Queued Testing of Transition Systems with Inputs and Outputs

Alex Petrenko[a] and Nina Yevtushenko[b]

a - CRIM, Centre de recherche informatique de Montreal,
550 Sherbrooke West, Suite 100, Montreal, H3A 1B9 Canada,
E-mail: Petrenko@crim.ca

b - Tomsk State University, 36 Lenin Str., Tomsk, 634050, Russia,
E-mail: Yevtushenko@elefot.tsu.ru

Abstract The paper studies testing based on input/output transition systems, also known as input/output automata. It is assumed that a tester can never prevent a system under test from producing outputs, while the system does not block inputs from the tester either. Thus, input from the tester and output from the system may occur simultaneously and should be queued in finite buffers between the tester and system. A framework for a so-called queued-quiescence testing is developed, based on the idea that the tester should consist of two test processes, one process is applying inputs via a queue to a system under test and another one is reading outputs from a queue until it detects no more outputs from the system, i.e., the tester detects quiescence in the system. The testing framework is then generalized with a so-called queued-suspension testing by considering a tester that has several pairs of input and output processes. It is demonstrated that such a tester can check finer implementation relations than a queued-quiescence tester. Procedures for test derivation are proposed for a given fault model comprising possible implementations.

1. Introduction

The problem of deriving tests from state-oriented models that distinguish between input and output actions is usually addressed with one of the two basic assumptions about the relationships between inputs and outputs. Assuming that a pair of input and output constitutes an atomic system's action, in other words, that a system cannot accept next input before producing output as a reaction to a previous input, one relies on the input/output Finite State Machine (FSM) model. There is a large body of work on test generation from FSM with various fault models and test architectures, for references see, e.g., [Petr01], [BoPe94]. A system, where next input can arrive even before output is produced in response to a previous input, is usually modeled by the input/output automaton model [LyTu89], also known as the input/output transition system (IOTS) model (the difference between them is marginal, at least from the testing perspective). Compared to the FSM model, this model has received a far less attention in the testing community, see, e.g., [BrTr01], [Phal93], [Sega93]. In this paper, we consider the IOTS model and take a close look on some basic assumptions underlying the existing IOTS testing frameworks.

One of the most important works on test generation from labeled transition systems with inputs and outputs is [Tret96]. In this paper, it is assumed that a tester (implementing a given test case) and an IOTS interact as two labeled transition systems and not as IOTS.

Accordingly, the LTS composition operator used to formalize this interaction does not distinguish between inputs and outputs, so the tester need not be input-enabled to satisfy compatibility conditions for composing IOTS [LyTu89]. The tester seems to be able to preempt output from the system any time it decides to send input to the system. This allows the tester to avoid choosing between inputs and outputs, keeping the test process deterministic. On the other hand, such a tester appears to be able to override the principle that "output actions can never be blocked by the environment" [Tret96, p.106].

Another assumption about the tester is taken by Tan and Petrenko [TaPe98]. In this work, it is recognized that the tester cannot block the system's outputs, it is only assumed that the tester can detect the situation when it offers input to the system, but the latter, instead of consuming it, issues an output (a so-called "exception"). An exception halts a current test run avoiding thus any further non-deterministic behavior and results in the verdict **inconclusive**. Notice that the tester of [Tret96] has only two verdicts, **pass** and **fail**.

Either approach relies on an assumption that is not always justified in a real testing environment. As an example, consider the situation when the tester cannot directly interact with the IUT, because of a context, such as queue or interface, between them. As pointed out in [dVBF02], to apply the test derivation algorithm of [Tret96], one has to take into account the presence of a queue context. It also states "the assumption that we can synthesize every stimulus and analyze every observation is strong", so that some problems in observing quiescence occur.

The case when IOTS is tested via infinite queues is investigated by Verhaard et al [VTKB92]. The proposed approach relies on an explicit combined specification of a given IOTS and queue context, so it is not clear how this approach could be implemented in practice. This context is also considered in [JJTV99], where a stamping mechanism is proposed to order the outputs with respect to inputs, while quiescence is ignored. A stamping process has to be synchronously composed with the IUT as the tester in [Tret96].

