# Work-in-progress
# Assume-guarantee reasoning with ioco

Laura Brandán Briones* and Corina Păsăreanu+** and Dimitra Giannakopoulou+***

*Faculty of Computer Science. University of Twente.
P.O.Box 217, 7500AE Enschede,
The Netherlands. Fax - (31 53)-489-3247
**QSS ***RIACS +NASA Ames
NASA Ames Research Center
Moffett Field, CA 94035-1000, USA
brandanl@cs.utwente.nl/dimitra,pcorina@email.arc.nasa.gov

**Abstract.** This paper presents a combination between the assume-guarantee paradigm and the testing relation **ioco**. The assume-guarantee paradigm is a "divide and conquer" technique that decomposes the verification of a system into smaller tasks that involve the verification of its components. The principal aspect of assume-guarantee reasoning is to consider each component separately, while taking into account assumptions about the context of the component. The testing relation **ioco** is a formal conformance relation for model-based testing that works on labeled transition systems. Our main result shows that, with certain restrictions, assume-guarantee reasoning can be applied in the context of **ioco**. This enables testing **ioco**-conformance of a system by testing its components separately.

## 1 Introduction

Conformance relations in testing try to identify if a system under test (known as SUT) behaves as expected. We consider formal conformance relations, where the expected *global* system behavior is given as a specification written in a formal language and a mathematical framework is used to assert the correctness criteria (i.e. that the system conforms to the specification). There are two main approaches in testing: "white-box" and "black-box". White-box testing assumes that the tester has the knowledge of the SUT's code (and because of that it is usually used in unit testing). Alternatively, black-box testing assumes that the tester has only access to the SUT through an interface, without any knowledge of the internal parts of the system. In this paper we present a black-box formal testing framework to test components of a system. Therefore, our approach applies to both unit and system testing.

In our approach, components and specifications are modeled as input-output transition systems (IOTS), which formalize descriptions for systems that interact with their environment by receiving inputs and offering outputs. An IOTS that has the input set $\mathcal{L}^I$ and the output set $\mathcal{L}^U$ is denoted as IOTS($\mathcal{L}^I, \mathcal{L}^U$). If an IOTS($\mathcal{L}^I, \mathcal{L}^U$) accepts all inputs in any state, then it is called **input-enabled**. The **ioco** relation (which stands for input output conformance) is a formal conformance relation that asserts when an implementation, that is modeled by an input-enabled IOTS, behaves as expected by a given IOTS specification. The input-enabling assumption of the implementation is known as the test hypothesis [IJW96].

A previous approach [vdBRT03], studies compositionality properties of **ioco** . For two pairs of input-enabled systems $i_1$, $s_1$ in IOTS($\mathcal{L}_1^I, \mathcal{L}_1^U$) and $i_2$, $s_2$ in IOTS($\mathcal{L}_2^I, \mathcal{L}_2^U$) the following compositional rule is proved:

$$\frac{i_1 \textbf{ ioco } s_1 \wedge i_2 \textbf{ ioco } s_2}{i_1||i_2 \textbf{ ioco } s_1||s_2}$$

In this previous framework the global specification is given as a composition of two systems $(s_1||s_2)$.

However, we believe that it is often natural and easier to write a global system specification as a single transition system (rather than to decompose it into subsystems $s_1$ and $s_2$). Moreover, it is often the case that components conform to their specifications only in specific *contexts* (or environments). From here comes our research question: is it possible to use formal testing in a compositional way, using a global specification and taking into account *assumptions* about the environments in which the components are supposed to operate?

Assume-guarantee reasoning is a technique that has long held promise for compositional verification. This technique is a "divide-and-conquer" approach that infers global system properties by checking individual components in isolation [CLM89,GGSV02,Pnu84], and taking into account environment assumptions. In its simplest form, it checks whether a component $M$ guarantees a property $P$ when it is part of a system that satisfies an assumption $A$, and checks that the remaining components in the system ($M$'s environment) satisfy $A$. Extensions that use an assumption for each component in the system also exist. Previous work developed techniques that automatically generate assumptions for performing assume-guarantee model checking at the design level [CGP03,GPB02], ameliorating the often difficult challenge of finding an appropriate assumption.

