Abstract. This paper presents a combination between the assume-guarantee paradigm and the testing relation \textit{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 \textit{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 \textit{ioco}. This enables testing \textit{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 \textit{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 \textit{ioco}. For two pairs of input-enabled systems $i_1, s_1$ in IOTS($\mathcal{L}^I_1, \mathcal{L}^U_1$) and $i_2, s_2$ in IOTS($\mathcal{L}^I_2, \mathcal{L}^U_2$) the following compositional rule is proved:
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 \(i_2 \ \text{ioco} \ A\) and \(i_1||A \ \text{ioco} \ S\) then \(i_1||i_2 \ \text{ioco} \ S\). 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. Section 4 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 \((Q,q^0,\mathcal{L},T)\), 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 $L^I$ and $L^U$ by $IOTS(L^I, L^U)$.

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 $\delta \notin L_\tau$) to represent quiescence. For an arbitrary $L \subseteq L_\tau$, 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 $L^U_\tau$-labeled paths.

A trace is a finite sequence of observable actions. The set of all traces over $L$ (with $L \subseteq L$) 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: $q \xrightarrow{a_1 \ldots a_n} q'$ denotes $q \xrightarrow{a_1} \ldots \xrightarrow{a_n} q'$, $q \Rightarrow q'$ denotes $q \xrightarrow{\tau_1 \ldots \tau_n} q'$ and $q \xrightarrow{a_1 \ldots a_n, \tau_1 \ldots \tau_n} q'$ denotes $q \Rightarrow a_1 \Rightarrow \ldots \Rightarrow a_n \Rightarrow q'$, with $a_i \in L_\tau$. We write $q \Rightarrow$ if $\exists q'$ such that $q \xrightarrow{a} q'$.

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

$$\forall q \in Q, a \in L^I : q \xrightarrow{a}$$

For $L \subseteq L^I_p$ and $\forall q \in Q, a \in L : q \xrightarrow{a}$ we say that $p$ is input-enabled with respect to $L$, denoted $p\text{-ie}(L)$.

The testing scenario on which $\text{ioco}$ 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 after the suspension trace $\sigma$(with $\sigma \subseteq L_\delta \tau$) to represent conforming to $p$. Whether this holds is decided on the basis of the suspension traces of $p$ (denoted $\text{Straces}(p)$). 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 $\text{out}(p)$ denotes the set of output and $\delta$-actions of $p$. And $\text{Straces}(p)$ denotes the suspension traces of $p$. 

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

\[ q \text{ ioco } p \iff \forall \sigma \in \text{Straces}(p) : \text{out}(q \text{ after } \sigma) \subseteq \text{out}(p \text{ after } \sigma) \]

As expected, in the case that the implementation’s actions are a subset of the specification’s actions, the \( \text{ioco} \) relation restricted to the specification’s actions follows easily. This is formalized in the following.

Definition 5. Let \( p \) be a IOTS(\( L_p^I, L_p^U \)) = \( \langle Q_p, q^0_p, L_p, T_p \rangle \) with \( L_p = L_p^I \cup L_p^U \), \( I \subseteq L_p^I \) and \( U \subseteq L_p^U \) we define the restriction of \( p \) in \( (I, U) \), denoted by \( R-p(I, U) \) as the IOTS defined by

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

Theorem 1. Let \( q \) be a IOTS(\( L_q^I, L_q^U \)) and \( p \) be a IOTS(\( L_p^I, L_p^U \)) with \( L_q^I \subseteq L_p^I \) and \( L_q^U \subseteq L_p^U \).

Let \( q' \) be the restriction of \( q \) in IOTS(\( L_p^I, L_p^U \)) (\( q' = R-q(L_p^I, L_p^U) \)) then

\[ \text{if } q' \text{ ioco } p \text{ then } q \text{ ioco } p \]

Proof. Since \( q' \) is \( \text{ioco} \) with respect to \( p \) then \( q' \) is \( \text{ie}(L_p^I) \) and \( L_q^U \subseteq L_p^U \) then \( \forall \sigma \in \text{Strace}(p) \) then \( \sigma \in \text{Strace}(q') \) then \( \sigma \in \text{Strace}(q) \). Let see that for all \( a \in \text{out}(q \text{ after } \sigma) \) then \( a \in \text{out}(p \text{ after } \sigma) \).