We also notice that we are aware of the only work [TaPe98] that uses fault models in test derivation from IOTS. In [Tret96] and [VTKB92], a test case is derived from a trace provided by the user.

The above discussion indicates a need for another approach that does not rely on such strong assumptions about the testing environment and incorporates a fault model to derive tests that can be characterized in terms of fault detection. In this paper, we report on our preliminary findings in attempts to elaborate such an approach. In particular, we introduce a framework for testing IOTS, assuming that a tester can never prevent a system under test from producing outputs, while the system does not block inputs from the tester either and, thus, input and output actions may occur simultaneously and should be queued in finite buffers between the tester and system.

The paper is organized as follows. In Section 2, we introduce some basic definitions and define a composition operator for IOTS based on a refined notion of compatibility of IOTS first defined in [LyTu89]. Section 3 presents our framework for a so-called queued-quiescence testing, based on the idea that the tester should consist of two test processes, one process is applying inputs to a system under test via a finite input queue and another one is reading outputs that the system puts into a finite output queue until it detects no more outputs from the system, i.e., the tester detects quiescence in the system. We elaborate such a tester and formulate several implementation relations that can be tested with a queued-quiescence tester. In Section 4, we discuss how queued-quiescence tests can be derived for a given specification and fault model that comprises a finite set of implementations. In Section 5, we generalize our testing framework with a so-called queued-suspension testing by

allowing a tester to have several pairs of input and output processes and demonstrate that a queued-suspension tester can check finer implementation relations than a queued-quiescence tester. We conclude by comparing our contributions with previous work and discussing further work.

2. Preliminaries

A *labeled transition system*, or simply a labeled transition system (LTS), is a 4-tuple $L = <S, \Sigma, \lambda, s_0>$, where S is a finite non-empty set of states with the initial state s_0; Σ is a finite set of actions; $\lambda \in S \times \Sigma \times S$ is a transition relation. In this paper, we consider only LTS such that $(s, a, s'), (s, a, s'') \in \lambda$ implies $s' = s''$. These are deterministic LTS.

Let $L_1 = <S, \Sigma_1, \lambda_1, s_0>$ and $L_2 = <T, \Sigma_2, \lambda_2, t_0>$, the *parallel composition* $L_1 \| L_2$ is defined as the LTS $<R, \Sigma_1 \cup \Sigma_2, \lambda, s_0t_0>$, where the set of states $R \subseteq S \times T$ and the transition relation λ are smallest sets obtained by application of the following inference rules:

- if $a \in \Sigma_1 \cap \Sigma_2$, $(s, a, s') \in \lambda_1$, and $(t, a, t') \in \lambda_2$ then $(st, a, s't') \in \lambda$;
- if $a \in \Sigma_1 \backslash \Sigma_2$, $(s, a, s') \in \lambda_1$, then $(st, a, s't) \in \lambda$;
- if $a \in \Sigma_2 \backslash \Sigma_1$, $(t, a, t') \in \lambda_1$, then $(st, a, st') \in \lambda$.

We use the LTS model to define a transition system with inputs and outputs. The difference between these two types of actions is that no system can deny an input action from its environment, while this is completely up to the system when to produce an output, so the environment cannot block the output. Formally, an *input/output transition system* (IOTS) L is a LTS in which the set of actions Σ is partitioned into two sets, the set of input actions I and the set of output actions O. Given state s of L, we further denote $init(s)$ the set of actions defined at s, i.e. $init(s) = \{a \in \Sigma \mid \exists s' \in S ((s, a, s') \in \lambda)\}$. The IOTS is *input-enabled* if each input action is enabled at any state, i.e., $I \subseteq init(s)$ for each s. State s of the IOTS is called *unstable* if there exists $o \in O$ such that $o \in init(s)$. Otherwise, state is *stable*. A sequence $a_1 \ldots a_k$ over the set Σ is called a *trace* of L in state s if there exist states $s_1, \ldots, s_{k+1}$ such that $(s_i, a_i, s_{i+1}) \in \lambda$ for all $i = 1, \ldots, k$ and $s_1 = s$. We use $traces(s)$ to denote the set of traces of L in state s. Following [Vaan91] and [Tret96], we refer to a trace that takes the IOTS from a given state to a stable state as to a *quiescent* trace.