In this paper we propose a combination of assume-guarantee reasoning with **ioco** as the conformance relation. The idea is to prove that, given an assumption $A$ such that $i_2$ **ioco** $A$ and $i_1||A$ **ioco** $S$ then $i_1||i_2$ **ioco** $S$. As a consequence, if $A$ is provided we can test in isolation components (which may be at different stages of development and possibly written by different developer teams), having a specification of the overall system. Our approach has also the potential to enable reliable component re-use and reliable integration of COTS components.

**Contents** The rest of the paper is organized as follows. Section 2 gives some background on the **ioco** conformance relation and Section 3 describes the parallel composition of IOTSs. Section4 presents assume-guarantee reasoning and the main theorem of the paper. Section 5 presents an example and Section 6 discusses the conclusions.

## 2 Background on the ioco testing relation

This section recalls several aspects of the **ioco** theory, for more details see [Tre96].

**Definition 1.** *An Input-Output Transition System (IOTS) is a 4-tuple $\langle Q, q^0, \mathcal{L}, T\rangle$, where*

- *$Q$ is a countable, non-empty set of states. With $q^0 \in Q$ as the initial state.*
- *$\mathcal{L}$ is a countable set of labels, partitioned into input ($\mathcal{L}^I$) and output ($\mathcal{L}^U$) actions, with $\mathcal{L}^I \cap \mathcal{L}^U = \emptyset$ and $\mathcal{L}^I \cup \mathcal{L}^U = \mathcal{L}$.*
- *$T \subseteq (Q \times (\mathcal{L} \cup \{\tau\}) \times Q)$ is the transition relation.*

We use a special label $\tau \notin \mathcal{L}$ to denote an internal action. For an arbitrary $L \in \mathcal{L}$, we use $L_\tau$ as a shorthand for $L \cup \{\tau\}$. For a system $p$, we use $Q_p$, $\mathcal{L}_p$, etc. to denote the components

of $p$. We write $q \xrightarrow{a} q'$ in the case $(q, a, q') \in T$. We use question mark "?" after a label to denote an input action and exclamation mark "!" to denote an output. We denote the class of all labeled transition systems over $\mathcal{L}^I$ and $\mathcal{L}^U$ by $IOTS(\mathcal{L}^I, \mathcal{L}^U)$.

A state that cannot do an internal action is called *stable*. A state that cannot do an output or internal action is called *quiescent*. We use the symbol $\delta$ (with $\delta \notin \mathcal{L}_\tau$) to represent quiescence. For an arbitrary $L \subseteq \mathcal{L}_\tau$, we use $L_\delta\tau$ as a shorthand for $L \cup \{\delta\}$. An *IOTS* is called **strongly responsive** if it always eventually enters a quiescent state; in other words, if it does not have any infinite $\mathcal{L}_\tau^U$-labeled paths.

A *trace* is a finite sequence of observable actions. The set of all traces over $L$ (with $L \subseteq \mathcal{L}$) is denoted by $L^*$; a trace in $L^*$ is denoted by $\sigma$, with $\epsilon$ denoting the empty sequence. If $\sigma_1, \sigma_2 \in L^*$, then $\sigma_1 \cdot \sigma_2$ is the concatenation of $\sigma_1$ and $\sigma_2$. We use the standard notation with single and double arrows for traces: $q \xrightarrow{a_1 \cdots a_n} q'$ denotes , $q \xrightarrow{a_1} \cdots \xrightarrow{a_n} q'$, $q \overset{\epsilon}{\Rightarrow} q'$ denotes $q \xrightarrow{\tau \cdots \tau} q'$ and $q \overset{a_1 \cdots a_n}{\Longrightarrow} q'$ denotes $q \overset{\epsilon}{\Rightarrow} \xrightarrow{a_1} \overset{\epsilon}{\Rightarrow} \cdots \overset{\epsilon}{\Rightarrow} \xrightarrow{a_n} \overset{\epsilon}{\Rightarrow} q'$, with $a_i \in \mathcal{L}_{\tau\delta}$. We write $q \overset{a}{\Rightarrow}$ if $\exists q'$ such that $q \overset{a}{\Rightarrow} q'$.

