testing is needed. Apart from technical contribution, we argue that friendly composition and hiding expose weaknesses of component specifications to the designer. The pruned behaviors after applying friendly composition and hiding indicate that assumptions made by individual component specifications may be too weak and deserve more careful analysis. We claim that the time spent on improving the quality of the component specifications so they allow compositional testing is rewarded with the avoidance of integration tests. We distinguish between the following scenarios in using our approach to derive better compositional specifications.

Scenario A: the designer can guarantee that the composed system will be used in the context defined by the friendly-composed specification assumptions. In this case, the specification is too weak and needs a revision. In our example, the race between the input action unilk? and the output action mcoffee! in S1 indicates a poor specification. We propose a different specification, which requires an additional action and in which the user is expected to make all requests before the machine is able to process them.

III. Preliminaries

In this section, we define labeled transition systems, parallel composition and hiding operations, input/output conformance relation (ioco), and recall the previous results on compositional properties of ioco.

A. Labeled Transition Systems

An input/output labeled transition system (IOLTS) is a formal model for specifying reactive systems. An IOLTS A is a tuple (Q, L', L'', T, q), where Q is a countable set of states, L' and L'' are disjoint countable sets of input and output labels, q ∈ Q is the initial state and T is the transition relation. We denote by L = L' ∪ L''. An IOLTS is said to be receptive, denoted by R-IOLTS, if for all q ∈ Q and for all a ∈ L', there exists an outgoing transition from q labeled by a. For instance, specifications S1 and S2 in Figure 2 are IOLTS, while implementations I1 and I2 in Figure 3 are R-IOLTS. Strongly-convergent IOLTS are transition systems that do not have loops consisting of only silent transitions. We use the standard abbreviated notation, where μ ∈ L ∪ {τ} and a ∈ L.

\[
\begin{align*}
q \xrightarrow{μ} q' & \equiv (q, μ, q') \in T \\
q \xrightarrow{μ_1 \cdots μ_n} q' & \equiv \exists q_0, \ldots, q_n, \text{st. } q = q_0 \xrightarrow{μ_1} q_1 \\
q \xrightarrow{μ_1 \cdots μ_n} & \equiv \exists q' \text{ st. } q \xrightarrow{μ_1 \cdots μ_n} q' \\
q \xrightarrow{τ} q' & \equiv q = q' \text{ or } q \xrightarrow{τ} q' \\
q \xrightarrow{a} q' & \equiv \exists q_1, q_2, q_1 \xrightarrow{a} q_2 \xrightarrow{a} q' \\
q \xrightarrow{a_1 \cdots a_n} q' & \equiv \exists q_0, \ldots, q_n, \text{st. } q = q_0 \xrightarrow{a_1} q_1 \\
q \xrightarrow{a_1 \cdots a_n} & \equiv \exists q' \text{ st. } q \xrightarrow{a_1 \cdots a_n} q' \\
q \xrightarrow{a_1 \cdots a_n} & \equiv \exists q' \text{ st. } q \xrightarrow{a_1 \cdots a_n} q'
\end{align*}
\]

A sequence σ ∈ (L ∪ {τ})∗ is an execution of an IOLTS A if ˆqA ˆσ. A sequence σ ∈ L∗ is a trace of A if ˆqA ˆσ. We denote by Traces(A) the set of all traces of A. The sequence coin · ucoffee · τ · coffee is an execution of the specification shown in Figure 4(b), while coin · ucoffee · coffee is its trace. Given a subset of labels L' ⊆ L and σ a sequence over L, we denote by σ|L' the projection σ to the set of labels L'.

We say that a state q ∈ Q of A is quiescent, denoted by δ(q), if it has no outgoing output or internal actions. Quiescent states emit a special quiescence action δ, which indicates that A cannot proceed without input from the environment.
The suspension automata are IOLTS, where quiescent actions are made explicit. Formally, given an IOLTS $A = (Q, L^I, L^O, T, \hat{q})$, its suspension automaton $A_\delta$ is the IOLTS $A_\delta = (Q, L^I, L^O \cup \{\delta\}, T \cup T_\delta, \hat{q})$, where $T_\delta = \{ q \xrightarrow{\delta} q \mid \delta(q) \}$. We denote by $\text{Straces}(A)$ the set $\{ \sigma \in (L \cup \{\delta\})^* \mid \hat{q} \xrightarrow{\sigma} A_\delta \}$ of traces of $A_\delta$, also called suspension traces. Specifications $S_1$ and $S_2$ shown in Figure 2 show explicitly quiescence actions, where for example $\delta \cdot \text{coin} \cdot \delta$ is in $\text{Straces}(S_1)$.

B. Parallel Composition

Two components can be integrated if their input/output actions do not conflict. In particular, we require that the intersection of their input (output) label sets is empty. Formally, we say that two IOLTS $A_1$ and $A_2$ are composable if $L^I_1 \cap L^I_2 = L^O_1 \cap L^O_2 = \emptyset$. When two composable IOLTS are composed, they synchronize on shared actions and move independently on other actions. Formally, parallel composition is defined as follows.

**Definition 1 (Parallel composition):** Let $A_1 = (Q_1, L^I_1, L^O_1, T_1, \hat{q}_1)$ and $A_2 = (Q_2, L^I_2, L^O_2, T_2, \hat{q}_2)$ be two composable IOLTS. Their parallel composition, denoted by $A_1 \parallel A_2$, is the IOLTS $(Q_{1\parallel2}, L^I_{1\parallel2}, L^O_{1\parallel2}, T_{1\parallel2}, \hat{q}_{1\parallel2})$, where $Q_{1\parallel2} = Q_1 \times Q_2$, $L^I_{1\parallel2} = (L^I_1 \setminus L^O_2) \cup (L^I_2 \setminus L^O_1)$, $L^O_{1\parallel2} = L^O_1 \cup L^O_2$, and $T_{1\parallel2}$ is defined by the rules:

\[
\begin{align*}
q_1 \xrightarrow{\mu} A_1, q'_1 \mu & \in (L_1 \cup \{\tau\}) \setminus L_2 \\
(q_1, q_2) & \xrightarrow{A_{1\parallel2}} (q'_1, q_2)
\end{align*}
\]

\[
\begin{align*}
q_2 \xrightarrow{\mu} A_2, q'_2 \mu & \in (L_2 \cup \{\tau\}) \setminus L_1 \\
(q_1, q_2) & \xrightarrow{A_{1\parallel2}} (q_1, q'_2)
\end{align*}
\]

\[
\begin{align*}
q_1 \xrightarrow{\mu} A_1, q'_1 \mu & \in L_1 \setminus L_2 \\
(q_1, q_2) & \xrightarrow{A_{1\parallel2}} (q'_1, q_2)
\end{align*}
\]

The specification $S_1 \parallel S_2$ shown in Figure 2(a) represents the parallel composition of specifications $S_1$ and $S_2$.

C. Hiding

The parallel composition of two components is often followed by hiding some of the actions on which they synchronize. We follow the process algebraic approach in which parallel composition and hiding operations are two separate operations, and formally define hiding as follows.

**Definition 2 (Hiding):** Let $A = (Q, L^I, L^O, T, \hat{q})$ be an IOLTS and $\Sigma \subseteq L^O$ be the subset of output actions. The hiding of $\Sigma$ in $A$, denoted by $h_\Sigma(A)$, is the tuple $(Q, L^I, L^O \setminus \Sigma, h_\Sigma(T), \hat{q})$, where $h_\Sigma(T)$ is obtained from $T$ by replacing every transition $(q, a, q') \in T$ labeled by an output action $a \in \Sigma$ by the transition $(q, \tau, q')$.

The specification $h_\Sigma(S_1 \parallel S_2)$ shown in Figure 2(b) represents the hiding of $\Sigma$ in $S_1 \parallel S_2$, where $\Sigma = \{\text{mtee}, \text{mcoffee}, \text{mcoffee}\text{milk}, \text{done} \}$.