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

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 \( p_i = \langle Q, q^0_i, L, T \rangle \) be IOTS. If \( L_1^I \cap L_2^I = L_1^U \cap L_2^U = \emptyset \) then \( p_1|p_2 = \langle Q, q^0, L, T \rangle \), where

\- \( Q = \{q_1||q_2 : q_1 \in Q_1, q_2 \in Q_2\} \)
\- \( L^I = (L_1^I \setminus L_2^U) \cup (L_2^I \setminus L_1^U) \)
\- \( L^U = L_1^U \cup L_2^U \)
\- \( T \) is the minimal set satisfying the following inference rules (\( a \in L_\tau \)):

\[ \begin{align*}
q_1 &\xrightarrow{a} q'_1, a \notin L_2 & \vdash q_1||q_2 &\xrightarrow{a} q'_1||q_2 \\
q_2 &\xrightarrow{a} q'_2, a \notin L_1 & \vdash q_1||q_2 &\xrightarrow{a} q_1||q'_2 \\
q_1 &\xrightarrow{a} q_1, q_2 &\xrightarrow{a} q_2, a \notin \tau & \vdash q_1||q_2 &\xrightarrow{a} q'_1||q'_2 \\
q_1 &\xrightarrow{a} q_1, q_2 &\xrightarrow{a} q_2, a \notin \tau & \vdash q_1||q_2 &\xrightarrow{a} q'_1||q'_2
\end{align*} \]

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 \( \text{Share}(p_1, p_2) = (L_1^I \cap L_2^U) \cup (L_2^I \cap L_1^U) \). Moreover let \( \text{Share}^I(p_1, p_2) = (L_1^I \cap L_2^I) \) and \( \text{Share}^U(p_1, p_2) = (L_1^U \cap L_2^U) \).
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
\[
\epsilon[S] = \epsilon
\]
\[
(a \cdot \sigma)[S] = \begin{cases} 
\sigma[S] & a \notin S \\
\ a \cdot (\sigma[S]) & a \in S 
\end{cases}
\]

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 \( p_1 \in \text{IOTS}(\mathcal{L}_1^I, \mathcal{L}_1^U) \) and \( p_2 \in \text{IOTS}(\mathcal{L}_2^I, \mathcal{L}_2^U) \) with \( \mathcal{L}_1^I \cap \mathcal{L}_2^I = \emptyset \) and \( \mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2 \) where \( \mathcal{L}^I = \mathcal{L}_1^I \cup \mathcal{L}_2^I / (\text{Share}(p_1, p_2)) \) and \( \mathcal{L}^U = \mathcal{L}_1^U \cup \mathcal{L}_2^U \), \( r \in \mathcal{Q}_{p_1||p_2} \), \( \sigma \in \mathcal{L}^\delta_\delta \), then
\[
p_1||p_2 \Rightarrow r \Leftrightarrow \exists p_1', p_2' : p_1 \Rightarrow p_1' \land p_2 \Rightarrow p_2' \land r = p_1'||p_2'
\]

### 4 Assume-guarantee reasoning with \textsf{ioco}

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

\[
i_1||A \textsf{ioco} S \land i_2 \textsf{ioco} A
\]

\[
i_1||i_2 \textsf{ioco} S
\]

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 \( i_1||i_2 \textsf{ioco} S \) by checking \( i_1||A \textsf{ioco} S \) and \( i_2 \textsf{ioco} A \) separately.

In Section 2, we said that in order to be able to apply \textsf{ioco} it is necessary to assume that the implementation is input-enabled. Therefore, to assert that: \( i_1||i_2 \textsf{ioco} S \), 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: \( i_1||A \textsf{ioco} S \) and \( i_2 \textsf{ioco} A \), then \( i_1||i_2 \) is \textsf{ioco} \( S \). Therefore, we do not require for the specification \( S \) to be input-enabled.