**Definition 2.** *A system in $IOTS(\mathcal{L}^I, \mathcal{L}^U)$: $p = \langle Q, q^0, \mathcal{L}, T \rangle$ is called input-enabled (denoted IOTS–***ie***) if all inputs are enabled in all states, i.e.*

$$\forall q \in Q, a \in \mathcal{L}^I : q \overset{a}{\Rightarrow}$$

*For $L \subseteq \mathcal{L}_p^I$ and $\forall q \in Q, a \in L : q \overset{a}{\Rightarrow}$ we say that $p$ is input-enabled with respect to $L$, denoted* $p-\mathbf{ie}(L)$.

The testing scenario on which **ioco** is based assumes that two things are given: 1) an IOTS constituting a specification of required behavior and 2) an implementation under test (SUT) that can be modeled as an input-enabled IOTS. This assumption is referred to as the test hypothesis [IJW96]. We would like to point out that we do not need to have the SUT model. We only assume that the implementation behaves as an IOTS.

Before presenting the **ioco** relation, we establish some basic properties. As in typical process algebra semantics, we sometimes use a transition system and its initial state interchangeably.

**Definition 3.** *Let $p \in IOTS$, let $P \subseteq Q_p$ be a set of states in $p$, let $q \in IOTS$ input-enabled and let $\sigma \in \mathcal{L}_\delta^*$, then*

- $p$ **after** $\sigma = \{p' \mid p \overset{\sigma}{\Rightarrow} p'\}$
- $\mathbf{out}(p) = \{a \in \mathcal{L}^U \mid p \xrightarrow{a}\} \cup \{\delta \mid p \xrightarrow{\delta}\}$
- $\mathbf{out}(P) = \bigcup\{\mathbf{out}(p) \mid p \in P\}$
- $Straces(p) = \{\sigma \in \mathcal{L}_\delta^* \mid p \overset{\sigma}{\Rightarrow}\}$

Given a specification $p$ and an (assumed) model of the SUT $q$, the relation $q$ **ioco** $p$ expresses that $q$ conforms to $p$. Whether this holds is decided on the basis of the *suspension traces* of $p$ (denoted Straces($p$)). After any suspension trace $\sigma$ from the specification, every output action (and also quiescence) that $q$ is capable of should be allowed according to the specification $p$. This is formalized by defining: $p$ **after** $q$, the set of states that can be reached in $p$ after the suspension trace $\sigma$. The set $\mathbf{out}(p)$ denotes the set of output and $\delta$-actions of $p$. And Straces($p$) denotes the suspension traces of $p$.

**Definition 4.** *Given an implementation $q \in IOTS$ input-enabled and a specification $p \in IOTS$*

$$q \textbf{ ioco } p \Leftrightarrow \forall \sigma \in Straces(p) : \textbf{out}(q \textbf{ after } \sigma) \subseteq \textbf{out}(p \textbf{ after } \sigma)$$

As expected, in the case that the implementation's actions are a subset of the specification's actions, the **ioco** relation restricted to the specification's actions follows easily. This is formalized in the following.

**Definition 5.** *Let $p$ be a $IOTS(\mathcal{L}_p^I, \mathcal{L}_p^U) = \langle Q_p, q_p^0, \mathcal{L}_p, T_p \rangle$ with $\mathcal{L}_p = \mathcal{L}_p^I \cup \mathcal{L}_p^U$, $I \subseteq \mathcal{L}_p^I$ and $U \subseteq \mathcal{L}_p^U$ we define the restriction of $p$ in $(I, U)$, denoted by $\textbf{R-}p(I, U)$ as the IOTS defined by*

- $Q = Q_p$
- $q^0 = q_p^0$
- $\mathcal{L} = I \cup U$
- $T = T_p \cap (Q \times (I \cup U \cup \{\tau\}) \times Q)$

