Synchronizing Asynchronous Conformance Testing

Neda Noroozi^{1,2}, Ramtin Khosravi^{3},
Mohammad Reza Mousavi^{1}, and Tim A.C. Willemse^{1}

^{1} Eindhoven University of Technology, Eindhoven, The Netherlands
^{2} Fanap Corporation (IT Subsidiary of Pasargad Bank), Tehran, Iran
^{3} University of Tehran, Tehran, Iran

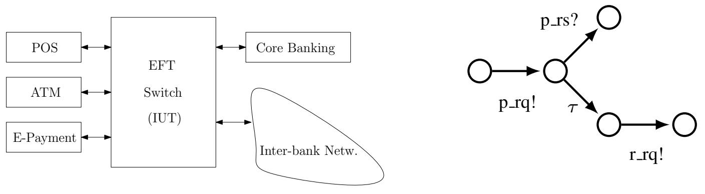
Abstract. We present several theorems and their proofs which enable using synchronous testing techniques such as input output conformance testing (**ioco**) in order to test implementations only accessible through asynchronous communication channels. These theorems define when the synchronous test-cases are sufficient for checking all aspects of conformance that are observable by asynchronous interaction with the implementation under test.

1 Introduction

Due to the ubiquitous presence of distributed systems (ranging from distributed embedded systems to the Internet), it becomes increasingly important to establish rigorous model-based testing techniques with an asynchronous model of communication in mind. This fact has been noted by the pioneering pieces work in the area of formal conformance testing, e.g., see [7, Chapter 5], [10] and [11], and has been addressed extensively by several researchers in this field ever since [2, 4–6, 12, 13].

We stumbled upon this problem in our attempt to apply input-output conformance testing (**ioco**) [8, 9] to an industrial embedded system from the banking domain [1]. A schematic view of the implementation under test (IUT) and its environment is given in Figure 1.(a). The IUT is an Electronic Funds Transfer (EFT) switch, which provides a communication mechanism among different components of a card-based financial system. On one side of the IUT, there are components that the end-user deals with, such as Automated Teller Machines (ATMs), Point-of-Sale (POS) devices and e-Payment applications. On the other side, there are Core-Banking systems and the inter-bank network connecting EFT switches of different financial institutions.

To test the EFT switch, an automated on-line test-case generator is connected to it; the tester communicates (using an adapter) via a network with the IUT. This communication is inherently asynchronous and hence subtleties concerning asynchronous testing arise naturally in our context. A simplified specification of the switch in which these subtleties appear is depicted in Figure 1.(b). In this figure, the EFT switch sends a purchase request to the core banking system and either receives a response or after an internal step (e.g., an internal time-out, denoted by τ) sends a reversal request to the POS. In the synchronous setting, after sending a purchase request and receiving a response, observing a reversal request will lead to the fail verdict. This is justified by the fact that receiving a response should force the system to take the left-hand-side



(a) (b)

Fig. 1. EFT Switch and a simplified specification

transition at the moment of choice in the depicted specification. However, in the asynchronous setting, a response is put on a channel and is yet to be communicated to the IUT. It is unclear to the remote observer when the response is actually consumed by the IUT. Hence, even when a response is sent to the system the observer should still expect to receive a reversal request.

The problems encountered in our practical case study have been encountered by other researchers. It is well-known that not all systems are amenable to asynchronous testing since they may feature phenomena (e.g., a choice between accepting input and generating output) that cannot be reliably observed in the asynchronous setting (e.g., due to unknown delays). In other words, to make sure that test-cases generated from the specification can test the IUT by asynchronous interactions and reach verdicts that are meaningful for the original IUT, either the class of IUTs, or the class of specifications, or the test-case generation algorithm (or a combination thereof) has to be adapted.

Related work. In [12, Chapter 8] and [13], both the class of IUTs has been restricted (to the so-called *internal choice* specifications) and further the test-case generation algorithm is adapted to generate a restricted set of test-cases. Then, it is argued (with a proof sketch) that in this setting, the verdict obtained through asynchronous interaction with the system coincides with the verdict (using the same set of restricted test-cases) in the synchronous setting. We give a full proof of this result in Section 3 and report a slight adjustment to it, without which a counter-example is shown to violate the property. It remains to be investigated what notion of conformance testing is induced by the class of test-cases proposed in [12, 13].

In [6] a method is presented for generating test-cases from the synchronous specification that are sound for the asynchronous implementation. The main idea is to saturate a test-case with observation delays caused by asynchronous interactions. In this paper, we adopt a restriction imposed on the implementation inspired by [6, Theorem 1] and prove that in the setting of **ioco** testing this is sufficient for using synchronous test-case for the asynchronous implementation (dating back to [7]).

In [4, 5] the asynchronous test framework is extended to the setting where separate test-processes can observe input and output events and relative distinguishing power of these settings are compared. Although this framework may be natural in practice, we avoid following the framework of [4, 5] since our ultimate goal is to compare asynchronous testing with the standard **ioco** framework and the framework of [4, 5] is notationally very different. For the same reason, we do not consider the approach of [2], which uses a stamping mechanism attached to the IUT, thus observing the actual order input and output before being distorted by the queues.

To summarize, the present paper re-visits the much studied issue of asynchronous testing and formulates and proves some theorems that show when it is (im)possible to synchronize asynchronous testing, i.e., interaction with an IUT through asynchronous channels and still obtain verdicts that coincide with that of testing the IUT using the synchronous interaction mechanisms.

Structure of the paper After presenting some preliminaries in Section 2, we give a full proof of the main result of [12, Chapter 8] and [13] (with a slight modification) in Section 3. Then, in Section 4, we re-formulate the same results in the pure **ioco** setting. Finally, in Section 5, we show that the restrictions imposed on the implementation in Section 4 are not only sufficient to obtain the results but also necessary and hence characterize the implementations for which asynchronous testing can be reduced to synchronous testing. The paper is concluded in Section 6.

2 Preliminaries

In this section, we review some common formal definitions from the literature of labeled transition systems and **ioco** testing [9].

Specifications, actions and traces. In our model-based testing approach, systems are typically formalized using variations of a labeled transition system (LTS). Let τ be a constant representing an unobservable action.

Definition 1 (LTS). *A labeled transition system (LTS) is a 4-tuple $\langle S, L, \rightarrow, s_0\rangle$, where S is a set of states, L is a finite alphabet that does not contain τ, $\rightarrow \subseteq S \times (L \cup \{\tau\}) \times S$ is the transition relation, and $s_0 \in S$ is the initial state.*

Fix an arbitrary LTS $\langle S, L, \rightarrow, s_0\rangle$. We shall often refer to the LTS by referring to its initial state s_0. Let $s, s' \in S$ and $x \in L \cup \{\tau\}$. We write $s \xrightarrow{x} s'$ rather than $(s, x, s') \in \rightarrow$; moreover, we write $s \xrightarrow{x}$ when $s \xrightarrow{x} s'$ for some s', and $s \not\xrightarrow{x}$ when not $s \xrightarrow{x}$. The transition relation is generalized to (weak) traces by the following deduction rules:

$$\frac{}{s \overset{\epsilon}{\Longrightarrow} s} \qquad \frac{s \overset{\sigma}{\Longrightarrow} s'' \quad s'' \xrightarrow{x} s' \quad x \neq \tau}{s \overset{\sigma x}{\Longrightarrow} s'} \qquad \frac{s \overset{\sigma}{\Longrightarrow} s'' \quad s'' \xrightarrow{\tau} s'}{s \overset{\sigma}{\Longrightarrow} s'}$$

In line with our notation for transitions, we write $s \overset{\sigma}{\Longrightarrow}$ if there is a s' such that $s \overset{\sigma}{\Longrightarrow} s'$, and $s \overset{\sigma}{\not\Longrightarrow}$ when no s' exists such that $s \overset{\sigma}{\Longrightarrow} s'$.