D. Input/Output Conformance Relation

Given an IOLTS $A$, the set $\text{out}(q) = \{ a \in L^O \mid q \xrightarrow{a} \}$ is the set of all outputs (including $\delta$ if $q$ is quiescent) that are defined when the system is in state $q$. The set $\text{out}$ after $\sigma = \{ q' \mid q \xrightarrow{\sigma} A_\delta \}$ denotes the set of states that can be reached in $A$ from $q$ after reading $\sigma$ in its suspension automaton $A_\delta$. We now present the formal definition of the $\text{icoo}$ relation.

**Definition 3:** Given a R-IOLTS $I$ and an IOLTS $S$, we say that $I \text{ icoo } S$ iff

\[\forall \sigma \in \text{Straces}(S), \text{out}(\hat{q}_I \text{ after } \sigma) \subseteq \text{out}(\hat{q}_S \text{ after } \sigma).\]

In the vending machine example, both $I_1 \text{ icoo } S_1$ and $I_2 \text{ icoo } S_2$, where $S_1$ and $S_2$ are depicted in Figure 2(a) and (b), and $I_1$ and $I_2$ are depicted in Figure 3(a) and (b), respectively. We now recall the results from [3] which state that $\text{icoo}$ is not preserved in general under parallel composition and hiding, but is preserved if all the specifications are receptive. Receptiveness of specifications can be achieved by demonic completion (see [3] for its formal definition).

**Theorem 1 ([3]):** Given two composable R-IOLTS $I_1$ and $I_2$, two composable IOLTS $S_1$ and $S_2$ and two composable R-IOLTS $S_1^*$ and $S_2^*$, we have

\[I_1 \text{ icoo } S_1 \land I_2 \text{ icoo } S_2 \not\Rightarrow (I_1 \parallel I_2) \text{ icoo } (S_1 \parallel S_2).
\]

\[I_1 \text{ icoo } S_1^* \land I_2 \text{ icoo } S_2^* \Rightarrow (I_1 \parallel I_2) \text{ icoo } (S_1^* \parallel S_2^*).
\]

**Theorem 2 ([3]):** Given a R-IOLTS $I$, an IOLTS $S$, and a R-IOLTS $S^*$, defined over the alphabet $L$, and a subset $\Sigma \subseteq L^O$, we have

\[I \text{ icoo } S \not\Rightarrow h_\Sigma(I) \text{ icoo } h_\Sigma(S).
\]

Consider the vending machine example, in which $I_1 \text{ icoo } S_1$ and $I_2 \text{ icoo } S_2$, the sequence $\sigma = \text{coin} \cdot \text{mtee}$ and the compositions of specifications and implementations before and after hiding (see Figures 4 and 5). The available output after executing $\sigma$ on $I_1 \parallel I_2$ is $\text{mcoffee}$, while the composed specification $S_1 \parallel S_2$ allows only $\delta$ after executing the same sequence, hence $I_1 \parallel I_2$ does not $\text{icoo}$-conform to $S_1 \parallel S_2$. After hiding the actions $\Sigma = \{\text{mtee}, \text{mcoffee}, \text{mcoffee}\text{milk}, \text{done}\}$ in the composition, we obtain additional traces which are not $\text{icoo}$-conformant. For instance, after executing the sequence $\text{coin} \cdot \text{mcoffee} \cdot \text{mcoffee}\text{milk}$ in $h_\Sigma(I_1 \parallel I_2)$, the possible outputs are $\text{coffee}$ and $\text{coffee}\text{milk}$, while the specification $h_\Sigma(S_1 \parallel S_2)$ allows only the action $\text{coffee}\text{milk}$ after executing the same sequence.

Demonic completion introduces new “chaotic” states to the original specification, from which all behaviors are allowed. Exploring chaotic states in test case generation is not useful. In order to avoid their exploration [3] defines the set of $\text{Utraces}$. Intuitively, $\text{Utraces}$ restricts $\text{Straces}$ by eliminating underspecified traces. We obtain the $\text{uioco}$ conformance relation by replacing $\text{Straces}$ with $\text{Utraces}$ in the definition of $\text{icoo}$. However, it turns out that $\text{uioco}$ does not preserve compositional properties (see Appendix 3 for details).
IV. OPTIMISTIC APPROACH TO COMPOSITION AND HIDING

In this section, we formalize the friendly composition and hiding operations, presented informally in Section II.

A. Friendly Environments

We have seen in Section II that parallel composition and hiding can introduce ambiguous states, and that ioco is not preserved under these two operations. An ambiguous state results from the parallel composition of two IOLTS in which one emits an action that the other one is not ready to accept.

Definition 4 (Ambiguous state): Given two composable IOLTS $A$ and $B$, a pair $(q_A, q_B) \in Q_A \times Q_B$ is an ambiguous state if there exists a shared action $a \in L_A \cap L_B$ such that either: (1) $a \in L_A^0$, $q_A \not\xrightarrow{a}$ and $q_B \xrightarrow{a}$; or (2) $a \in L_B^0$, $q_B \not\xrightarrow{a}$ and $q_A \xrightarrow{a}$.

In the parallel composition $S_1 \parallel S_2$ of the vending machine example, depicted in Figure 2(a), the state $3A$ is ambiguous because $S_1$ emits the output action mteel, while $S_2$ does not accept it.

Inspired by contract-based design and interface theories, we propose an optimistic approach to composition and hiding. In this optimistic setting, we look for a friendly environment which steers the specification away from ambiguous states. A friendly environment is helpful towards the systems, by always accepting the system’s outputs and never providing actions that the system cannot accept as inputs.

Definition 5 (Friendly environment): Given an IOLTS $A = (Q, L^I, L^O, T, \hat{q})$, a friendly environment for $A$ is a strongly-convergent IOLTS $E = (Q_E, L^O_E, L^I_E, T_E, \hat{q}_E)$ such that $A \parallel E$ does not have ambiguous states.

A composition-friendly environment does not allow a composed system to reach an ambiguous state in the composition.

Definition 6 (Composition-friendly environment): A composed IOLTS $A \parallel B$ is a composition-friendly environment for $A$ if $B$ is its friendly environment such that for all ambiguous states $(q_A, q_B) \in Q_A \parallel Q_B$, for all $q_E \in E$, $((q_A, q_B), q_E)$ is not reachable in $(A \parallel B) \parallel E$.

A friendly environment is maximal if it admits more behaviors than any other friendly environment. The maximal friendly environment is used to compute the largest portion of the specification that is guaranteed to preserve conformance under composition and hiding. The resulting specification characterizes all sequences for which integration testing is not necessary.

Definition 7 (Maximal friendly environment): A friendly environment $E$ for an IOLTS $A$ is said to be maximal if for all friendly environments $E'$ for $A$ it holds $\text{Traces}(E') \subseteq \text{Traces}(E)$.

The IOLTS fragment that interacts correctly with its friendly environment $E$ is called its $E$-reachable fragment. It is obtained by composing the IOLTS with $E$, while keeping the original meaning of inputs and outputs.

Definition 8 (Environment reachable fragment): Let $A$ be an IOLTS and $E$ be its friendly environment. The $E$-reachable fragment of $A$ is an IOLTS $(Q, L^I, L^O, T, \hat{q})$, where $Q = Q_A \parallel E$, $\hat{q} = \hat{q}_A \parallel E$, $L^I = L^I_A$, $L^O = L^O_A$, and $T = T_A \parallel E$.

B. Friendly Composition, Friendly Hiding and ioco

We are now ready to formally define friendly composition and hiding, and show that ioco is a pre-congruence for these two operations. We use the maximal friendly environment to restrict the classical parallel composition and hiding operations to a fragment that guarantees avoiding ambiguous states.

Definition 9 (Friendly composition): Given two composable IOLTS $A$ and $B$, we say that they are compatible if there exists a composition-friendly environment $E$ for $A \parallel B$. Given two compatible IOLTS $A$ and $B$, their friendly composition, denoted by $A \parallel B$, is an $E$-reachable fragment of $A \parallel B$, where $E$ is the maximal composition-friendly environment for $A \parallel B$.

Lemma 1: For any two compatible IOLTS $A$ and $B$, there exists a maximal composition-friendly environment for $A \parallel B$.