**Theorem 1.** *Let $q$ be a $IOTS(\mathcal{L}_q^I, \mathcal{L}_q^U)$ and $p$ be a $IOTS(\mathcal{L}_q^I, \mathcal{L}_q^U)$ with $\mathcal{L}_q^I \subseteq \mathcal{L}_p^I$ and $\mathcal{L}_q^U \subseteq \mathcal{L}_p^U$. Let $q'$ be the restriction of $q$ in $IOTS(\mathcal{L}_p^I, \mathcal{L}_p^U)$ ($q' = \textbf{R-}q(\mathcal{L}_p^I, \mathcal{L}_p^U)$) then*

$$\textit{if } q' \textbf{ ioco } p \textit{ then } q \textbf{ ioco } p$$

*Proof.* Since $q'$ is **ioco** with respect to $p$ then $q'$ is $\textbf{ie}(\mathcal{L}_p^I)$ and $\mathcal{L}_{q'}^U \subseteq \mathcal{L}_p^U$ then $\forall\, \sigma \in$ *Strace* $(p)$ then $\sigma \in$ *Strace* $(q')$ then $\sigma \in$ *Strace* $(q)$. Let see that for all $a \in \textbf{out}(q \textbf{ after } \sigma)$ then $a \in \textbf{out}(p \textbf{ after } \sigma)$.

If $a \in \textbf{out}(q \textbf{ after } \sigma)$ then, because $\sigma \in$ *Strace* $(p)$, $a \in \textbf{out}(q' \textbf{ after } \sigma)$. Now using that $q' \textbf{ ioco } p$, it follows $a \in \textbf{out}(p \textbf{ after } \sigma)$.

## 3 Composition in input-output transition systems

The integration of components can be modeled algebraically by putting the components in parallel while synchronizing their common actions. The synchronization of the processes $p_1$ and $p_2$ is denoted by $p_1||p_2$.

**Definition 6.** *For $i = 1, 2$ let $p_i = \langle Q, q^0, \mathcal{L}, T \rangle$ be IOTS. If $\mathcal{L}_1^I \cap \mathcal{L}_2^I = \mathcal{L}_1^U \cap \mathcal{L}_2^U = \emptyset$ then $p_1||p_2 = \langle Q, q_1^0||q_2^0, \mathcal{L}, T \rangle$, where*

- $Q = \{q_1||q_2 \mid q_1 \in Q_1, q_2 \in Q_2\}$
- $\mathcal{L}^I = (\mathcal{L}_1^I \setminus \mathcal{L}_2^U) \cup (\mathcal{L}_2^I \setminus \mathcal{L}_1^U)$
- $\mathcal{L}^U = \mathcal{L}_1^U \cup \mathcal{L}_2^U$
- *$T$ is the minimal set satisfying the following inference rules ($a \in \mathcal{L}_\tau$):*

$$\begin{array}{lcl}
q_1 \xrightarrow{a} q_1', a \notin \mathcal{L}_2 & \vdash & q_1||q_2 \xrightarrow{a} q_1'||q_2 \\
q_2 \xrightarrow{a} q_2', a \notin \mathcal{L}_1 & \vdash & q_1||q_2 \xrightarrow{a} q_1||q_2' \\
q_1 \xrightarrow{a!} q_1', q_2 \xrightarrow{a?} q_2', a \notin \tau & \vdash & q_1||q_2 \xrightarrow{a!} q_1'||q_2' \\
q_1 \xrightarrow{a?} q_1', q_2 \xrightarrow{a!} q_2', a \notin \tau & \vdash & q_1||q_2 \xrightarrow{a!} q_1'||q_2'
\end{array}$$

Here, inputs $a?$ in one system are matched with outputs $a!$ in the other system, the result being an output $a!$ in the parallel composition of the two systems.

Given two systems $p_1$ and $p_2$, let $Share(p_1, p_2)$ denote $(\mathcal{L}_1^I \cap \mathcal{L}_2^U) \cup (\mathcal{L}_2^I \cap \mathcal{L}_1^U)$. Moreover let $Share^I(p_1, p_2) = (\mathcal{L}_1^I \cap \mathcal{L}_2^U)$ and $Share^U(p_1, p_2) = (\mathcal{L}_2^I \cap \mathcal{L}_1^U)$.