To define a composition of IOTS, we first state compatibility conditions that define when two IOTS can be composed by relaxing the original conditions of [LyTu89]. Note that $L_1 \| L_2$ for IOTS L_1 and L_2 means the synchronous parallel composition of LTS that are obtained from IOTS by neglecting the difference between inputs and outputs, so these IOTS are treated as LTS.

Definition 1. Let $L_1 = <S, \Sigma_1, \lambda_1, s_0>$, $\Sigma_1 = I_1 \cup O_1$, and $L_2 = <T, \Sigma_2, \lambda_2, t_0>$, $\Sigma_2 = I_2 \cup O_2$, be two IOTS such that the sets $I_1 \cap I_2$ and $O_1 \cap O_2$ are empty. Let st be a state of the composition $L_1 \| L_2$. The L_1 and L_2 are *compatible in state st* if

- $a \in init(s)$ implies $a \in init(t)$ for any $a \in I_2 \cap O_1$ and
- $a \in init(t)$ implies $a \in init(s)$ for any $a \in I_1 \cap O_2$.

The L_1 and L_2 are said to be *compatible* if they are compatible in the state s_0t_0; otherwise they are *incompatible*. L_1 and L_2 are *fully compatible* if they are compatible in all the states of the composition $L_1 \| L_2$.

Clearly, two input-enabled IOTS with $I_1 = O_2$ and $I_2 = O_1$ are fully compatible, but the converse is not true. Based on the notion of compatibility we define what we mean by a

parallel composition of two IOTS. Let $IOTS(I, O)$ denote the set of all possible IOTS over the input set I and output set O.

Definition 2. The *composition operator*][: $IOTS(I_1, O_1) \times IOTS(I_2, O_2) \rightarrow IOTS((I_1 \cup I_2) \setminus (O_1 \cup O_2), O_1 \cup O_2)$, where the sets $I_1 \cap I_2$ and $O_1 \cap O_2$ are empty, is defined as follows. Let $L_1 = \langle S, \Sigma_1, \lambda_1, s_0 \rangle$, $\Sigma_1 = I_1 \cup O_1$, and $L_2 = \langle T, \Sigma_2, \lambda_2, t_0 \rangle$, $\Sigma_2 = I_2 \cup O_2$ be compatible IOTS. If L_1 and L_2 are compatible in state st of the composition $L_1 \parallel L_2$, then $st -a\rightarrow s't'$ in $L_1 \parallel L_2$ implies the same transition in $L_1][L_2$. If L_1 and L_2 are incompatible in state st, then there are no outgoing transitions from the state in $L_1][L_2$, i.e., st is a deadlock.

The IOTS $L_1][L_2$ can be obtained from the LTS $L_1 \parallel L_2$ by pruning outgoing transitions from states where the IOTS L_1 and L_2 are not compatible. For fully compatible IOTS, the results of both operators, || and][, coincide.

3. Framework for Queued-Quiescence Testing of IOTS

In a typical testing framework, it is usually assumed that the two systems, an implementation under test (IUT) and tester, form a closed system. This means that if L_1 is a tester, while L_2 is an IOTS modeling the IUT, then $\Sigma_1 = \Sigma_2$, $I_1 = O_2$, and $I_2 = O_1$. To be compatible with any IOTS over the given alphabet $\Sigma_2 = I_2 \cup O_2$ the tester should be input-enabled. However, the input-enableness has two implications on a behavior of the tester. Its behavior becomes infinite since inputs enabled in each state create cycles and non-deterministic since the tester has to choose non-deterministically between input and output. Both features are usually considered undesirable. Testers should be deterministic and have no cycles. The two requirements are contradicting.