Definition 10 (Friendly hiding): Given an IOLTS $A$ and $\Sigma \subseteq L^O$, the friendly $\Sigma$-hiding of $A$, denoted by $\hat{h}_\Sigma(A)$, is an $E$-reachable fragment of $h_\Sigma(A)$, where $E$ is the maximal friendly environment for $h_\Sigma(A)$.

The specifications $S_1 \otimes S_2$ and $\hat{h}_\Sigma(S_1 \otimes S_2)$ depicted in Figure 2(a) and (b) represent friendly composition of $S_1$ and $S_2$, followed by the friendly hiding of the synchronization actions $\Sigma = \{\text{mteel, mcoffee, mcoffeemilk, done}\}$.

We note that for receptive models, the friendly composition and friendly hiding coincide with the parallel composition and hiding. We state the main technical contributions of the paper that ioco relation is preserved under friendly composition and friendly hiding.

Theorem 3: Given two compatible R-IOLTS $I_1$ and $I_2$ and two compatible IOLTS $S_1$ and $S_2$, we have

$I_1 \text{ioco} S_1 \otimes I_2 \text{ioco} S_2 \Rightarrow (I_1 \otimes I_2) \text{ioco} (S_1 \otimes S_2)$.

Theorem 4: Given a R-IOLTS $I$ and an IOLTS $S$ defined over the set $L^O$ of output labels and $\Sigma \subseteq L^O$, we have

$I \text{ioco} S \Rightarrow \hat{h}_\Sigma(I) \text{ioco} \hat{h}_\Sigma(S)$.

Corollary 1: Given two compatible R-IOLTS $I_1$ and $I_2$, two compatible IOLTS $S_1$ and $S_2$, and $\Sigma \subseteq L^O_1 \parallel L^O_2$, we have

$I_1 \text{ioco} S_1 \otimes I_2 \text{ioco} S_2 \Rightarrow \hat{h}_\Sigma(I_1 \otimes I_2) \text{ioco} \hat{h}_\Sigma(S_1 \otimes S_2)$.

C. Computing Friendly Composition and Hiding

In this section, we present the algorithms for effectively computing the friendly composition and hiding. We first create the deterministic maximal friendly environment $E$ for an IOLTS $A$. It is constructed by swapping input and output labels of $A$ and applying a variant of the subset construction that determines the IOLTS afterwards.

Our variant of the subset construction works as follows. Consider an IOLTS $A$, the standard subset construction $C$ of $A$, and a state $S$ of $C$. When $C$ is in $S$, the maximal environment must accept all output transitions from $S$. In contrast, the environment provides an input to $C$ only if all states in $S$...
can accept it. As we can see, $E$ quantifies existentially over outputs in $A$ and universally over inputs in $A$.

**Definition 11 (Maximal deterministic friendly environment):**
The maximal deterministic friendly environment of an IOLTS $A = (Q, L^1, L^0, T, \bar{q})$ is an IOLTS $E_{\text{max}}(A) = (2^Q, L^0, L^1, T_E, \bar{q})$, where $Q = \bar{q}$ after $\epsilon$, and $(S, \alpha, S') \in T_E$ if: 1) $S' = S$ after $\alpha$; 2) $\alpha \in L^1$ implies that $q \not\in A$ for all $q \in S$; and 3) $\alpha \in L^0$ implies that there exists $q \in S$ such that $q \not\in A$.

**Algorithm 1** Friendly composition

**Input:** composable IOLTS $A_1$ and $A_2$

**Output:** $A_1 \otimes A_2$ or not compatible

$E \leftarrow E_{\text{max}}(A_1 \parallel A_2)$

$A_{\text{mb}} \leftarrow$ states in $E$ that contain an ambiguous s. of $A_1 \parallel A_2$

$\text{Prune} \leftarrow \emptyset$

for all $S \in A_{\text{mb}}$

$\text{Prune} \leftarrow \text{Prune} \cup \{\text{state } S' \text{ in } E \mid \exists r \in L^0_{A_1A_2} \; : \; S' \not\in \text{Prune} \}$

remove states in $\text{Prune}$ from $E$

if $E$ has no initial state then return not compatible

else return $E$-reachable fragment of $(A_1 \parallel A_2)$

**Algorithm 1** constructs the friendly composition for IOLTS $A_1$ and $A_2$ or returns the information that they are not compatible. First, it constructs the maximal deterministic friendly environment $E_{\text{max}}(A_1 \parallel A_2)$. The algorithm then computes the set $A_{\text{mb}}$ of states in $E_{\text{max}}(A_1 \parallel A_2)$ that contain an ambiguous state from $A_1 \parallel A_2$ and prunes away all states in $E_{\text{max}}(A_1 \parallel A_2)$ that reach $A_{\text{mb}}$ by a trace of output labels in $A_1 \parallel A_2$. If the initial state is removed in the process, then no friendly environment for $A_1 \parallel A_2$ exists. Otherwise, $E$ is the resulting maximal composition-friendly environment for $A_1 \parallel A_2$ and the $E$-reachable fragment of $A_1 \parallel A_2$ is their friendly composition. The friendly composition $A_1 \otimes A_2$ constructed by the algorithm is of size at most $|A_1| \cdot |A_2|$. If $A_1$ and $A_2$ are both deterministic, then $A_1 \otimes A_2$ is of size $|A_1| \cdot |A_2|$. The alternating bit protocol is a data transfer protocol supporting re-transmission of lost or corrupted messages. For the sake of simplicity, we assume a perfect link (no corruption of messages). The protocol consists of a sender $A$ and a receiver $B$. Initially, the sender $A$ waits for data to be transmitted ($\text{Put?}$). Upon receiving the data, it sends it to $B$ together with a sequence bit ($\text{Data!}$) and waits for an acknowledgment ($\text{Ack0?}$). If the acknowledgment does not arrive before a timeout, $A$ re-transmits the data. This behavior is modeled by a $\tau$-transition, which abstracts away the actual time-out value. Once $A$ receives the acknowledgment, it flips the sequence bit and repeats the procedure ($\text{Data!}$ and $\text{Ack1?}$).

Receiver $B$ behaves in a similar way. It first waits for data marked with the sequence bit ($\text{Data0?}$) and possibly takes a time-out transition. Upon receiving the data, it sends it to the external environment ($\text{Put!}$) and, in the next step, sends an acknowledgment marked with the same sequence bit to the sender ($\text{Data0!}$). The receiver then repeats the procedure with the flipped sequence bit.

Sender $A$ and receiver $B$ (cf. Figure 8) are not compatible, i.e., no friendly environment guarantees the correctness of the protocol. There is a number of composition-ambiguous states in the parallel composition of $A$ and $B$, mainly due to the time-out ($\tau$) transitions. For instance, $A$ can be either in state 1 or 2 after reading the trace $\text{Put!} \cdot \text{Data0!}$. In state 2, $A$ expects the input $\text{Ack0?}$ which is not the case in state 1, where $A$ is ready to re-transmit $\text{Data0!}$ and is brought to after a time-out. Similar problems occur with the receiver $B$ due to its own time-out transitions.

**Algorithm 2** Friendly hiding

**Input:** IOLTS $A$, $\Sigma \subseteq L^0_A$

**Output:** $h_{\Sigma}(A)$

$E \leftarrow E_{\text{max}}(h_{\Sigma}(A))$

return $E$-reachable fragment of $h_{\Sigma}(A)$

**Algorithm 2** computes the friendly $\Sigma$-hiding of an IOLTS $A$, where $\Sigma \subseteq L^0_A$. To obtain the maximal deterministic friendly environment, we simply hide actions in $A$, determine it and then construct its maximal friendly environment $E$. The friendly hiding $h_{\Sigma}(A)$ computed by the algorithm is of size $2^{|A|}$ and if $A$ is deterministic, then $h_{\Sigma}(A)$ is of size $|A|$. We note that there always exists a friendly environment $E$ for arbitrary $A$. It suffices that $E$ accepts all outputs from $A$ and does not provide any inputs.