Definition 2 (Traces and Enabled Actions). *Let $s \in S$ and $S' \subseteq S$. We define:*

1. $\mathsf{traces}(s) =_{def} \{\sigma \in L^* \mid s \overset{\sigma}{\Longrightarrow}\}$*, and we define* $\mathsf{traces}(S') =_{def} \bigcup_{s \in S'} \mathsf{traces}(s)$
2. $\mathbf{init}(s) =_{def} \{a \in L \cup \{\tau\} \mid s \xrightarrow{a}\}$*, and we define* $\mathbf{init}(S') =_{def} \bigcup_{s \in S'} \mathbf{init}(s)$*,*
3. $\mathbf{Sinit}(s) =_{def} \{a \in L \mid s \overset{a}{\Longrightarrow}\}$*, and we define* $\mathbf{Sinit}(S') =_{def} \bigcup_{s \in S'} \mathbf{Sinit}(s)$*.*

A state in an LTS is said to *diverge* if it is the source of an infinite sequence of τ-labeled transitions. An LTS is *divergent* if one of its reachable states diverges.

Inputs, Outputs and Quiescence. In LTSs labels are treated uniformly. When engaging in an interaction with an actual implementation, the initiative to communicate is often not fully symmetric: the implementation is stimulated and observed. We therefore refine the LTS model to incorporate this distinction.

Definition 3 (IOLTS). *An input-output labeled transition system (IOLTS) is an LTS $\langle S, L, \rightarrow, s_0 \rangle$, where the alphabet L is partitioned into a set L_I of inputs and a set L_U of outputs.*

Throughout this paper, whenever we are dealing with an IOLTS (or one of its refinements), we tacitly assume that the given alphabet L for the IOLTS is partitioned in sets L_I and L_U. In our examples we distinguish inputs from outputs by annotating them with question- (?) and exclamation-mark (!), respectively. Note that these annotations are *not* part of action names.

Quiescence, defined below, is an essential ingredient in the more advanced conformance testing theories. In its traditional phrasing, it characterizes system states that do not produce outputs and which are stable, i.e., those that cannot evolve to another state by performing a silent action.

Definition 4 (Quiescence). *State $s \in S$ is called* quiescent*, denoted by $\delta(s)$, iff $\mathbf{init}(s) \subseteq L_I$. We say s is* weakly quiescent*, denoted by $\delta_q(s)$, iff $\mathbf{Sinit}(s) \subseteq L_I$.*

The notion of weak quiescence is appropriate in the asynchronous setting, where the lags in the communication media interfere with the observation of quiescence: an observer cannot tell whether a system is engaged in some internal transitions or has come to a standstill. By the same token, in an asynchronous setting it becomes impossible to distinguish divergence from quiescence; we re-visit this issue in our proofs of synchronizing asynchronous conformance testing.

Testing hypotheses. Several formal testing theories build on the assumption that the implementations can be modeled by a particular IOLTS; this assumption is part of the so-called *testing hypothesis* underlying the testing theory. Not all theories rely on the same assumptions. We introduce two models, viz., the *input output transition systems*, used in Tretmans' testing theory [9] and the *internal choice input output transition systems*, introduced by Weiglhofer and Wotawa [12, 13].

Tretmans' input-output transition systems are basically plain IOLTSs with the additional assumption that inputs can always be accepted.

Definition 5 (IOTS). *A state $s \in S$ in an IOLTS $M = \langle S, L, \rightarrow, s_0 \rangle$ is* input-enabled *iff $L_I \subseteq \mathbf{Sinit}(s)$. The IOLTS M is an* input output transition system *(IOTS) iff every state $s \in S$ is input-enabled.*

From hereon, we denote the class of input output transition systems ranging over L_I and L_U by $\mathsf{IOTS}(L_I, L_U)$. Weiglhofer and Wotawa's internal choice input output transition systems relax Tretmans' input-enabledness requirement; at the same time, however, they impose an additional restriction on the presence of inputs.

Definition 6 (Internal choice IOTS). *An IOLTS $\langle S, L, \rightarrow, s_0 \rangle$ is an* internal choice input output transition system *(IOTS$^{\sqcap}$) if:*

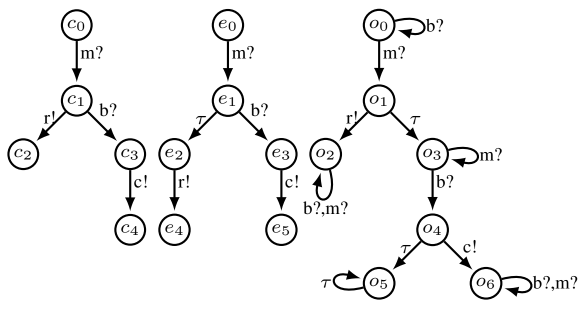


Fig. 2. IOLTS with different moments of choice (*m:money, r:refund, b:button, c:coffee)*

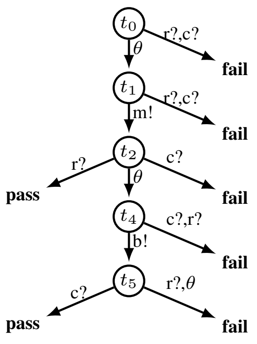
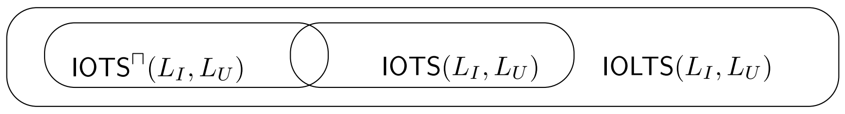


Fig. 3. A test case

1. *quiescent states are input-enabled, i.e., for all* $s \in S$, *if* $\delta(s)$, *then* $L_I \subseteq \mathbf{Sinit}(s)$
2. *only quiescent states may accept inputs, i.e., for all* $s \in S$, *if* $\mathbf{init}(s) \cap L_I \neq \emptyset$ *then* $\delta(s)$.

We denote the class of IOTS$^\sqcap$ models ranging over L_I and L_U by $\mathsf{IOTS}^\sqcap(L_I, L_U)$. The following Venn-diagram visualizes the relation between the two different testing hypotheses.


$\mathsf{IOTS}^\sqcap(L_I, L_U)$
$\mathsf{IOTS}(L_I, L_U)$
$\mathsf{IOLTS}(L_I, L_U)$

We note that the intersection between $\mathsf{IOTS}^\sqcap(L_I, L_U)$ and $\mathsf{IOTS}(L_I, L_U)$ is in a sense only fulfilled by the most superficial models, viz., those IOLTSs that never provide proper outputs. If requirement 2 is dropped from Definition 6, then clearly $\mathsf{IOTS}^\sqcap(L_I, L_U)$ subsumes $\mathsf{IOTS}(L_I, L_U)$.