Note that Definition 6 gives only constraints on the input and output sets. Moreover, the parallel composition may give rise to an IOTS that is not strongly responsive, even if the components are. We therefore implicitly restrict ourselves to cases where the parallel composition is strongly responsive and to binary parallel composition only.

The following definition and lemma are necessary to prove our principal Theorem 2. We first introduce some notation for the projection of a trace on a label set.

**Definition 7.** *Let $a \in \mathcal{L}_\delta$ and $S \subseteq \mathcal{L}_\delta$, then*

$$\epsilon \lceil S = \epsilon$$
$$(a \cdot \sigma) \lceil S = \begin{cases} \sigma \lceil S & a \notin S \\ a \cdot (\sigma \lceil S) & a \in S \end{cases}$$

Let a system $p$ be the parallel composition of systems: $p_1$ and $p_2$, then $p = p_1 || p_2$. Under some restriction, for all reachable states $r$ in $p$ it is possible to find an state $r_1$ in $p_1$ and an state $r_2$ in $p_2$ such that $r = r_1 || r_2$. Moreover, this result holds in the other way around. The next Lemma 1 asserts this result, its proof can be found in [vdBRT03].

**Lemma 1.** *Let $p_1 \in IOTS(\mathcal{L}_1^I, \mathcal{L}_1^U)$ and $p_2 \in IOTS(\mathcal{L}_2^I, \mathcal{L}_2^U)$ with $\mathcal{L}_{p_1}^I \cap \mathcal{L}_{p_2}^I = \mathcal{L}_{p_1}^U \cap \mathcal{L}_{p_2}^U = \emptyset$ and $\mathcal{L} = \mathcal{L}^I \cup \mathcal{L}^U$ where $\mathcal{L}^I = \mathcal{L}_1^I \cup \mathcal{L}_2^I / (Share(p_1, p_2)) \wedge \mathcal{L}^U = \mathcal{L}_1^U \cup \mathcal{L}_2^U$, $r \in Q_{p_1||p_2}, \sigma \in \mathcal{L}_\delta^*$, then*

$$p_1 || p_2 \stackrel{\sigma}{\Rightarrow} r \Leftrightarrow \exists p_1', p_2' : p_1 \stackrel{\sigma \lceil L_{p_1}^\delta}{\Longrightarrow} p_1' \wedge p_2 \stackrel{\sigma \lceil L_{p_2}^\delta}{\Longrightarrow} p_2' \wedge r = p_1' || p_2'$$

## 4 Assume-guarantee reasoning with ioco

In this section we consider the following simple assume-guarantee rule:

$$\frac{i_1 || A \textbf{ ioco } S \wedge i_2 \textbf{ ioco } A}{i_1 || i_2 \textbf{ ioco } S}$$

The rule says that if, under assumption $A$, $i_1$ conforms with $S$, and $i_2$ discharges $A$, then the parallel composition $i_1 || i_2$ conforms with $S$. We will show (Theorem 2) that under certain restrictions, this rule is sound. Therefore, we can show that $i_1 || i_2$ **ioco** $S$ by checking $i_1 || A$ **ioco** $S$ and $i_2$ **ioco** $A$ separately.

In Section 2, we said that in order to be able to apply **ioco** it is necessary to assume that the implementation is input-enabled. Therefore, to assert that: $i_1 || i_2$ **ioco** $S$, we will also assume that $i_1 || i_2$ is input-enabled with respect to $S$'s inputs. In Theorem 2 we will prove that, for an input-enabled system $A$ such that: $i_1 || A$ **ioco** $S$ and $i_2$ **ioco** $A$, then $i_1 || i_2$ is **ioco** $S$. Therefore, we do not require for the specification $S$ to be input-enabled.