**V. Evaluation**

To evaluate our approach to compositional testing in the $\text{JiOCo}$-theory, we implemented a proof-of-concept tool for computing friendly composition and hiding. We applied the tool to the alternating bit protocol taken from the CADP examples repository [7]. We were interested in the quality of the component specifications with respect to compositional testing. We also compared the size of the friendly composition of the component specifications to the size of the specification obtained by the demonic completion approach.

The alternating bit protocol is a data transfer protocol supporting re-transmission of lost or corrupted messages. For the sake of simplicity, we assume a perfect link (no corruption of messages). The protocol consists of a sender $A$ and a receiver $B$. Initially, the sender $A$ waits for data to be transmitted ($\text{Put?}$). Upon receiving the data, it sends it to $B$ together with a sequence bit ($\text{Data0!}$) and waits for an acknowledgment ($\text{Ack0?}$). If the acknowledgment does not arrive before a timeout, $A$ re-transmits the data. This behavior is modeled by a $\tau$-transition, which abstracts away the actual time-out value. Once $A$ receives the acknowledgment, it flips the sequence bit and repeats the procedure ($\text{Data1!}$ and $\text{Ack1?}$).

Receiver $B$ behaves in a similar way. It first waits for data marked with the sequence bit ($\text{Data0?}$) and possibly takes a time-out transition. Upon receiving the data, it sends it to the external environment ($\text{Put!}$) and, in the next step, sends an acknowledgment marked with the same sequence bit to the sender ($\text{Data0!}$). The receiver then repeats the procedure with the flipped sequence bit.

Sender $A$ and receiver $B$ (cf. Figure 8) are not compatible, i.e., no friendly environment guarantees the correctness of the protocol. There is a number of composition-ambiguous states in the parallel composition of $A$ and $B$, mainly due to the time-out ($\tau$) transitions. For instance, $A$ can be either in state 1 or 2 after reading the trace $\text{Put!} \cdot \text{Data0!}$. In state 2, $A$ expects the input $\text{Ack0?}$ which is not the case in state 1, where $A$ is ready to re-transmit $\text{Data0!}$ and is brought to after a time-out. Similar problems occur with the receiver $B$ due to its own time-out transitions.
we obtain specifications of smaller size than by demonically completing the component specifications and then applying parallel composition and hiding. This is in particular visible when comparing the size of $h_\Sigma(d(A) \parallel d(B))$ (76 transitions and 34 states), to the one of $h_\Sigma(A'\otimes B')$ (24 transitions and 12 states). We note that in our approach, only foreseen interactions between components are taken into account, which is not the case with the demonic completion approach. While our framework may require manual improvement of the component specifications, we argue that this is the right procedure to arrive at specifications of good quality for compositional testing. Although more automated, the demonic completion approach to compositional testing admits many useless implementations.

To summarize, the case study shows that the alternating bit protocol specification was not modeled with compositional testing in mind. Parts of the sender and receiver specifications are not sufficiently specified for cooperative interactions and do not admit compositional testing with your approach. We improved the specifications by strengthening the assumptions where needed. We note that despite the strengthening, the improved specifications are not input-enabled.

We finally remark that although the individual component specifications are strongly convergent, their composition with
VI. RELATED WORK

This paper is inspired by [3] and extends it by defining new composition and hiding operations adapted for underspecified models and preserving compositional properties of $\text{ioco}$. Compositional properties of the real-time conformance relation $\text{tioco}$ were studied in [8]. In order to preserve compositional testing with $\text{tioco}$, specifications are required to be receptive. Compositional properties of the $\text{cspio}$ conformance relation for model-based testing with CSP specifications were studied in [9]. CSP operations are shown to be monotonic with respect to $\text{cspio}$ when the specifications are input-enabled, or the implementations are not receptive, and after each trace, the input actions accepted by the implementation are a subset of those offered by the specification. The compositional testing problem for systems modeled as networks of abstract components, based on coalgebraic definitions, was considered in [10]. Once again, specifications must be receptive to preserve compositional properties in testing. Assume-guarantee reasoning is combined with $\text{ioco}$ in [11] in order to allow compositional testing. This work is complementary to ours, as it starts from a global specification of the complete system, and uses assumptions about components to divide and conquer the testing process. A methodology to reduce the efforts of integration testing is presented in [12]. It combines model-based integration with model-based testing, but does not provide formal arguments that support the proposed approach.

This paper is also inspired by the interface theories [4], [5]. In contrast to interface automata, used in the context of contract-based design, this work focuses on compositional properties in testing. Instead of iterative design through stepwise refinement, the $\text{ioco}$-theory assumes the existence of an implementation. We consider $\text{ioco}$ with its explicit treatment of quiescence as the refinement relation, rather than alternation simulation used in interface automata. The integration of specifications in the $\text{ioco}$-theory separates parallel composition from hiding, thus allowing for multicast and broadcast communication. In interface automata, parallel composition and hiding are combined into a single operation, thus allowing point-to-point communication only. We also mention similar frameworks in contract-based design: synchronous interfaces [14], and real-time interfaces [15], [16].

VII. CONCLUSION AND FUTURE PERSPECTIVES

We proposed a novel approach to compositional testing for $\text{ioco}$ based on friendly composition and hiding. Our framework characterizes foreseen interactions between components and minimizes the effort needed for integration testing. In addition to the technical results, this paper gives new insights to compositional testing in general and the associated difficulties. In particular, we use our approach to provide guidelines for identifying weaknesses in component specifications and improving them with compositional testing in mind. Since high-level specifications are typically much smaller than the actual implementations, we argue that this additional effort in model analysis is rewarded with a reduction of effort in expensive integration testing.

In our framework, we assume that the composition of component specifications is the specification of the integrated system. In the future, we will study how to exploit our results when the overall system is specified with a separate model. In particular, we will investigate whether our results can be combined with [11]. In addition, we will study whether we can weaken the notion of the ambiguous states, while preserving compositional properties of $\text{ioco}$. Although we considered asynchronous models and $\text{ioco}$-theory, we are confident that our results can be adapted to different modeling frameworks and conformance relations. In fact, many issues related to compositional testing come from the power of the IOLTS model, where components compete in executing the actions without much restrictions. We will adapt our work to the synchronous data-flow systems, which we believe have more robust properties with respect to composition and hiding.

ACKNOWLEDGMENT

This research was funded in part by the European Research Council (ERC) under grant agreement 267989 (QUAREM), by the ARTEMIS IU under grant agreement numbers 269335 (MBAT) and 295373 (nSafeCer), and by the Austrian Science Fund (FWF) project S1402-N23 (RISE).

REFERENCES


APPENDIX A
Proofs

Lemma 1: For any two compatible IOLTs $A$ and $B$ there exists a maximal composition-friendly environment for $A \parallel B$.

Proof: Consider compatible IOLTs $A$ and $B$. By definition $A \parallel B$ have at least one friendly environment. Suppose that there is no maximal composition-friendly environment for $A \parallel B$. This implies that there are two composition-friendly environments, say $E_1$ and $E_2$, such that every composition-friendly environment $E$ satisfies:

$$\text{Traces}(E) \not\supseteq \text{Traces}(E_1) \cup \text{Traces}(E_2).$$  \hspace{1cm} (1)

W.l.o.g. assume that $E_1$ and $E_2$ have pairwise disjoint sets of states, except for their initial state, which is the same. We construct the IOLTs $E_{lc} = (Q_{lc}, L^1, L^0, T_{lc}, q_{lc})$, s.t.

- $Q_{lc} = Q_{E_1} \cup Q_{E_2}$,
- $T_{lc} = T_{E_1} \cup T_{E_2}$,
- $\hat{q}_{lc} = \hat{q}_{E_1} = \hat{q}_{E_2}$.

$E_{lc}$ starts in the shared initial state and then behaves like $E_1$ or $E_2$, so $\text{Traces}(E) = \text{Traces}(E_1) \cup \text{Traces}(E_2)$.