Example 1. The two labeled transition systems c_0 and e_0 in Figure 2 model a coffee machine which after receiving money, either refunds or accepts it, lets the coffee button be pressed and delivers coffee consequently. IOLTS o_0 in Figure 2 models a disordered coffee machine which after pressing coffee button may or may not deliver coffee. In IOLTS c_0, after doing the first transition, inserting money, there is a choice between input and output. Although IOLTS e_0 does not feature an immediate race between input and output actions, the possibility of output $r!$ can be ruled out by providing input $b?$. In the IOLTS o_0, however, there is a moment of time after which no output can be observed, i.e., after taking the unobservable transition the system reaches the quiescent state and the input $b?$ is accepted by the system.

IOLTSs c_0 and e_0 are not internal choice IOTSs while o_0 is. None the aforementioned IOLTSs are IOTSs; they can be made IOTSs by adding self-loops for all absent input transitions at each and every state.

Testing. We next define the notion of a test case. We assume that it can, in the most general case, be described by a tree-shaped IOLTS. Such a test case prescribes when an input should be fed to the implementation-under-test and when its possible outputs should be observed. In a test case, the *observation* of quiescence is modeled using a special θ symbol.

Definition 7 (Test case). *A test case is an IOLTS $\langle S, L, \rightarrow, s_0 \rangle$, where S is a finite set of states reachable from $s_0 \in S$, the terminal states* **pass** *and* **fail** *are part of S, and we have $\theta \in L_I$. In addition, the transition relation $\rightarrow$ is acyclic and deterministic such that:*

1. **pass** *and* **fail** *states appear only as targets of transitions labeled by an element of L_I, and*
2. *for all $s \in S \setminus \{\mathbf{pass}, \mathbf{fail}\}$, we require that $\mathbf{init}(s) = (L_I \setminus \{\theta\}) \cup \{x\}$ for some $x \in L_U \cup \{\theta\}$.*

We denote the class of test cases ranging over inputs L_I and outputs L_U by $\mathsf{TTS}(L_U, L_I)$.

Notice that the observation θ is an *input* to a test case; this is in line with the view that outputs produced by an implementation are inputs to a test case. Moreover, we note that a test case has no transitions labeled with the silent action τ.

We formalize the way a test case communicates with an actual implementation, modeled by an IOLTS.

Definition 8 (Synchronous execution). *Let $M = \langle S, L, \rightarrow, s_0 \rangle$ be an IOLTS, and let $\langle T, L', \rightarrow, t_0 \rangle$ be a test case, such that $L_I = L'_U$ and $L_U = L'_I \setminus \{\theta\}$. Let $s, s' \in S$ and $t, t' \in T$. Then the synchronous execution of the test case and M is defined through the following inference rules:*

$$\frac{s \xrightarrow{\tau} s'}{t \rceil\!\lceil s \xrightarrow{\tau} t \rceil\!\lceil s'}\ (R1) \qquad \frac{t \xrightarrow{x} t' \quad s \xrightarrow{x} s'}{t \rceil\!\lceil s \xrightarrow{x} t' \rceil\!\lceil s'}\ (R2) \qquad \frac{t \xrightarrow{\theta} t' \quad \delta(s)}{t \rceil\!\lceil s \xrightarrow{\theta} t' \rceil\!\lceil s}\ (R3)$$

Finally, we state what it means for an implementation to *pass* a test case.

Definition 9 (Verdict). *Let implementation M be given by IOLTS $\langle S, L, \rightarrow, s_0 \rangle$, and let $\langle T, L \cup \{\theta\}, \rightarrow, t_0 \rangle$ be a test case. We say that state $s \in S$* passes *the test case, denoted s **passes** t_0 iff there is no $\sigma \in (L \cup \{\theta\})^*$ and no state $s' \in S$, such that $t_0 \rceil\!\lceil s \overset{\sigma}{\Longrightarrow} \mathbf{fail} \rceil\!\lceil s'$.*

3 Adapting IOCO to Asynchronous Setting

In order to perform conformance testing in the asynchronous setting in [12] and [13] both the class of implementations and test cases are restricted. Then, it is argued (with a proof sketch) that in this setting, the verdict obtained through asynchronous interaction with the system coincides with the verdict (using the same set of restricted test-cases) in the synchronous setting. In this section, we re-visit the approach of [12] and [13], give full proof of their main result and point out a slight imprecision in it.

3.1 Test Cases for Internal Choice Implementations

Asynchronous communication delays obscure the observation of the tester; for example, the tester cannot precisely establish when the input sent to the system is actually consumed by it.

Internal choice test-cases, formally defined below, only allow for providing an input if quiescence has been observed beforehand.

Definition 10 (Internal choice test case). *A test case $\langle S, L, \rightarrow, s_0 \rangle$ is an* internal choice test case *(abbreviated to TTS$^{\sqcap}$) if for all $s \in S$, all $x \in L_U$ and all $\sigma \in L^*$, if $\sigma x \in \mathsf{traces}(s)$ then $\sigma = \sigma' \cdot \theta$.*

We denote the class of internal choice test cases ranging over inputs L_I and outputs L_U by $\mathsf{TTS}^{\sqcap}(L_U, L_I)$.

Example 2. Figure 3 shows an internal choice test case for o_0 in Figure 2. In this test case, inputs for the implementation are enabled only in states reached by a θ-transition.

The property given below illustrates that, indeed, the interaction between an internal choice test case and an IOLTS proceed in an orchestrated fashion: the IOLTS is only provided stimuli whenever it has reached a stable situation.

Property 1. Let $\langle S, L, \rightarrow, s_0 \rangle$ be an arbitrary IOLTS and $\langle T, L', \rightarrow, t_0 \rangle$ be an internal choice test case. Let $x \in L'_U \setminus \{\theta\}$ $(= L_I)$, $\sigma \in L'^*$, $s, s' \in S$ and $t, t' \in T$. We have the following property:

$$t \rceil\!| s \xRightarrow{\sigma \cdot x} t' \rceil\!| s' \text{ implies } \exists \sigma' \in L'^* : \sigma = \sigma' \cdot \theta$$

3.2 Asynchronous Communication

Asynchronous communication, as described in [7, Chapter 5], can be simulated by modelling the communications with the implementation through two dedicated FIFO channels. One is used for sending the inputs to the implementation, whereas the other is used to queue the outputs produced by the implementation. We assume that the channels are unbounded. By adding channels to an implementation, its visible behavior changes. This is formalized below.

Definition 11 (Queue operator). *Let $\langle S, L, \rightarrow, s_0 \rangle$ be an arbitrary IOLTS, $\sigma_i \in L_I^*$, $\sigma_u \in L_U^*$ and $s, s' \in S$. The unary queue operator $[\sigma_u \ll \text{-} \ll \sigma_i]$ is then defined by the following axioms and inference rules:*

$$[\sigma_u \ll s \ll \sigma_i] \xrightarrow{a} [\sigma_u \ll s \ll \sigma_i \cdot a], \qquad a \in L_I \qquad (A1)$$

$$[x \cdot \sigma_u \ll s \ll \sigma_i] \xrightarrow{x} [\sigma_u \ll s \ll \sigma_i], \qquad x \in L_U \qquad (A2)$$

$$\frac{s \xrightarrow{\tau} s'}{[\sigma_u \ll s \ll \sigma_i] \xrightarrow{\tau} [\sigma_u \ll s' \ll \sigma_i]} \qquad (I1)$$

$$\frac{s \xrightarrow{a} s' \quad a \in L_I}{[\sigma_u \ll s \ll a \cdot \sigma_i] \xrightarrow{\tau} [\sigma_u \ll s' \ll \sigma_i]} \qquad (I2)$$

$$\frac{s \xrightarrow{x} s' \quad x \in L_U}{[\sigma_u \ll s \ll \sigma_i] \xrightarrow{\tau} [\sigma_u \cdot x \ll s' \ll \sigma_i]} \quad \textit{(I3)}$$

We abbreviate $[\langle\rangle \ll s \ll \langle\rangle]$ to $Q(s)$. Given an initial state s_0 of an IOLTS M, the initial state of M in queue context is given by $Q(s_0)$.