**Theorem 2.** *Let $i_1 \in IOTS\text{–}\mathbf{ie}(\mathcal{L}_1^I, \mathcal{L}_1^U)$, $A, i_2 \in IOTS\text{–}\mathbf{ie}(\mathcal{L}_2^I, \mathcal{L}_2^U)$ and $S \in IOTS(\mathcal{L}_3^I, \mathcal{L}_3^U)$. With $\mathcal{L}_1^I \cap \mathcal{L}_2^I = \mathcal{L}_1^U \cap \mathcal{L}_2^U = \emptyset$ then*

$$\frac{i_1 || A \textbf{ ioco } S \wedge i_2 \textbf{ ioco } A}{i_1 || i_2 \textbf{ ioco } S}$$

*Proof.* To prove Theorem 2 we prove that for any trace $\sigma \in \text{Straces}(S)$:

$$\mathbf{out}(i_1||i_2 \textbf{ after } \sigma) \subseteq \mathbf{out}(S \textbf{ after } \sigma)$$

Note that because $A, i_2 \in$ IOTS–$\mathbf{ie}(\mathcal{L}_2^I, \mathcal{L}_2^U)$ it is also the case that $Share(i_1, i_2) = Share(i_1, A)$.

Suppose $a$ in $\mathbf{out}(i_1||i_2 \textbf{ after } \sigma)$ then we need to prove that $a \in \mathbf{out}(S \textbf{ after } \sigma)$.

1. Let $a \neq \delta$
   - $a \notin Share(i_1, i_2)$
     - Let $a \in \mathbf{out}(i_1 \textbf{ after } \sigma)$, because: $Share(i_1, i_2) = Share(i_1, A)$ then $a \in \mathbf{out}(i_1||A \textbf{ after } \sigma)$. Using $i_1||A$ **ioco** $S$ we obtain $a \in \mathbf{out}(S \textbf{ after } \sigma)$.
     - Let $a \in \mathbf{out}(i_2 \textbf{ after } \sigma)$, because: $i_2$ **ioco** $A$ then $a \in \mathbf{out}(A \textbf{ after } \sigma)$. Using Definition 6: $a \in \mathbf{out}(i_1||A \textbf{ after } \sigma)$, and because: $i_1||A$ **ioco** $S$ we obtain $a \in \mathbf{out}(S \textbf{ after } \sigma)$.
   - $a \in Share(i_1, i_2)$
     - Let $a \in \mathbf{out}(i_1 \textbf{ after } \sigma)$, because: $A - \mathbf{ie}(Share(i_1, i_2)) = A - \mathbf{ie}(Share(i_1, A))$ then $a \in \mathbf{out}(i_1||A \textbf{ after } \sigma)$. Using $i_1||A$ **ioco** $S$ we obtain $a \in \mathbf{out}(S \textbf{ after } \sigma)$.
     - Let $a \in \mathbf{out}(i_2 \textbf{ after } \sigma)$, because: $i_2$ **ioco** $A$ then $a \in \mathbf{out}(A \textbf{ after } \sigma)$. Because: $a \in \mathbf{out}(i_1||i_2 \textbf{ after } \sigma)$ then $a \in \mathbf{out}(i_1||A \textbf{ after } \sigma)$.Using $i_1||A$ **ioco** $S$ we obtain $a \in \mathbf{out}(S \textbf{ after } \sigma)$.
2. Let $a = \delta$
   - $\delta \in \mathbf{out}(i_2 \textbf{ after } \sigma)$ then, because: $i_2$ **ioco** $A$, $\delta \in \mathbf{out}(A \textbf{ after } \sigma)$
     - Let $\delta \in \mathbf{out}(i_1 \textbf{ after } \sigma)$, from Definition 6: $\delta \in \mathbf{out}(i_1||A \textbf{ after } \sigma)$. Using $i_1||A$ **ioco** $S$ we obtain $\delta \in \mathbf{out}(S \textbf{ after } \sigma)$, and since $a = \delta$, we conclude that $a \in \mathbf{out}(S \textbf{ after } \sigma)$.
     - Let $\delta \notin \mathbf{out}(i_1 \textbf{ after } \sigma)$. We will show that this case is impossible. Since $\delta \notin \mathbf{out}(i_1 \textbf{ after } \sigma)$, then there must exist a $b \in \mathbf{out}(i_1 \textbf{ after } \sigma)$:
       * Let $b \notin Share(i_1, A) = Share(i_1, i_2)$ then, from Definition 6 we obtain $b \in \mathbf{out}(i_1||i_2 \textbf{ after } \sigma)$, which contradicts the original assumption that $\delta \in \mathbf{out}(i_1||i_2 \textbf{ after } \sigma)$ .
       * Let $b \in Share(i_1, A) = Share(i_1, i_2)$ then, because $A - \mathbf{ie}(Share^U(i_1, i_2)) \wedge i_2 - \mathbf{ie}(\mathcal{L}_A^I)\}$ we obtain $b \in \mathbf{out}(i_1||i_2 \textbf{ after } \sigma)$, which again contradicts the original assumption that $\delta \in \mathbf{out}(i_1||i_2 \textbf{ after } \sigma)$.
   - $\delta \in \mathbf{out}(i_1 \textbf{ after } \sigma)$. This case is proven in a way similar to the above.
     - Let $\delta \in \mathbf{out}(i_2 \textbf{ after } \sigma)$ then, because $i_2$ **ioco** $A$, $\delta \in \mathbf{out}(A \textbf{ after } \sigma)$. From Definition 6: $\delta \in \mathbf{out}(i_1||A \textbf{ after } \sigma)$. Using $i_1||A$ **ioco** $S$ we obtain $\delta \in \mathbf{out}(S \textbf{ after } \sigma)$.
     - Let $\delta \notin \mathbf{out}(i_2 \textbf{ after } \sigma)$. We will show that this case is impossible. Since $\delta \notin \mathbf{out}(i_2 \textbf{ after } \sigma)$, there must exist a $b \in \mathbf{out}(i_2 \textbf{ after } \sigma)$:
       * Let $b \in Share(i_1, i_2)$ then, because $i_1 - \mathbf{ie}(Share(i_1, i_2))$ and using Definition 6 we obtain $b \in \mathbf{out}(i_1||i_2 \textbf{ after } \sigma)$, which is a contradiction.
       * Let $b \notin Share(i_1, i_2)$ then, from Definition 6: $b \in \mathbf{out}(i_1||i_2 \textbf{ after } \sigma)$, which again contradicts the original assumption.