Suppose that $E_{lc}$ is not a composition friendly environment for $A \parallel B$. This means that there exists a state $s = ((q_A, q_B), q_E)$ reachable in $(A \parallel B) \parallel E_{lc}$, such that $s$ is ambiguous in $(A \parallel B) \parallel E$ or $(q_A, q_B)$ is ambiguous in $A \parallel B$. State $s$ is also reachable in $(A \parallel B) \parallel E_1$ or $(A \parallel B) \parallel E_2$, hence either $E_1$ or $E_2$ would not be a composition-friendly environment for $A \parallel B$.

By contradiction, $E_{lc}$ is a friendly environment for $A \parallel B$.

We conclude that $E_{lc}$ is a composition-friendly environment for $A \parallel B$. It violates assumption $[\Pi]$, so by contradiction there exists a maximal friendly environment for $A \parallel B$. \hfill □

Lemma 2: The following two statements hold:

1) For any two R-IOLTS $A$ and $B$, $\text{Straces}(A \otimes B) = \text{Straces}(A \parallel B)$.
2) For a R-IOLTS $A$ and $\Sigma \subseteq L_A^0$ we have $\text{Straces}(h_{\Sigma}(A)) = \text{Straces}(h_{\Sigma}(A))$.

Proof: First we prove an auxiliary result. Let $A$ be an R-IOLTS, $E$ be its maximal friendly environment and $A_E$ be the $E$-reachable fragment of $A$. By definition environment $E$ accepts all outputs issued by $A$. Moreover, since $A$ is receptive and $E$ is maximal, every state of the environment provides all inputs to $A$. As a consequence, $E$ does not restrict $A$, so $\text{Straces}(A_E) = \text{Straces}(A)$.

We prove statement 1). The parallel composition $A \parallel B$ is also an R-IOLTS. $A$ and $B$ are receptive, so there are no ambiguous states in $A \parallel B$. As a consequence, the maximal friendly environment $E$ of $A \parallel B$ is also a composition-friendly environment $E$ of $A \parallel B$. The friendly composition $A \otimes B$ is the $E$-reachable fragment of $A \parallel B$, so by our auxiliary result $\text{Straces}(A \otimes B) = \text{Straces}(A \parallel B)$.

Now we prove statement 2). Since $A$ is a receptive module, then so is $h_{\Sigma}(A)$. Let $E$ be a maximal friendly environment for $h_{\Sigma}(A)$. The friendly composition $h_{\Sigma}(A)$ is the $E$-reachable fragment of $h_{\Sigma}(A)$, so by our auxiliary result $\text{Straces}(h_{\Sigma}(A)) = \text{Straces}(h_{\Sigma}(A))$. \hfill □

Lemma 3: Let $A$ and $B$ be two compatible IOLTS. Then

$$\forall \sigma \in \text{Straces}(A \otimes B) :$$

$$((q_A, q_B), q_E) \in A \otimes B \text{ after } \sigma \quad \Rightarrow \quad (q_A \in A \text{ after } \sigma \downarrow_{L_A \cup \delta} \text{ and } q_B \in B \text{ after } \sigma \downarrow_{L_B \cup \delta}).$$

Proof: Proof is by induction on the length of $\sigma$.

For $\sigma = \epsilon$ the property holds, because by definition $((\hat{q}_A, \hat{q}_B), \hat{q}_E)$ is the initial state of $A \otimes B$ and any transition $((\hat{q}_A, \hat{q}_B), \hat{q}_E) \xrightarrow{\delta} \hat{q}_A \otimes B_A \hat{q}_A$ can be matched by transitions $\hat{q}_A \xrightarrow{\delta} A_A q_A$. Assume that the property holds for trace $\sigma$ and suppose that $((q_A, q_B), q_E) \in A \otimes B \text{ after } \sigma \alpha$. Then $q_E$ is reachable in $E \parallel (A \parallel B)$, so by induction there exists a friendly environment $E'$ that satisfies $q_E \in E' \parallel (A \parallel B)$.
Then there exists a predecessor state \(((q'_A, q'_B), q'_E) \in A \otimes B\) after \(\sigma\) s.t.
\[
((q'_A, q'_B), q'_E) \xrightarrow{\alpha \rightarrow A \otimes B_i} ((q''_A, q''_B), q''_E) \xrightarrow{\delta \rightarrow A \otimes B_i} ((q_A, q_B), q_E)
\]
By the induction hypothesis
\[
q'_A \in A \text{ after } \sigma_{\downarrow \downarrow \downarrow \downarrow \downarrow L_A \cup \{\delta\}} \quad q'_B \in B \text{ after } \sigma_{\downarrow \downarrow \downarrow \downarrow \downarrow L_B \cup \{\delta\}}.
\]
We consider different cases for \(\alpha\). Suppose that \(\alpha\) is local to \(A\), i.e. \(\alpha \in L_A \setminus L_B\). Then the transitions
\[
q'_A \xrightarrow{\alpha \rightarrow A_i} q''_A \xrightarrow{\delta \rightarrow A_i} q_A
\]
is possible by the definition of friendly composition, which gives \(q_A \in A\) after \(\sigma \alpha_{\downarrow \downarrow \downarrow \downarrow \downarrow L_A \cup \{\delta\}}\) as required. From the observations
\[
\sigma \alpha_{\downarrow \downarrow \downarrow \downarrow \downarrow L_B \cup \{\delta\}} = \sigma_{\downarrow \downarrow \downarrow \downarrow \downarrow L_B \cup \{\delta\}} \quad q'_B = q_B \quad q''_B \xrightarrow{\delta \rightarrow B_i} q_B
\]
we get \(q_B \in B\) after \(\sigma \alpha_{\downarrow \downarrow \downarrow \downarrow \downarrow L_B \cup \{\delta\}}\) so the property holds. The argument for \(\alpha \in L_B \setminus L_A\) is symmetric.

Suppose that \(\alpha \in L_A \cap L_B\). Then transitions
\[
q'_A \xrightarrow{\alpha \rightarrow A_i} q''_A \xrightarrow{\delta \rightarrow A_i} q_A \quad q'_B \xrightarrow{\alpha \rightarrow B_i} q''_B \xrightarrow{\delta \rightarrow B_i} q_B
\]
are possible, so the lemma holds.

Let \(\alpha = \delta\). We show that \(q'_A, q'_B\) are quiescent in \(A\) and \(B\), respectively. Suppose that \(q'_A\) is not quiescent in \(A\). If \(q'_A\) had an outgoing \(\tau\)-transition, then state \(((q'_A, q'_B), q'_E)\) would also have a \(\tau\)-transition in \(A \otimes B\), which contradicts that assumption that \(((q'_A, q'_B), q'_E)\) is quiescent. Similarly, if \(q'_A\) had some output action, then by definition of a friendly environment state \(((q'_A, q'_B), q'_E)\) would also have this action, so it would not be quiescent. By contradiction \(q'_A\) is quiescent in \(A\) and the same argument shows that \(q'_B\) is quiescent in \(B\). This implies that the following transitions are possible
\[
q'_A \xrightarrow{\delta \rightarrow A_i} q'_A = q_A \quad q'_B \xrightarrow{\delta \rightarrow B_i} q'_B = q_B
\]
so the lemma holds.

\[\blacksquare\]

**Lemma 4:** Let \(A\) and \(B\) be two compatible IOLTs. Then
\[
\forall \sigma \in \text{Traces}(A \otimes B) : (q_A \in A \text{ after } \sigma_{\downarrow \downarrow \downarrow \downarrow \downarrow L_A \cup \{\delta\}} \land q_B \in B \text{ after } \sigma_{\downarrow \downarrow \downarrow \downarrow \downarrow L_B \cup \{\delta\}}) \implies \exists q_E \in Q_E : ((q_A, q_B), q_E) \in A \otimes B \text{ after } \sigma.
\]
where \(E\) is the friendly environment used in construction of \(A \otimes B\).