Observe that for an arbitrary IOLTS M with initial state s_0, $Q(s_0)$ is again an IOLTS. We have the following property, relating the traces of an IOLTS to the traces it has in the queued context.

Property 2. Let $\langle S, L, \rightarrow, s_0\rangle$ be an arbitrary IOLTS. Then for all $s, s' \in S$, we have $s \overset{\sigma}{\Longrightarrow} s'$ implies $Q(s) \overset{\sigma}{\Longrightarrow} Q(s')$.

The possibility of internal transitions is not observable to the remote asynchronous observer and hence, in [12, 13], weak quiescence is adopted to denote quiescence in the queue context.

Definition 12 (Synchronous execution in the queue context). *Let $M = \langle S, L, \rightarrow, s_0\rangle$ be an IOLTS, and let $\langle T, L', \rightarrow, t_0\rangle$ be a test case, such that $L_I = L'_U$ and $L_U = L'_I \setminus \{\theta\}$. Let $s, s' \in S$ and $t, t' \in T$. Then the synchronous execution of the test case and $Q(M)$ is defined through the following inference rules:*

$$\frac{[\sigma_u \ll s \ll \sigma_i] \xrightarrow{\tau} [\sigma'_u \ll s' \ll \sigma'_i]}{t \rceil\!\lceil [\sigma_u \ll s \ll \sigma_i] \xrightarrow{\tau} t \rceil\!\lceil [\sigma'_u \ll s' \ll \sigma'_i]} \quad \textit{(R1')}$$

$$\frac{t \xrightarrow{x} t' \qquad [\sigma_u \ll s \ll \sigma_i] \xrightarrow{x} [\sigma'_u \ll s' \ll \sigma'_i]}{t \rceil\!\lceil [\sigma_u \ll s \ll \sigma_i] \xrightarrow{x} t' \rceil\!\lceil [\sigma'_u \ll s' \ll \sigma'_i]} \quad \textit{(R2')}$$

$$\frac{t \xrightarrow{\theta} t' \qquad \delta_q([\sigma_u \ll s \ll \sigma_i])}{t \rceil\!\lceil [\sigma_u \ll s \ll \sigma_i] \xrightarrow{\theta} t' \rceil\!\lceil [\sigma_u \ll s \ll \sigma_i]} \quad \textit{(R3')}$$

The property below characterizes the relation between the test runs obtained by executing an internal choice test case in the synchronous setting and by executing a test case in the queued setting.

Property 3. Let $\langle S, L, \rightarrow, s_0\rangle$ be an IOLTS and let $\langle T, L', \rightarrow, t_0\rangle$ be a TTS$^{\sqcap}$. Consider arbitrary states $s, s' \in S$ and $t, t' \in T$ and an arbitrary test run $\sigma \in L'^*$. We have the following properties:

1. $t \rceil\!\lceil s \overset{\sigma}{\Longrightarrow} t' \rceil\!\lceil s'$ implies $t \rceil\!\lceil Q(s) \overset{\sigma}{\Longrightarrow} t' \rceil\!\lceil Q(s')$
2. $\mathbf{Sinit}(t \rceil\!\lceil s) = \mathbf{Sinit}(t \rceil\!\lceil Q(s))$.

The proposition below proves to be essential in establishing the correctness of our main results in the remainder of Section 3. It essentially establishes the links between the internal behaviors of an implementation in the synchronous and the asynchronous settings.

Proposition 1. *Let $\langle S, L, \rightarrow, s_0 \rangle$ be an IOLTS and let $\langle T, L', \rightarrow, t_0 \rangle$ be a TTS$^\sqcap$. For all states $t \in T$, $s, s' \in S$, all $\sigma_i \in L_I^*$ and $\sigma_u \in L_U^*$, we have:*

1. $s \overset{\epsilon}{\Longrightarrow} s'$ *iff* $t \rceil\!\| s \overset{\epsilon}{\Longrightarrow} t \rceil\!\| s'$ *(R1*)*
2. $[\sigma_u \ll s \ll \sigma_i] \overset{\epsilon}{\Longrightarrow} [\sigma_u \ll s' \ll \sigma_i]$ *iff* $s \overset{\epsilon}{\Longrightarrow} s'$*(I1*).*

3.3 Sound Verdicts of Internal Choice Test Cases

In [13, 6], it is argued that providing inputs to an IUT only after observing quiescence (i.e., in a stable state), eliminates the distortions in observable behavior, introduced by communicating to the IUT using queues. Hence, a subset of synchronous test-cases, namely those which only provide an input after observing quiescence, are safe for testing asynchronous systems. This is summarized in the following (non)theorem from [13, 12] (and paraphrased in [6]):

Claim (Theorem 1 in [13]). Let M be an arbitrary IOTS$^\sqcap$ with initial state s_0, and let $\langle T, L, \rightarrow, t_0 \rangle$ be a TTS$^\sqcap$. Then s_0 $\mathbf{passes}$ t_0 iff $Q(s_0)$ $\mathbf{passes}$ t_0.

In [6], the claim is taken for granted, and, unfortunately, in [13, 12] only a proof sketch is provided for the above claim; the proof sketch is rather informal and leaves some room for interpretation, as illustrated by the following excerpt:

> "...An implementation guarantees that it will not send any output before receiving an input after quiescence is observed..."

As it turns out, the above result does not hold in its full generality, as illustrated by the following example.

Example 3. Consider the internal choice test case with initial state t_0 in Figure 3. Consider the implementation modeled by the IOTS$^\sqcap$ depicted in Figure 2, starting in state o_0. Clearly, we find that o_0 $\mathbf{passes}$ t_0; however, in the asynchronous setting, $Q(o_o)$ $\mathbf{passes}$ t_0 does not hold. This is due to the divergence in the implementation, which gives rise to an observation of quiescence in the queued context, but not so in the synchronous setting.

The claim *does* hold for non-divergent internal choice implementations. Note that divergence is traditionally also excluded from testing theories such as **ioco**. In this sense, assuming non-divergence is no restriction. Apart from the following theorem, we tacitly assume in all our formal results to follow that the implementation IOLTSs are non-divergent.

Theorem 1. *Let $M = \langle S, L, \rightarrow, s_0 \rangle$ be an arbitrary IOTS$^\sqcap$ and let $\langle T, L' \cup \{\tau\}, \rightarrow, t_0 \rangle$ be a TTS$^\sqcap$. If M is non-divergent, then s_0 $\mathbf{passes}$ t_0 iff $Q(s_0)$ $\mathbf{passes}$ t_0.*

Given the pervasiveness of the original (non-)theorem, a formal correctness proof of our corrections to this theorem (i.e., *our* Theorem 1) is highly desirable. In the remainder of this section, we therefore give the main ingredients for establishing a full proof for Theorem 1. We start by establishing a formal correspondence between observations of quiescence in the synchronous setting and observations of *weak* quiescence in the asynchronous setting. (Due to space limit, some proofs are omitted here, but can be found in the technical report [3].)