It turns out that a tester processing outputs of an IUT separately from inputs could meet both requirements. To achieve this, it is sufficient to decompose the tester into two processes, one for inputs and another for outputs. Intuitively, this could be done as follows. The input test process only sends to the IUT via input buffer a given (finite) number of consecutive test stimuli. In response to the submitted input sequence, the IUT produces outputs that are stored in another (output) buffer. The output test process, that is simply an observer, only accepts outputs of the IUT by reading the output buffer. All the output sequences the specification IOTS can produce in response to the submitted input sequence, should take the output test process into terminal states labeled with the verdict **pass**, while any other output sequence produced by an IUT should take the output test process to a terminal state labeled with the verdict **fail**. Since the notion of a tester is based on the definition of a set of output sequences that the specification IOTS can produce in response to a submitted input sequence, we formalize both notions as follows.

Let L be an IOTS defined over the action set $\Sigma = I \cup O$ and $pref(\alpha)$ denote the set of all the prefixes of a sequence α over the set Σ. The set $pref(\alpha)$ has the empty sequence ε. Also given a set $P \subseteq \Sigma^*$, let $\{\beta \in pref(\gamma) \mid \gamma \in P\} = pref(P)$.

Definition 3. Given an input word $\alpha \in I^*$, the *input test process* with α is a tuple $\alpha = \langle pref(\alpha), \varnothing, I, \lambda_\alpha, \varepsilon \rangle$, where the set of inputs is empty, while the set of outputs is I, $\lambda_\alpha = \{(\beta, a, \beta a) \mid \beta a \in pref(\alpha)\}$, and the initial state is ε.

We slightly abuse α to denote both, the input sequence and the input test process that executes this sequence. It is easy to see that each input test process is fully compatible with any IOTS L that is input-enabled and defined over the set of inputs I. Notice that in this paper, we consider only input-enabled IOTS specifications, while an implementation IOTS (that models an IUT) is always assumed to be input-enabled.

To define an output test process that complements an input test process α, we have first to determine all the output sequences, valid and invalid, the output test process has to expect from IUT. The number of valid output sequences becomes infinite when the specification oscillates, in other words, when it has cycles that involve only outputs. Further, we always assume that the specification IOTS $Spec = <S, I \cup O, \lambda, s_0>$ does not possess this property. Thus, in response to α, the IOTS *Spec* can execute any trace that is a completed trace [Glab90] of the IOTS α][*Spec* leading into a terminal state, i.e., into state g, where $init(g) = \varnothing$. Let $ctraces(\alpha$][*Spec*) be the set of all such traces. It turns out that the set $ctraces(\alpha$][*Spec*) is closely related to the set of quiescent traces of the specification $qtraces(Spec)$, viz. it includes each quiescent trace β whose input projection, denoted $\beta_{\downarrow I}$, is the sequence α.

Proposition 4. $ctraces(\alpha$][*Spec*$) = \{\beta \in qtraces(Spec) \mid \beta_{\downarrow I} = \alpha\}$.

Thus, the set $ctraces(\alpha$][*Spec*$)_{\downarrow O} = \{\beta_{\downarrow O} \mid \beta \in qtraces(Spec)$ & $\beta_{\downarrow I} = \alpha\}$ contains all the output sequences that can be produced by the *Spec* in response to the input sequence α.

Let $qtraces(s)$ be the set of quiescent traces of *Spec* in state s. Given a quiescent trace $\beta \in qtraces(s)$, the sequence $\beta_{\downarrow I}\beta_{\downarrow O}\delta$ is said to be a *queued-quiescent trace* of *Spec* in state s, where $\delta \notin \Sigma$ is a designated symbol that denotes the absence of outputs, i.e., quiescence. We use $Qqtraces(s)$ to denote the set of queued-quiescence traces of s $\{(\beta_{\downarrow I}\beta_{\downarrow O}\delta) \mid \beta \in qtraces(s)\}$ and $Qqtraces_o(s, \alpha)$ to denote the set $\{\beta_{\downarrow O}\delta \mid \beta \in qtraces(s)$ & $\beta_{\downarrow I} = \alpha\}$. Next, we define the output test process itself.