**Proof:** We first prove an auxiliary lemma. Let \(A\) and \(B\) be two compatible IOLTs and \(\sigma\) be an \(s\)-trace in \(A \otimes B\). Then
\[
\forall (\cdot, q_E) \in A \otimes B \text{ after } \sigma : (q_A \in A \text{ after } \sigma_{\downarrow \downarrow \downarrow \downarrow \downarrow L_A \cup \{\delta\}} \land q_B \in B \text{ after } \sigma_{\downarrow \downarrow \downarrow \downarrow \downarrow L_B \cup \{\delta\}} \implies ((q_A, q_B), q_E) \in A \otimes B \text{ after } \sigma.
\]
where \(E\) is the friendly environment used in construction of \(A \otimes B\).

Assume that the auxiliary lemma holds. If \(\sigma \in \text{Traces}(A \otimes B)\), then there exists some \((\cdot, q_E) \in A \otimes B \text{ after } \sigma\). Lemma 4 follows from the auxiliary lemma.

Proof of the auxiliary lemma is by induction on the length of \(\sigma\). For the induction base let \(\sigma = \epsilon\). Assume that \(((s_A, s_B), q_E) \in A \otimes B \text{ after } \epsilon\). This means that \(A \otimes B\) has a transition
\[
((q_A, q_B), q_E) \xrightarrow{\delta \rightarrow A \otimes B_i} ((s_A, s_B), q_E).
\]
\(\tau\)-transitions in \(A \otimes B\) arise from independent \(\tau\)-transitions in \(A, B\) and \(E\), so \(q_E \xrightarrow{\delta \rightarrow E} q_E\). Let \(q_A \in A \text{ after } \epsilon\) and \(q_B \in B \text{ after } \epsilon\).

Then by definition of friendly composition \(((q_A, q_B), q_E) \xrightarrow{\delta \rightarrow A \otimes B_i} ((q_A, q_B), q_E)\).

For the induction step assume that the property holds for trace \(\sigma\). Consider arbitrary
\[
((s_A, s_B), q_E) \in A \otimes B \text{ after } \sigma_\alpha
\]
and assume that
\[
q_A \in A \text{ after } \sigma_{\alpha_{\downarrow \downarrow \downarrow \downarrow \downarrow L_A \cup \{\delta\}}} \quad q_B \in B \text{ after } \sigma_{\alpha_{\downarrow \downarrow \downarrow \downarrow \downarrow L_B \cup \{\delta\}}}.
\]
State \((s_A, s_B), q_E)\) has a predecessor state \(((s'_A, s'_B), q'_E)\) in \(A \otimes B\) after \(\sigma\) s.t.:

\[
((s'_A, s'_B), q'_E) \xrightarrow{\alpha} A \otimes B \quad ((s_A, s_B), q_E) \xrightarrow{\delta} A \otimes B_s \quad ((s_A, s_B), q_E).
\]

(2)

We consider possible cases for \(\alpha\). Let \(\alpha \in L_A \setminus L_B\). There exists a predecessors \(q'_A\) of \(q_A\):

\[
q'_A \in A \text{ after } \sigma_{\downarrow L_A \cup \{\delta\}} \quad \text{s.t. } q'_A \xrightarrow{\alpha} A_s \quad q_A \xrightarrow{\delta} A_s q_A.
\]

(3)

From \(\sigma_{\downarrow L_B \cup \{\delta\}} = \sigma_{\downarrow L_B \cup \{\delta\}}\) it follows that \(q_B \in B\) after \(\sigma_{\downarrow L_B \cup \{\delta\}}\). Then, by the induction hypothesis:

\[
\forall (i, q) \in A \otimes B \text{ after } \sigma : ((q'_A, q_B), q) \in A \otimes B \text{ after } \sigma.
\]

Instantiating \(q\) with \(q'_E\) gives

\[
((q'_A, q_B), q'_E) \in A \otimes B \text{ after } \sigma.
\]

From (2) and the definition of friendly composition we get:

\[
q'_E \xrightarrow{\alpha} E \quad q'_E \xrightarrow{\delta} E \quad q_E.
\]

(4)

By (3), (4) and the definition of friendly composition state \(((q'_A, q_B), q'_E)\) can execute action \(\alpha\) in \(A \otimes B\) and then any following silent transitions that lead to \(((q_A, q_B), q_E)\):

\[
((q'_A, q_B), q'_E) \xrightarrow{\alpha} A \otimes B_s \quad ((q_A, q_B), q_E)\]

This gives \(((q_A, q_B), q_E)\) \(A \otimes B\) after \(\sigma\) as required. Symmetrical argument holds for \(\alpha \in L_B \setminus L_A\).

Let \(\alpha \in L_A \cap L_B\). States \(q_A\) and \(q_B\) have some predecessors \(q'_A\), \(q'_B\):

\[
q'_A \in A \text{ after } \sigma_{\downarrow L_A \cup \{\delta\}} \quad \text{s.t. } q'_A \xrightarrow{\alpha} A_s q_A.
\]

(5)

\[
q'_B \in B \text{ after } \sigma_{\downarrow L_B \cup \{\delta\}} \quad \text{s.t. } q'_B \xrightarrow{\alpha} B_s q_B.
\]

(6)

It follows from (2) that

\[
q'_E \xrightarrow{\alpha} E \quad q'_E \xrightarrow{\delta} E \quad q_E.
\]

(7)

Then by the induction hypothesis we get:

\[
((q'_A, q_B), q'_E) \in A \otimes B \text{ after } \sigma.
\]

By (7), (5), (6) and the definition of friendly composition the following transition is possible in \(A \otimes B\):

\[
((q'_A, q'_B), q'_E) \xrightarrow{\alpha} A \otimes B_s \quad ((q_A, q_B), q_E)\]

Let \(\alpha = \delta\). We note that in (2) \(q'_E = q^*_E = q_E\). State \(q_E\) cannot have any outgoing \(\tau\)-transitions, because otherwise \(((q'_A, q'_B), q'_E)\) would not be quiescent. States \(q_A, q_B\) are quiescent, therefore:

\[
q_A \in A \text{ after } \sigma_{\downarrow L_A \cup \{\delta\}} \quad \text{and } q_A \xrightarrow{\delta} A_s q_A.
\]

\[
q_B \in B \text{ after } \sigma_{\downarrow L_B \cup \{\delta\}} \quad \text{and } q_B \xrightarrow{\delta} B_s q_B.
\]

By the induction hypothesis

\[
((q_A, q_B), q_E) \in A \otimes B \text{ after } \sigma.
\]

State \(((q_A, q_B), q_E)\) cannot have any \(\tau\)-transitions, because \(q_A, q_B\) are quiescent and \(q_E\) does not have such transitions. Any output transitions in \(A \otimes B\) arise from synchronization of \(A\) or \(B\) with the friendly environment. States \(q_A, q_B\) are quiescent, hence \(((q_A, q_B), q_E)\) does have any output actions. It follows that \(((q_A, q_B), q_E)\) is quiescent in \(A \otimes B\):

\[
(q_A, q_B, q_E) \xrightarrow{\delta} A \otimes B_s \quad ((q_A, q_B), q_E).
\]

Since \(((s_A, s_B), q_E)\) was arbitrary, so the auxiliary lemma holds for any \(((s_A, s_B), q_E)\) \(A \otimes B\) after \(\sigma\).

\[
\square
\]

**Theorem 3:** Given two compatible R-IOLTS \(I_1\) and \(I_2\) and two compatible IOLTS \(S_1\) and \(S_2\), we have

\[
I_1 \text{ioco } S_1 \land I_2 \text{ioco } S_2 \rightarrow (I_1 \otimes I_2) \text{ioco } (S_1 \otimes S_2).
\]

**Proof:** Let \(E_S, E_I\) be the friendly environments used in construction of \(S_1 \otimes S_2\) and \(I_1 \otimes I_2\), respectively. Assume that \(I_1\) ioco \(S_1\) and \(I_2\) ioco \(S_2\). Let \(\sigma\) be an arbitrary suspension trace in \(S_1 \otimes S_2\) and let

\[
P = S_1 \otimes S_2 \text{ after } S_1 \otimes S_2 \sigma \quad R = I_1 \otimes I_2 \text{ after } I_1 \otimes I_2 \sigma.
\]

Suppose that \(\alpha \in \text{out}_{I_1 \otimes I_2}(R)\). We need to show that \(\alpha \in \text{out}_{S_1 \otimes S_2}(P)\).
Consider an arbitrary state \( ((q_{I_1}, q_{I_2}), q_E) \in R \) such that
\[
\alpha \in \text{out}_{I_1 \odot I_2}(((q_{I_1}, q_{I_2}), q_E)).
\]

Consider different cases for \( \alpha \).