Lemma 1. *Let $\langle S, L, \rightarrow, s_0 \rangle$ be an IOTS$^{\sqcap}$. Let $s \in S$ be an arbitrary state. Then $\delta_q(Q(s))$ implies $\delta(s')$ for some $s' \in S$ satisfying $s \overset{\epsilon}{\Longrightarrow} s'$.*

The above lemma results that all inputs a TTS$^{\sqcap}$ gives as stimuli to an implementation, modeled as an IOTS$^{\sqcap}$, can be consumed. Note that this is a non-trivial statement, given that an IOTS$^{\sqcap}$ is not input-enabled in all states. The proposition below as a consequence of the given property, states that every test execution can lead to a state in which both communication queues are empty.

Proposition 2. *Let $\langle S, L, \rightarrow, s_0 \rangle$ be an IOTS$^{\sqcap}$, and let $\langle T, L', \rightarrow, t_0 \rangle$ be a TTS$^{\sqcap}$. Assume arbitrary states $t' \in T$ and $s, s' \in S$, and an arbitrary test run $\sigma \in L'^*$. Then for all $\sigma_i \in L_I^*$ and $\sigma_u \in L_U^*$:*

$$t_0 \rceil\!| Q(s) \overset{\sigma}{\Longrightarrow} t' \rceil\!|_{[\sigma_u \ll s' \ll \sigma_i]} \textit{ implies } \exists s'' \in S \bullet t_0 \rceil\!| Q(s) \overset{\sigma}{\Longrightarrow} t' \rceil\!| Q(s'')$$

As a consequence of the above proposition, we find the following corollary. It states that each asynchronous test execution can be chopped into individual observations such that before and after each observation the communication queue is empty.

Corollary 1. *Let $\langle S, L, \rightarrow, s_0 \rangle$ be an IOTS$^{\sqcap}$, and let $\langle T, L', \rightarrow, t_0 \rangle$ be a TTS$^{\sqcap}$. Assume arbitrary states $t' \in T$ and $s, s' \in S$, and an arbitrary test run $\sigma \in L'^*$ and $x \in L'$. Then $t_0 \rceil\!| Q(s) \overset{\sigma \cdot x}{\Longrightarrow} t' \rceil\!| Q(s')$ implies $\exists t'' \in T, s'' \in S \bullet t_0 \rceil\!| Q(s) \overset{\sigma}{\Longrightarrow} t'' \rceil\!| Q(s'') \overset{x}{\Longrightarrow} t' \rceil\!| Q(s')$. Moreover, if $x = \theta$ then $\delta_q(Q(s'))$.*

The lemma below establishes a correspondence between the test runs that can be executed in the asynchronous setting and those runs one would obtain in the synchronous setting. The lemma is basic to the correctness of our main results in this section.

Lemma 2. *Let $\langle S, L, \rightarrow, s_0 \rangle$ be an IOTS$^{\sqcap}$, and let $\langle T, L', \rightarrow, t_0 \rangle$ be a TTS$^{\sqcap}$. Let $s, s' \in S$ and $t' \in T$ be arbitrary states. Then, for all $\sigma \in L'^*$, such that $t_0 \rceil\!| Q(s) \overset{\sigma}{\Longrightarrow} t' \rceil\!| Q(s')$, there is a non-empty set $\mathbb{S} \subseteq \{s'' \in S \mid s' \overset{\epsilon}{\Longrightarrow} s''\}$ such that*

1. *$\{s'' \in S \mid \delta(s'') \wedge s' \overset{\epsilon}{\Longrightarrow} s''\} \subseteq \mathbb{S}$ if $\exists \sigma' \in L'^* \bullet \sigma = \sigma' \cdot \theta$*
2. *$s' \in \mathbb{S}$ if $\nexists \sigma' \in L'^* \bullet \sigma = \sigma' \cdot \theta$*
3. *$\forall s'' \in \mathbb{S} \bullet t_0 \rceil\!| s \overset{\sigma}{\Longrightarrow} t' \rceil\!| s''$.*

Proof. We prove this lemma by induction on the length of $\sigma \in L'^*$.

– Induction basis. Assume that the length of σ is 0, i.e., $\sigma = \epsilon$. Assume that $t_0 \rceil\!| Q(s) \overset{\epsilon}{\Longrightarrow} t_0 \rceil\!| Q(s')$. By Proposition 1(2) we have $s \overset{\epsilon}{\Longrightarrow} s'$. Set $\mathbb{S} = \{s'' \mid s' \overset{\epsilon}{\Longrightarrow} s''\}$. Let $s'' \in \mathbb{S}$ be an arbitrary state. Proposition 1(1) leads to $t_0 \rceil\!| s \overset{\epsilon}{\Longrightarrow} t_0 \rceil\!| s'$ and $t_0 \rceil\!| s' \overset{\epsilon}{\Longrightarrow} t_0 \rceil\!| s''$; by transitivity, we have the desired $t_0 \rceil\!| s \overset{\epsilon}{\Longrightarrow} t_0 \rceil\!| s''$. It is also clear that $s' \in \mathbb{S}$. We thus find that $\mathbb{S}$ meets the desired conditions.
– Inductive step. Assume that the statement holds for all σ' of length at most $n - 1$. Suppose that the length of σ is n. Assume that $t_0 \rceil\!| Q(s) \overset{\sigma}{\Longrightarrow} t' \rceil\!| Q(s')$. By Corollary 1, there is some $s_{n-1} \in S$, a $t_{n-1} \in T$ and $\sigma_{n-1} \in L'^*$ and $x \in L'$, such that $\sigma = \sigma_{n-1} \cdot x$ and $t_0 \rceil\!| Q(s) \overset{\sigma_{n-1}}{\Longrightarrow} t_{n-1} \rceil\!| Q(s_{n-1}) \overset{x}{\Longrightarrow} t' \rceil\!| Q(s')$.
By induction, there must be a set $\mathbb{S}_{n-1} \subseteq \{s'' \in S \mid s_{n-1} \overset{\epsilon}{\Longrightarrow} s''\}$, such that

1. $\{s'' \in S \mid \delta(s'') \wedge s_{n-1} \overset{\epsilon}{\Longrightarrow} s''\} \subseteq \mathbb{S}_{n-1}$ if $\exists \sigma' \in L'^* \bullet \sigma = \sigma' \cdot \theta$
2. $s_{n-1} \in \mathbb{S}_{n-1}$ if $\nexists \sigma' \in L'^* \bullet \sigma = \sigma' \cdot \theta$
3. $\forall s'' \in \mathbb{S}_{n-1} \bullet t_0]\!|s \overset{\sigma_{n-1}}{\Longrightarrow} t_{n-1}]\!|s''$.

We next distinguish three cases: $x \in L_I$, $x \in L_U$ and $x \notin L_I \cup L_U$.