## 5 Example

We illustrate assume-guarantee reasoning with **ioco** on a simple example: a coffee machine depicted in Figure 5. The upper part shows the architecture of a coffee machine, with two components (**money** and **drinks**). The lower part shows the global specification $S$.

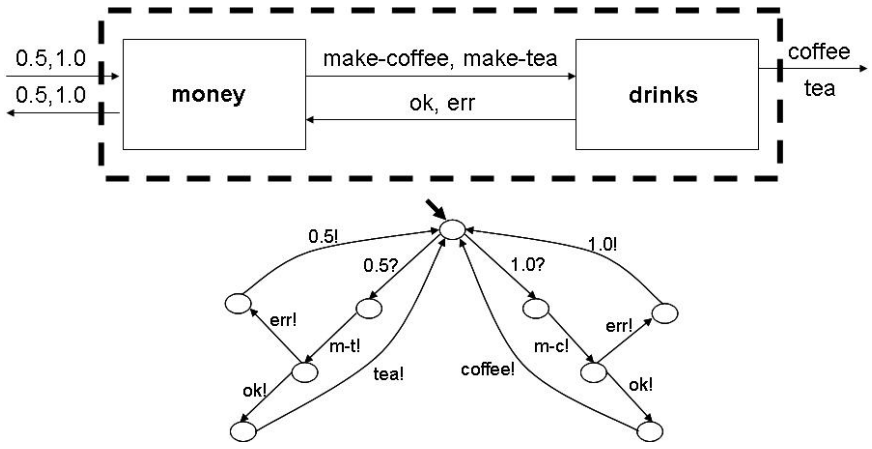


**Fig. 1.** Architecture of a coffee machine (modified version of [vdBRT03]) and its global specification

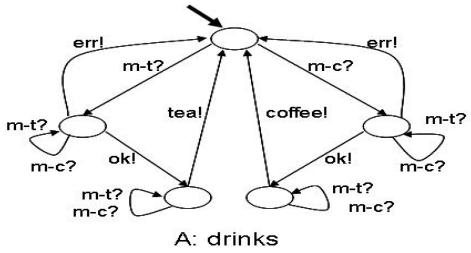


**Fig. 2.** Transition systems: assumption $A$

Figure 5 shows an assumption $A$ for the drinks component.