Let \( \alpha \in L_1^O \setminus L_2^I \). By Lemma 3 we have that
\[
q_{I_1} \in I_1 \text{ after } \sigma_{L_1 \cup \{\delta\}} \quad \sigma_{L_1 \cup \{\delta\}} \in \text{Straces}(S_1).
\]

By the definition of friendly composition \( \alpha \in \text{out}_{I_1}(q_{I_1}) \). S-trace \( \sigma_{L_1 \cup \{\delta\}} \) is shared between \( S_1 \) and \( I_1 \), so from the assumption \( I_1 \text{ loco } S_1 \) it follows that:
\[
\alpha \in \text{out}(S_1 \text{ after } \sigma_{L_1 \cup \{\delta\}}),
\]
so there exists a state
\[
q_{S_1} \in S_1 \text{ after } \sigma_{L_1 \cup \{\delta\}}
\]
\( \text{s.t. } \alpha \in \text{out}_{S_1}(q_{S_1}) \). By Lemma 3 there exists a state
\[
q_{S_2} \in S_2 \text{ after } \sigma_{L_2 \cup \{\delta\}}
\]
by Lemma 4
\[
\exists q_E \in Q_{E\delta} : ((q_{S_1}, q_{S_2}), q_E) \in S_1 \otimes S_2 \text{ after } \sigma.
\]

In composition \( S_1 \parallel S_2 \) state \( (q_{S_1}, q_{S_2}) \) has an outgoing action \( \alpha \). Friendly environment \( E \) must accept on \( \alpha \), so \( \alpha \in \text{out}_{A \cup B}(((q_{S_1}, q_{S_2}), q_E)) \) as required.

Let \( \alpha \in L_2^O \setminus L_1^I \). The proof is symmetric to the previous case.

Let \( \alpha \in L_1^O \wedge \alpha \in L_2^I \). Following the same argument as in the first case, we get
\[
\exists q_E \in Q_{E\delta} : ((q_{S_1}, q_{S_2}), q_E) \in S_1 \otimes S_2 \text{ after } \sigma.
\]
\( \text{s.t. } \alpha \in \text{out}(S_1 \text{ after } \sigma_{L_1 \cup \{\delta\}}) \). In \( S_1 \parallel S_2 \) state \( q_{S_2} \) must accept \( \alpha \), because otherwise \( ((q_{S_1}, q_{S_2}), q_E) \) would be an ambiguous state in \( S_1 \parallel S_2 \) and \( E \) would not be a friendly environment. By definition of a friendly environment \( E \) must synchronize on \( \alpha \), so \( \alpha \in \text{out}_{A \cup B}(((q_{S_1}, q_{S_2}), q_E)) \).

Let \( \alpha \in L_2^O \wedge \alpha \in L_1^I \). The proof is symmetric to the previous case.

Let \( \alpha = \delta \). We note that state \( ((q_{I_1}, q_{I_2}), q_E) \) is quiescent \( I_1 \otimes I_2 \). We show that \( q_{I_1}, q_{I_2} \) are quiescent in \( I_1 \) and \( I_2 \). If \( q_{I_1} \) or \( q_{I_2} \) had an output transition, then \( ((q_{I_1}, q_{I_2}), q_E) \) would also have this action by definition of a friendly environment; but then \( ((q_{I_1}, q_{I_2}), q_E) \) would not be quiescent. Similarly, if \( q_{I_1} \) or \( q_{I_2} \) had a \( \tau \)-transition, then state \( ((q_{I_1}, q_{I_2}), q_E) \) would also have \( \tau \)-transition and would not be quiescent. By contradiction \( q_{I_1} \) and \( q_{I_2} \) are quiescent, so
\[
\delta \in \text{out}_{I_1}(q_{I_1}) \quad \delta \in \text{out}_{I_2}(q_{I_2}).
\]

By Lemma 3 and assumption \( I_i \text{ loco } S_i, i = 1, 2 \) we get
\[
\delta \in \text{out}(S_1 \text{ after } \sigma_{L_1 \cup \{\delta\}}) \quad \delta \in \text{out}(S_1 \text{ after } \sigma_{L_1 \cup \{\delta\}}).
\]
This implies that there are states \( q_{S_1}, q_{S_2} \):
\[
q_{S_1} \in S_1 \text{ after } \sigma_{L_1 \cup \{\delta\}} \quad \text{s.t. } \delta \in \text{out}_{S_1}(q_{S_1})
\]
\[
q_{S_2} \in S_2 \text{ after } \sigma_{L_2 \cup \{\delta\}} \quad \text{s.t. } \delta \in \text{out}_{S_2}(q_{S_2})
\]
From Lemma 4 we obtain
\[
\exists q_E \in Q_{E\delta} : ((q_{S_1}, q_{S_2}), q_E) \in S_1 \otimes S_2 \text{ after } \sigma.
\]

Transition systems are allowed to make only finite number of \( \tau \)-transitions from a given state. So there exists \( q_E' \) reachable by \( \tau \)-transitions from \( q_E \) in \( E\delta \), s.t. \( q_E' \) does not have any \( \tau \)-transitions:
\[
q_E \Rightarrow_{E\delta} q_E' \quad \text{s.t. } q_E \not\Rightarrow_{E\delta} \quad q_E' \not\Rightarrow_{E\delta}.
\]
State \( ((q_{S_1}, q_{S_2}), q_E') \) is also reachable by suspension trace \( \sigma \) in \( S_1 \otimes S_2 \). This state cannot have any \( \tau \)-transitions, because none of \( q_{S_1}, q_{S_2} \) or \( q_E' \) has them. Moreover, \( ((q_{S_1}, q_{S_2}), q_E') \) does not have any output actions, because \( q_{S_1}, q_{S_2} \) are quiescent and the friendly environment by itself does not generate output actions in \( S_1 \otimes S_2 \). We conclude that \( ((q_{S_1}, q_{S_2}), q_E') \) is quiescent in \( S_1 \otimes S_2 \), so
\[
\delta \in \text{out}_{S_1 \otimes S_2}(((q_{S_1}, q_{S_2}), q_E'))
\]
and the theorem holds.
Lemma 5: Given an IOLTS $A$ and its friendly environment $E$, it holds that
\[ \forall \sigma \in \straces(A_E), \forall \alpha \in L_A^O \cup \{ \delta \} : \sigma \alpha \in \straces(A) \implies \sigma \alpha \in \straces(A_E), \]
where $A_E$ is the $E$-reachable fragment of $A$.

Proof: Assume that $\sigma \in \straces(A_E)$ and $\sigma \alpha \in \straces(A)$. Then, there is a state $(q_A, q_E) \in A_E$ after $\sigma$ s.t. $q_A \xrightarrow{\alpha} A_s, q'_A$.

Let $\alpha \in L_A^O$. By definition of a friendly environment, state $q_E$ must accept $\alpha$:
\[ \exists q'_E \in Q_E : q_E \xrightarrow{\alpha} q'_E. \]
Then, by definition of friendly composition the following transition is possible in $A_E$:
\[ (q_A, q_E) \xrightarrow{\alpha} A_{E \delta}, (q'_A, q'_E), \]
so $\sigma \alpha \in \straces(A_E)$.