1. Case $x = \theta$. We thus find that $t_{n-1}]\!|Q(s_{n-1}) \overset{\theta}{\Longrightarrow} t_n]\!|Q(s')$. As a result of Corollary 1, we have $\delta_q(s')$. We then find as a result of Lemma 1, there must be some state $s'' \in S$ such that $s_{n-1} \overset{\epsilon}{\Longrightarrow} s' \overset{\epsilon}{\Longrightarrow} s''$ and $\delta(s'')$. Consider the set $\mathbb{S}_n = \{s'' \in S \mid \delta(s'') \wedge s' \overset{\epsilon}{\Longrightarrow} s''\}$.
   Let s'' be an arbitrary state in $\mathbb{S}_n$. Distinguish between cases $s_{n-1} \notin \mathbb{S}_{n-1}$ and $s_{n-1} \in \mathbb{S}_{n-1}$. In the case, $s_{n-1} \notin \mathbb{S}_{n-1}$, we know from the construction of $\mathbb{S}_{n-1}$ that $s'' \in \mathbb{S}_{n-1}$ and $s'' \overset{\epsilon}{\Longrightarrow} s''$ always holds. In the case $s_{n-1} \in \mathbb{S}_{n-1}$, we have that $s_{n-1} \overset{\epsilon}{\Longrightarrow} s' \overset{\epsilon}{\Longrightarrow} s''$. We thus find that $\forall s'' \in \mathbb{S}_n \; \exists \bar{s} \in \mathbb{S}_{n-1} \bullet t_0]\!|s \overset{\sigma_{n-1}}{\Longrightarrow} t_{n-1}]\!|\bar{s} \overset{\epsilon}{\Longrightarrow} t_{n-1}]\!|s'' \xrightarrow{\theta} t']\!|s''$.
   Thus $\mathbb{S}_n$ has the desired requirement that $t_0]\!|s \overset{\sigma_{n-1} \cdot x}{\Longrightarrow} t']\!|s''$ for all $s'' \in \mathbb{S}_n$. Also, $\{s'' \in S \mid \delta(s'') \wedge s' \overset{\epsilon}{\Longrightarrow} s''\} \subseteq \mathbb{S}_n$ is concluded from construction of $\mathbb{S}_n$. Hence, $\mathbb{S}_n$ satisfies all desired conditions.
2. Case $x \in L_I$. By Property 1, we find that the last step in σ_{n-1} must be θ. It follows from corollary 1 that $Q(s_{n-1})$ is weakly quiescent and consequently $\delta_q(s_{n-1})$. By induction we have that $\{s'' \in S \mid \delta(s'') \wedge s_{n-1} \overset{\epsilon}{\Longrightarrow} s''\} \subseteq \mathbb{S}_{n-1}$. Consider the set $\mathbb{S}_n = \{s'' \in S \mid s' \overset{\epsilon}{\Longrightarrow} s''\}$.
   Transition $t_{n-1}]\!|Q(s_{n-1}) \overset{x}{\Longrightarrow} t']\!|Q(s')$ implies that $s_{n-1} \overset{x}{\Longrightarrow} s'$. By Lemma 1 and Definition 6, we know that $\exists \bar{s} \in S$ such that $s_{n-1} \overset{\epsilon}{\Longrightarrow} \bar{s} \overset{x}{\Longrightarrow} s'$ and $\delta(\bar{s})$. From construction of $\mathbb{S}_{n-1}$, we know that $\bar{s}$ is in $\mathbb{S}_{n-1}$. We thus have $\forall s'' \in \mathbb{S}_n \; \exists \bar{s} \in \mathbb{S}_{n-1} \bullet t_0]\!|s \overset{\sigma_{n-1}}{\Longrightarrow} t_{n-1}]\!|\bar{s} \overset{x}{\Longrightarrow} t']\!|s''$.
   It is clear form construction of $\mathbb{S}_n$ that $s' \in \mathbb{S}_n$ as the required condition that $s' \in \mathbb{S}_n$ if the last step of σ is not θ-labeled transition. We thus find that $\mathbb{S}_n$ fulfills all desired requirements.
3. Case $x \in L_U$. Analogous to the previous case.

We are now in a position to establish the correctness of Theorem 1. We provide the proof below:

Proof (Theorem 1). We prove the theorem by contraposition.

1. Case $\Rightarrow$. Suppose not $Q(s)$ **passes** t_0. By Definition 9 and Proposition 2, $t_0]\!|Q(s) \overset{\sigma'}{\Longrightarrow}$ **fail**$]\!|Q(s')$, for some $\sigma' \in L'^*$ and $s' \in S$. As a result of Lemma 2, there is a non-empty set $\mathbb{S} \subseteq \{s'' \in S \mid s' \overset{\epsilon}{\Longrightarrow} s''\}$ such that for all $s'' \in \mathbb{S}$, $t_0]\!|s \overset{\sigma'}{\Longrightarrow}$ **fail**$]\!|s''$, which was what we needed to prove.
2. Case $\Leftarrow$. Assume, that not s **passes** t_0. Then there are $\sigma' \in L'^*$ and $s'' \in S$, $t_0]\!|s \overset{\sigma'}{\Longrightarrow}$ **fail**$]\!|s''$. Using Property 3 leads to $t_0]\!|Q(s) \overset{\sigma'}{\Longrightarrow}$ **fail**$]\!|Q(s'')$.

4 Adapting Asynchronous Setting to IOCO

In this section, we re-cast the results of the previous section to the setting with **ioco** test-cases. We first define **ioco** and then show that the results of the previous section cannot be trivially generalized to the **ioco**-setting. Then using an approach inspired by [7, Chapter 5] and [6], we show how to re-formulate Theorem 1 in this setting.

4.1 Input Output Conformance

The **ioco** testing theory formalizes the conformance of an implementation to its specification. In this theory, implementations are assumed to behave according to an (unkown) IOTS; as a consequence, implementations are assumed to be input enabled. Contrary to implementations, specifications are not required to be input enabled; this facilitates under-specifying the behavior of a system. Informally, the **ioco** conformance relation captures whether the observable behaviors of the implementation are valid observable behaviors, given a specification. The observable behaviors are essentially augmented traces, called *suspension traces*, consisting of inputs, outputs and quiescence.

For a given set of states S of an arbitrary IOLTS with transition relation $\rightarrow \subseteq S \times (L \cup \{\tau\}) \times S$, suspension traces are defined through an auxiliary transition relation $\Longrightarrow_\delta \subseteq S \times (L \cup \{\delta\})^* \times S$, specified by the following deduction rules:

$$\frac{}{s \overset{\epsilon}{\Longrightarrow}_\delta s} \qquad \frac{s \overset{\sigma}{\Longrightarrow}_\delta s' \qquad \delta(s')}{s \overset{\sigma\delta}{\Longrightarrow}_\delta s'} \qquad \frac{s \overset{\sigma}{\Longrightarrow}_\delta s'' \qquad s'' \overset{x}{\Longrightarrow} s'}{s \overset{\sigma x}{\Longrightarrow}_\delta s'}$$

Henceforth, given an alphabet L, we write L_δ to denote the set $L \cup \{\delta\}$.

Definition 13 (Suspension traces, Out and After). *Let $\langle S, L, \rightarrow, s_0\rangle$ be an IOLTS. Let $s \in S$ be an arbitrary state, $S' \subseteq S$ and $\sigma \in L_\delta^*$.*

1. *The set of* suspension traces *of s, denoted* $\mathsf{Straces}(s)$ *is the set* $\{\sigma \in L_\delta^* \mid s \overset{\sigma}{\Longrightarrow}_\delta\}$*; we set* $\mathsf{Straces}(S') = \bigcup_{s' \in S'} \mathsf{Straces}(s')$
2. *The* outputs *of s, denoted* $\mathbf{out}(s)$ *is the set* $\{x \in L_U \mid s \xrightarrow{x}\} \cup \{\delta \mid \delta(s)\}$*; we set* $\mathbf{out}(S') = \bigcup_{s' \in S'} \mathbf{out}(s')$
3. *The* σ-reachable states *of s, denoted* $s\,\mathbf{after}\,\sigma$ *is the set* $\{s' \in S \mid s \overset{\sigma}{\Longrightarrow}_\delta s'\}$*; we set* $S'\,\mathbf{after}\,\sigma = \bigcup_{s' \in S'} s'\,\mathbf{after}\,\sigma$.