**Theorem 2.** Let \( i_1 \in \text{IOTS-\textsf{ie}}(\mathcal{L}_1^I, \mathcal{L}_1^U) \), \( A, i_2 \in \text{IOTS-\textsf{ie}}(\mathcal{L}_2^I, \mathcal{L}_2^U) \) and \( S \in \text{IOTS}(\mathcal{L}_3^I, \mathcal{L}_3^U) \). With \( \mathcal{L}_1^I \cap \mathcal{L}_2^I = \emptyset \) then
\[
i_1||A \textsf{ioco} S \land i_2 \textsf{ioco} A
\]

\[
i_1||i_2 \textsf{ioco} S
\]

**Proof.** To prove Theorem 2 we prove that for any trace \( \sigma \in \text{Straces}(S) \):
out(i_1||i_2 after σ) ⊆ out(S after σ)

Note that because A, i_2 ∈ IOTS – ie(L^I_2, L^U_2) it is also the case that Share(i_1, i_2) = Share(i_1, A).

Suppose a in out(i_1||i_2 after σ) then we need to prove that a ∈ out(S after σ).

1. Let a ≠ δ
   - a ∉ Share(i_1, i_2)
     - Let a ∈ out(i_1 after σ), because: Share(i_1, i_2) = Share(i_1, A) then a ∈ out(i_1||A after σ). Using i_1||A ioco S we obtain a ∈ out(S after σ).
     - Let a ∈ out(i_2 after σ), because: i_2 ioco A then a ∈ out(A after σ). Using Definition 6: a ∈ out(i_1||A after σ), and because: i_1||A ioco S we obtain a ∈ out(S after σ).
   - a ∈ Share(i_1, i_2)
     - Let a ∈ out(i_1 after σ), because: A – ie(Share(i_1, i_2)) = A – ie(Share(i_1, A)) then a ∈ out(i_1||A after σ). Using i_1||A ioco S we obtain a ∈ out(S after σ).
     - Let a ∈ out(i_2 after σ), because: i_2 ioco A then a ∈ out(A after σ). Because: a ∈ out(i_1||i_2 after σ) then a ∈ out(i_1||A after σ). Using i_1||A ioco S we obtain a ∈ out(S after σ).

2. Let a = δ
   - δ ∈ out(i_2 after σ) then, because: i_2 ioco A, δ ∈ out(A after σ)
     - Let δ ∈ out(i_1 after σ), from Definition 6: δ ∈ out(i_1||A after σ). Using i_1||A ioco S we obtain δ ∈ out(S after σ), and since a = δ, we conclude that a ∈ out(S after σ).
     - Let δ ∉ out(i_1 after σ). We will show that this case is impossible. Since δ ∉ out(i_1 after σ), then there must exist a b ∈ out(i_1 after σ):
       * Let b ∉ Share(i_1, A) = Share(i_1, i_2) then, from Definition 6 we obtain b ∈ out(i_1||i_2 after σ), which contradicts the original assumption that δ ∈ out(i_1||i_2 after σ).
       * Let b ∈ Share(i_1, A) = Share(i_1, i_2) then, because A – ie(Share(L^U_2(i_2), i_2)) ∧ i_2 – ie(L^I_2) we obtain b ∈ out(i_1||i_2 after σ), which again contradicts the original assumption that δ ∈ out(i_1||i_2 after σ).
   - δ ∈ out(i_1 after σ). This case is proven in a way similar to the above.
     - Let δ ∈ out(i_2 after σ) then, because i_2 ioco A, δ ∈ out(A after σ). From Definition 6: δ ∈ out(i_1||A after σ). Using i_1||A ioco S we obtain δ ∈ out(S after σ).
     - Let δ ∉ out(i_2 after σ). We will show that this case is impossible. Since δ ∉ out(i_2 after σ), there must exist a b ∈ out(i_2 after σ):
       * Let b ∈ Share(i_1, i_2) then, because i_1 – ie(Share(i_1, i_2)) and using Definition 6 we obtain b ∈ out(i_1||i_2 after σ), which is a contradiction.
       * Let b ∉ Share(i_1, i_2) then, from Definition 6: b ∈ out(i_1||i_2 after σ), 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.
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 verify our results.

6 Conclusions and Future Work

We discussed assume-guarantee reasoning in the context of the \textit{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 first approach to use \textit{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 \textit{ioco}.
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 \textit{ioco}.

References

\begin{itemize}
\end{itemize}