Then, the implementations drinks and money illustrated in Figure 5 are accepted implementations for the coffee machine specification. We used the TorX tool [TB03] to verity our results.

## 6 Conclusions and Future Work

We discussed assume-guarantee reasoning in the context of the **ioco** relation. We presented an assume-guarantee rule and we showed its soundness. In comparison with previous work on compositional testing, we do not require the specification $S$ to be given as a set of components. Moreover, we do not require for the specification $S$ to be input-enabled.

To the best of our knowledge, this paper presents the fist approach to use **ioco** in the context of assume-guarantee reasoning. This result allows one to implement and test components of a system separately, while drawing conclusions about their composition with respect to **ioco** .

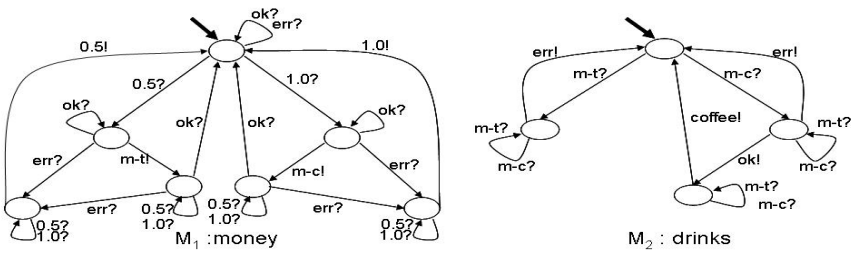


**Fig. 3.** Transition systems: implementations **money** and **drinks**

For future work, we would like to apply the presented theory on realistic applications. We also plan to investigate if our previous work for automated assumption generation will apply in the context of **ioco** .

## References


[CGP03] J. M. Cobleigh, D. Giannakopoulou, and C. S. Păsăreanu. Learning assumptions for compositional verification. In *In International Conference on Tools and Algorithms for the Construction and Analysis of Systems.TACAS*, Apr. 2003.

[CLM89] E. M. Clarke, D. E. Long, and K. L. McMillan. Compositional model checking. In *In Symposium on Logic in Computer Science*, June 1989.

[GGSV02] W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes. Generating finite state machines from abstract state machines. In *In International Symposium on Software Testing and Analysis*, July 2002.

[GPB02] D. Giannakopoulou, C. S. Păsăreanu, and H. Barringer. Assumption generation for software component verification. In *In International Conference on Automated Software Engineering*, Sept. 2002.

[IJW96] ITU-T SG 10/Q.8 ISO/IEC JTC1/SC21 WG1. Information retrieval, transfer and management for osi; framework: Formal methods in conformance testing. In *Committee Draft CD 13245-1, ITU-T proposed recommendation Z.500. ISO-ITU-T*, Geneve, 1996.

[Pnu84] A. Pnueli. In transition from global to modular temporal reasoning about programs. In *In K. Apt, editor, Logic and Models of Concurrent Systems, volume 13*, 1984.

[TB03] J. Tretmans and E. Brinksma. Torx: Automated model-based testing. In *First European Conference on Model-Driven Software Engineering, Nuremberg*. A.Hartmann and K.Dussa-Ziegler, 2003.

[Tre96] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. In *Software-Concepts and Tools, 17(3)*, pages 103–120. Also: Technical Report N0. 96-26, Center for Telematics and Information Technology, University of Twente, The Netherlands, 1996.

[vdBRT03] Machiel van der Bijl, Arend Rensink, and Jan Tretmans. Component based testing with ioco. Technical report, University of Twente, 2003.