The above abbreviations are used in the intensional characterization of the **ioco** testing relation, given below.

Definition 14 (ioco). *Let $\langle I, L, \rightarrow, i_0\rangle$ be an IOTS, and let IOLTS $\langle S, L, \rightarrow, s_0\rangle$ be a specification. We say that implementation i_0 is* input-output conform *specification s_0, denoted $i_0\,\mathbf{ioco}\,s_0$, iff*

$$\forall \sigma \in \mathsf{Straces}(s_0) \bullet \mathbf{out}(i_0\,\mathbf{after}\,\sigma) \subseteq \mathbf{out}(s_0\,\mathbf{after}\,\sigma)$$

The **ioco** testing relation has been shown to admit a sound and complete test case generation algorithm, see, e.g., [9]. Soundness means, intuitively, that the algorithm will never generate a test case that, when executed on an implementation, leads to a *fail* verdict if the test runs are in accordance with the specification. Completeness is more esoteric: if the implementation has a behavior that is not in line with the specification, then there is a test case that, in theory, has the capacity to detect that non-conformance.

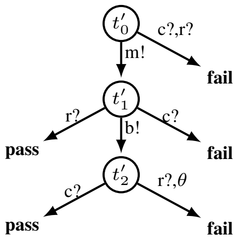


Fig. 4. An **ioco** test case

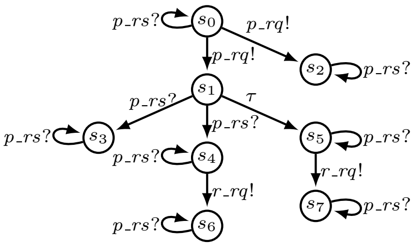


Fig. 5. A delay right-closed IOTS

As the exact workings of the algorithm are impertinent to our main results in this section, we will forego an explanation of it.

In the following example, we motivate that the definitions and the constraints used in the previous section cannot be used for the **ioco** setting.

Example 4. Figure 4 shows a test case for IOLTS o_0 in Figure 2, which is an internal choice IOTS. Assume that at the same time o_0 is also used as the implementation; o_0 is not input-enabled in all states, and making it input-enabled violates the internal choice assumption. In fact, as observed in Section 2, the intersection of IOTSs and internal choice IOTSs only include pathological IOTSs that do not produce any output. For the purpose of this example, we use the theory of **ioco** on internal choice IOTSs nevertheless.

For o_0 as specification and implementation, we have that o_0 **ioco** o_0. However, we can reach a fail verdict for o_0 under the queue context when using the test case t'_0. Consider the sequence $m?b?r!$; in the queue context, the execution $t'_0 \rceil\!\rceil Q(o_0) \xrightarrow{m?} t'_1 \rceil\!\rceil_{[\epsilon \ll o_0 \ll m?]} \overset{\epsilon}{\Longrightarrow} t'_1 \rceil\!\rceil Q(o_1) \overset{\epsilon}{\Longrightarrow} t'_1 \rceil\!\rceil_{[r! \ll o_2 \ll \epsilon]} \xrightarrow{b?} t'_2 \rceil\!\rceil_{[r! \ll o_2 \ll b?]} \xrightarrow{r!} \mathbf{fail} \rceil\!\rceil_{[\epsilon \ll o_2 \ll b?]}$ is possible, which leads to the **fail** state. Note that the fail verdict is reached even if we omit divergence from the implementation o_0. This shows that Theorem 1 cannot be trivially generalized to the **ioco** setting (even when excluding divergence and allowing for non-input-enabled states).

4.2 Synchronizing Theorem for ioco

In this section, we investigate implementations for which **ioco** test cases cannot distinguish between synchronous and asynchronous modes of testing. To this end, we consider the relation between traces of a system and those of the system in queue context.

Definition 15 (Delay relation). *Let L be a finite alphabet partitioned in L_I and L_U. The* delay relation $@ \subseteq L_\delta^* \times L_\delta^*$ *is defined by the following deduction rules:*

$$\frac{}{\sigma \mathrel{@} \sigma}\ REF \qquad \frac{\rho_i, \sigma_i \in L_I^* \qquad \sigma_u \in L_U^*}{\rho_i \cdot \sigma_u \cdot \sigma_i \mathrel{@} \rho_i \cdot \sigma_i \cdot \sigma_u}\ PUSH \qquad \frac{\sigma \mathrel{@} \sigma' \qquad \rho @ \rho'}{\sigma \cdot \rho \mathrel{@} \sigma' \cdot \rho'}\ COM$$

Proposition 3. *Let $\langle S, L, \rightarrow, s_0 \rangle$ be an IOTS. Let $s \in S$ and $\sigma \in L_\delta^*$. Then $\sigma \in$* $\mathsf{Straces}(Q(s))$ *implies there is a $\sigma' \in$* $\mathsf{Straces}(s)$ *such that $\sigma' \mathrel{@} \sigma$.*

Definition 16 (Delay right-closed IOTS). *Let $M = \langle S, L, \rightarrow, s_0 \rangle$ be an IOTS. A set $L' \subseteq L_\delta^*$ is* delay right-closed *iff for all $\sigma \in L'$ and $\sigma' \in L_\delta^*$, if $\sigma @ \sigma'$ then $\sigma' \in L'$. The IOTS M is* delay right-closed *iff* $\mathsf{Straces}(s_0)$ *is delay right-closed.*

We denote the class of delay right-closed IOTSs ranging over L_I and L_U by $\mathsf{IOTS}^{@}(L_I, L_U)$. The property below gives an alternative characterisation of delay right-closed IOTSs.

Property 4. Let $M = \langle I, L, \rightarrow, i_0 \rangle$ be an IOTS. The IOTS M is *delay right-closed* if for all $\sigma \in L_\delta^*$, all $x \in L_U$ and $a \in L_I$, we have:

$$\sigma \cdot x \cdot a \in \mathsf{Straces}(i_0) \text{ then } \sigma \cdot a \cdot x \in \mathsf{Straces}(i_0)$$

Example 5. Consider the IOTS s_0 given in Figure 5. It is not hard to check that s_0 is delay right-closed.

As stated in the following theorem, the verdicts obtained by executing an arbitrary test case on a delay right-closed IOTS do not depend on the execution context. That is, the verdict does not change when the communication between the implementation and the test case is synchronous or asynchronous.

Theorem 2. *Let $\langle I, L, \rightarrow, i_0 \rangle$ be a delay right-closed IOTS and let $\langle T, L', \rightarrow, t_0 \rangle$ be an arbitrary test case. Then i_0* **passes** *t_0 iff $Q(i_0)$* **passes** *t_0.*

Before we address the proof of the above theorem, we first establish the correctness of the lemma below, stating that the suspension traces of a delay right-closed IOTS, as observed in an asynchronous setting are indistinguishable from the set of suspension traces observable in the synchronous setting.

Lemma 3. *Let $\langle S, L, \rightarrow, s_0 \rangle$ be a delay right-closed IOTS. Then* $\mathsf{Straces}(Q(s_0)) = \mathsf{Straces}(s_0)$.

Proof. We divide the proof obligation into two parts: $\mathsf{Straces}(Q(s_0)) \subseteq \mathsf{Straces}(s_0)$ and $\mathsf{Straces}(s_0) \subseteq \mathsf{Straces}(Q(s_0))$. It is not hard to verify that the latter holds vacuously, even for arbitrary IOTSs.