Let $\alpha = \delta$. Then state $q_A$ is quiescent in $A$. By definition a friendly environment, make only a finite number of $\tau$-transitions, so $q_E$ can reach by $\tau$-transitions a state $q'_E$ that has no $\tau$-transitions available:
\[ q_E \xrightarrow{\tau} q'_E \quad q'_E \xrightarrow{\tau} E. \]
Then, state $(q_A, q_E)$ can reach $(q_A, q'_E)$ by $\tau$-transitions in $A_E$:
\[ (q_A, q_E) \xrightarrow{\tau} A_{E \delta}, (q_A, q'_E). \]
State $(q_A, q'_E)$ does not have any output actions, since $q_A$ is quiescent and the friendly environment by itself cannot add output actions to $A_E$. State $(q_A, q'_E)$ also does not have any $\tau$-transitions, because neither of $q_A$ or $q'_E$ has them. This implies that $(q_A, q'_E)$ is quiescent and the following transitions are possible in $A_E$:
\[ (q_A, q_E) \xrightarrow{\tau} A_{E \delta}, (q_A, q'_E) \xrightarrow{\delta} A_{E \delta}, (q_A, q'_E). \]

Theorem 4: Given a R-IOLTS $I$ and an IOLTS $S$ defined over the set $L^O$ of output labels, and $\Sigma \subseteq L^O$, we have
\[ I \ioco S \rightarrow \hat{h}_{\Sigma}(I) \ioco \hat{h}_{\Sigma}(S). \]

Proof: Assume that $I \ioco S$ and $\Delta \in \straces(\hat{h}_\Sigma(S))$. Suppose
\[ \alpha \in \text{out}(\hat{h}_\Sigma(I) \text{ after } \Delta), \]
then we need to show that $\alpha \in \text{out}(\hat{h}_\Sigma(S) \text{ after } \Delta)$.

Let $\pi$ be a suspension trace in $I$ that becomes $\Delta$ after hiding and also leads to action $\alpha$
\[ \pi_{\downarrow(L^O \setminus \Sigma) \cup \{ \delta \}} = \Delta \quad \alpha \in \text{out}(I \text{ after } \pi). \]
Assume towards contradiction that $\pi \notin \straces(S)$.

Let $\pi'$ be the longest prefix of $\pi$ that is an $s$-trace in $S$. Let $\pi'\beta$ be the one action longer prefix of $\pi$. Then $\beta$ cannot be an output action or $\delta$, because by Lemma 5 $\pi'\beta$ would also be a prefix of $S$. It follows that $\beta$ must be some input $i^? \in L_E^I$, so that $\pi \in \straces(S) \quad \pi i^? \notin \straces(S)$.
This implies that there is a state $s$ in $\hat{h}_\Sigma(S)$ after $\Delta'$ that cannot input $i$?
\[ s \in \hat{h}_\Sigma(S) \text{ after } \Delta' \quad \text{s.t.} \quad s \xrightarrow{i^?}. \]
Input actions are not hidden, so $\Delta$ can be written as:
\[ \Delta = \Delta'i^?\Delta'' \quad \text{where} \quad \pi_{\downarrow(L_A \setminus \Sigma) \cup \{ \delta \}} = \Delta'. \]
From the assumption $\Delta \in \hat{h}_\Sigma(S)$ it follows that there exists a state $s'$ reachable after $\Delta'$ that can accept $i$?
\[ s' \in \hat{h}_\Sigma(S) \text{ after } \Delta' \quad \text{s.t.} \quad s' \xrightarrow{i^?}. \]
Let $E$ be the friendly environment used in construction of $\hat{h}_\Sigma(S)$. Consider any state $(s', s_E) \in \hat{h}_\Sigma(S)$, where $s_E$ belongs $E$. Environment does not distinguish between states reachable by the same traces, so $(s, s_E)$ would also be reachable in $\hat{h}_\Sigma(S)$. 
This means that $s_E \nrightarrow h$, because otherwise $(s, s_E)$ would be an ambiguous state and $E$ would not be an environment. This implies that $\Delta' \notin \text{Straces}(h_\Sigma(S))$ which contradicts our assumption, therefore $\pi \in \text{Straces}(S)$.

Since $\pi$ is an an s-trace of $S$, then by assumption $I \text{ ioco } S$ and by the fact $\alpha \in \text{out}(I \text{ after } \pi)$ we get

$$\alpha \in \text{out}(S \text{ after } \pi)$$

After hiding $\pi \alpha$ becomes s-trace $\Delta\alpha$ in $h_\Sigma(S)$, which proves the theorem. ■

**APPENDIX B**

**Utraces, ioco and Compositionality**

In this section, we show that the $\text{ioco}$ relation is not preserved under parallel composition and hiding. We first recall the definition of $\text{Utraces}$, as in [3].

**Definition 12:** Given an IOLTS $A$, we define the set $\text{Utraces}(q)$, where $q \in Q$, as follows:

$$\text{Utraces}(q) = \{\sigma \in (L \cup \{\delta\})^* \mid q \Rightarrow (\exists q', \sigma_1 \cdot \alpha \cdot \sigma_2 = \sigma : \alpha \in L^I \land q \Rightarrow q' \land q' \neq \phi)\}$$

The conformance relation $\text{ioco}$ is defined as follows.

**Definition 13:** Given a R-IOLTS $I$ and an IOLTS $S$, we say that $I \text{ ioco } S$ iff

$$\forall \sigma \in \text{Utraces}(S), \text{ out}(\hat{q}_I \text{ after } \sigma) \subseteq \text{out}(\hat{q}_S \text{ after } \sigma)$$

**Theorem 5:** Given two composable R-IOLTS $I_1$ and $I_2$ and two composable IOLTS $S_1$ and $S_2$, we have

$$I_1 \text{ ioco } S_1 \land I_2 \text{ ioco } S_2 \nrightarrow (I_1 \parallel I_2) \text{ ioco } (S_1 \parallel S_2).$$

**Proof:** In order to show that $\text{ioco}$ is not preserved under parallel composition, we use the same example from [3] depicted in Figure 11, which was used to show non-compositionality of $\text{ioco}$ under parallel composition. Since all the IOLTS $(S_1, S_2, I_1, I_2, S_1 \parallel S_2$ and $I_1 \parallel I_2)$, are deterministic and do not have $\tau$-transitions, there are no under-specified traces in any of the IOLTS and $\text{Utraces}$ coincide with the $\text{Straces}$. Hence, we have that $I_1 \text{ ioco } S_1$ and $I_2 \text{ ioco } S_2$. Now consider the trace $x \in \text{Utraces}(S_1 \parallel S_2)$. We have that $\text{out}(\hat{q}_{S_1} \parallel \hat{q}_{S_2} \text{ after } x) = \{\delta\}$, while $\text{out}(\hat{q}_{I_1} \parallel \hat{q}_{I_2} \text{ after } x) = \{x\}$. It follows that $(I_1 \parallel I_2) \nrightarrow \text{ ioco } (S_1 \parallel S_2)$.

**Theorem 6:** Given a R-IOLTS $I$ and an IOLTS $S$, defined over the alphabet $L$, and a subset $\Sigma \subseteq L^O$, we have

$$I \text{ ioco } S \nrightarrow h_{\Sigma}(I) \text{ ioco } h_{\Sigma}(S).$$

**Proof:** Consider the specification $S$ and the implementation $I$, depicted in Figures 12(a) and (c). It is clear that $I \text{ ioco } S$. Now consider $h_\Sigma(S)$ and $h_\Sigma(I)$ where $\Sigma = \{a\}$ (shown in Figures 12(b) and (d)). We choose the trace $\sigma = i$ and show that it
Fig. 12. Counter-example for preservation of $uioco$ under hiding.

is in $\text{Utraces}(h_\Sigma(S))$. The only decomposition of $\sigma$ of the form $\sigma_1 \cdot \alpha \cdot \sigma_2$ such that $\alpha \in L^I_{h_\Sigma(S)}$ is $\epsilon \cdot i \cdot \epsilon$. Now, after reading the trace $\epsilon$, $h_\Sigma(S)$ can be in either states 1 or 2. We have that both $1 \Rightarrow 3$ and $2 \Rightarrow 3$, hence $i \in \text{Utraces}(h_\Sigma(S))$. We have that $\text{out}(\hat{q}_{h_\Sigma(S)} \text{ after } i) = \{\delta\}$. On the other hand, $\text{out}(\hat{q}_{h_\Sigma(I)} \text{ after } i) = \{\delta, b\}$. It follows that $h_\Sigma(I) \not\equiv_{uioco} h_\Sigma(S)$. 