It therefore remains to show that $\mathsf{Straces}(Q(s_0)) \subseteq \mathsf{Straces}(s_0)$. Consider a $\sigma \in \mathsf{Straces}(Q(s_0))$; by Proposition 3, $\exists \sigma' \in \mathsf{Straces}(s_0) \bullet \sigma' @ \sigma$. As s_0 is delay right-closed, we obtain the required $\sigma \in \mathsf{Straces}(s_0)$.

The above lemma is at the basis of the correctness of Theorem 2.

Proof (Theorem 2). Using the lemma given above, the proof of the theorem follows from the observation that for all test cases $\langle T, L', \rightarrow, t_0 \rangle$ and all $\sigma \in L'^*$:

$$\exists i' \in I \bullet t_0 \rceil\!| i_0 \overset{\sigma}{\Longrightarrow} \mathbf{fail} \rceil\!| i' \text{ iff } \exists i' \in I, \sigma_i \in L_I^*, \sigma_u \in L_U^* \bullet t_0 \rceil\!| Q(i_0) \overset{\sigma}{\Longrightarrow} \mathbf{fail} \rceil\!|_{[\sigma_u \ll} i'_{\ll \sigma_i]}$$

Theorem 3. *Let $\langle I, L, \rightarrow, i_0 \rangle$ be a delay right-closed IOTS and let IOLTS $\langle S, L, \rightarrow, s_0 \rangle$ be a specification. Then i_0* **ioco** *s_0 iff $Q(i_0)$* **ioco** *s_0.*

Proof. Follows from the existence of a sound and complete test suite that can test for **ioco**, and the proof of Theorem 2.

5 Necessary and Sufficient Conditions

In the previous section, we presented a class of implementation, called delay right-closed, whose synchronous and asynchronous test executions lead to the same verdict. We now show that delayed right-closedness of implementations is also a necessary condition to ensure the same verdict in the synchronous and the asynchronous setting.

Theorem 4. *Let $M = \langle I, L, \rightarrow, i_0 \rangle$ be an IOTS. If for every test case $\langle T, L', \rightarrow, t_0 \rangle$, we have i_0 $\mathbf{passes}$ $t_0 \Leftrightarrow Q(i_0)$ $\mathbf{passes}$ t_0, then M is a delay right-closed IOTS.*

Proof. We prove the theorem by contraposition, i.e., we show that if we test a non-delay right-closed IOTS, there is a test case that can detect this by giving a *pass* verdict in the synchronous setting but a *fail* verdict in the asynchronous setting.

Let $\langle I, L, \rightarrow, i_0 \rangle$ be an IOTS that is not delay right-closed. Thus, there is some $x \in L_U$, $a \in L_I$ such that $\sigma \cdot x \cdot a \in \mathsf{Straces}(i_0)$, but not $\sigma \cdot a \cdot x \in \mathsf{Straces}(i_0)$. Let $\langle T, L', \rightarrow, t_0 \rangle$ be a test case such that there is a $t' \in T$ satisfying:

1. $t_0 \overset{\sigma}{\Longrightarrow} t'$,
2. $t' \xrightarrow{a} t''$, and $t'' \xrightarrow{x} \mathbf{fail}$.
3. for all σ' such that $t_0 \overset{\sigma'}{\Longrightarrow} \mathbf{fail}$ we have $\sigma' = \sigma \cdot a \cdot x$.

Observe that the existence of such a test case is immediate. Then there are $\sigma_i \in L_I^*$, $\sigma_u \in L_U^*$ and a state $i \in (i_0 \,\mathbf{after}\, \sigma)$ such that $t_0]\!] Q(i_0) \overset{\sigma \cdot a \cdot x}{\Longrightarrow} \mathbf{fail}]\!]_{[\sigma_u \ll i \ll \sigma_i \cdot a]}$, i.e., not $Q(i_0)$ $\mathbf{passes}$ t_0. However, we do not have $t_0]\!] i_0 \overset{\sigma \cdot a \cdot x}{\Longrightarrow} \mathbf{fail}]\!] i$. By construction of the test case, we find that i_0 $\mathbf{passes}$ t_0.

6 Conclusions

In this paper, we presented theorems which allow for using test-cases generated from ordinary specifications in order to test asynchronous systems. These theorems establish sufficient conditions when the verdict reached by testing the asynchronous system (remotely, through FIFO channels) corresponds with the local testing through synchronous interaction. In the case of **ioco** testing theory, we show that the presented sufficient conditions are also necessary.

It remains to find an intensional characterization of the notion of conformance induced by the class of test-cases generated in the approach of [13]. The presented conditions for synchronizing **ioco** are semantic in nature and we intend to formulate syntactic conditions that imply the semantic conditions presented in this paper. For example, it is interesting to find out which composition of programming constructs and / or patterns of interaction satisfy the constraints established in this paper. The research reported in this paper is inspired by our practical experience with testing asynchronous systems reported in [1]. We plan to apply the insights obtained from this theoretical study to our practical cases and find out to what extent the constraints of this paper apply to the implementation of our case studies.

Acknowledgments. We would like to thank Sjoerd Cranen (TU/e) and Maciej Gazda (TU/e) for their useful comments and suggestions.

References

1. H.R. Asadi, R. Khosravi, M.R. Mousavi, and N. Noroozi. Towards model-based testing of electronic funds transfer systems. In *Proc. of FSEN 2011*, LNCS. Springer, 2011.
2. C. Jard, T. Jéron, L. Tanguy, and C. Viho. Remote testing can be as powerful as local testing. In *Proc. of FORTE XII*, volume 156 of *IFIP Proc.*, pp. 25–40. Kluwer, 1999.
3. N. Noroozi, R. Khosravi,M.R. Mousavi,T.A.C. Willemse. Synchronizing Asynchronous Conformance Testing. Computer Science Report, No. 11-10, 16 pp. Eindhoven: Technische Universiteit Eindhoven, 2011.
4. A. Petrenko and N. Yevtushenko. Queued testing of transition systems with inputs and outputs. In *Proc. of FATES 2002*, pp. 79–93, 2002.
5. A. Petrenko, N. Yevtushenko, and J. Huo. Testing transition systems with input and output testers. In *Proc. of Testcom 2003*, volume 2644 of *LNCS*, pp. 129–145. Springer, 2003.
6. A. Simao and A. Petrenko. From test purposes to asynchronous test cases. In *Proc. of ICSTW 2010*, pp. 1–10. IEEE CS, 2010.
7. J. Tretmans. *A formal Approach to conformance testing*. PhD thesis, Univ. of Twente, The Netherlands, 1992.
8. J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. *Software—Concepts and Tools*, 3:103–120, 1996.
9. J. Tretmans. Model based testing with labelled transition systems. In *Formal Methods and Testing*, volume 4949 of *LNCS*, pp. 1–38. Springer, 2008.
10. J. Tretmans and L. Verhaard. A queue model relating synchronous and asynchronous communication. In *Proc. of PSTV'92*, vol. C-8 of *IFIP Tr.*, pp. 131-145. North-Holland, 1992.
11. L. Verhaard, J. Tretmans, P. Kars, and E. Brinksma. On asynchronous testing. In *Proc. of IWPTS'93*, volume C-11 of *IFIP Tr.*, pp. 55–66. North-Holland, 1993.
12. M. Weiglhofer. *Automated Software Conformance Testing*. PhD thesis, TU Graz, 2009.
13. M. Weiglhofer and F. Wotawa. Asynchronous input-output conformance testing. In *Proc. of COMPSAC'09*, pp. 154–159. IEEE CS, 2009.