Given the input test process α and the set $Qqtraces_o(s_0, \alpha)$, we define a set of output sequences $out(\alpha)$ the output test process can receive from an IUT. Intuitively, it is sufficient to consider all the shortest invalid output sequences along with all valid ones. Any valid sequence should not be followed by any further output action, as the specification becomes quiescent, while any premature quiescence indicates that the observed sequence is not a valid output sequence. The set $out(\alpha)$ is defined as follows. For each $\beta \in pref(Qqtraces_o(s_0, \alpha))$ the sequence $\beta \in out(\alpha)$ if $\beta \in Qqtraces_o(s_0, \alpha)$, otherwise $\beta a \in out(\alpha)$ for all $a \in O \cup \{\delta\}$ such that $\beta a \notin pref(Qqtraces_o(s_0, \alpha))$.

Definition 5. The *output test process* for the IOTS *Spec* and the input test process α is a tuple $<pref(out(\alpha)), O \cup \{\delta\}, \varnothing, \lambda_\alpha, \varepsilon>$, where $pref(out(\alpha))$ is the state set, $O \cup \{\delta\}$ is the input set, the output set is empty, $\lambda_\alpha = \{(\beta, a, \beta a) \mid \beta a \in pref(out(\alpha))\}$ and ε is the initial state. State $\beta \in pref(out(\alpha))$ is labeled with the verdict **pass** if $\beta \in Qqtraces_o(s_0, \alpha)$ or with the verdict **fail** if $\beta \in out(\alpha) \backslash Qqtraces_o(s_0, \alpha)$.

We reuse $out(\alpha)$ to denote the output test process that complements the input test process α. For a given input sequence $\alpha \in I^*$ the pair $(\alpha, out(\alpha))$ is called a *queued-quiescence tester* or *test case*.

To describe the way the output tester interacts with an IOTS $Imp \in IOTS(I, O)$ after the input test process α has terminated its execution against *Imp*, we denote $(\alpha$][*Imp*$)_{\downarrow O,\delta}$ the

IOTS that is obtained from (α][*Imp*) by first projecting it onto the output alphabet *O* and subsequent augmenting all the stable states of the resulting projection by self-looping transitions labeled with δ. In doing so, we treat the symbol δ as an input of the output test process, assuming that the tester synchronizing on δ just detects the fact that its buffer has no more symbols to read. Strictly speaking, treated as an output, a repeated δ violates the compatibility of a system in a stable state and a tester that reaches its terminal state. With this in mind, the correctness of the construction of the output test process (the soundness of the tester) can be stated as follows.

Proposition 6. For any *Imp* ∈ *IOTS*(*I*, *O*) if the IOTS (α][*Imp*)$\downarrow_{O,\delta}$][*out*(α) reaches a state where (α][*Imp*)$\downarrow_{O,\delta}$ and *out*(α) are incompatible, then the output tester *out*(α) is in a terminal state. For the *Spec* a state, where (α][*Spec*)$\downarrow_{O,\delta}$ and *out*(α) are incompatible is reached only after quiescence, while the output tester is in a terminal state labeled with the verdict **pass**.

Thus, the tester composed of two independent processes meets both requirements, namely, it is compatible in all the states, save for the terminal ones, with any IOTS, it has no cycles and never need choosing between input and output, thus the tester possesses the required properties.

The composition (α][*Imp*)$\downarrow_{O,\delta}$][*out*(α) of a queued-quiescence tester for a given specification with an implementation IOTS *Imp* over the same action set as the specification has one or several terminal states. In a particular test run, one of these states with the verdict **pass** or **fail** is reached. Considering the distribution of verdicts in the terminal states of the composition, the three following cases are possible:

Case 1. All the states have **fail**.

Case 2. States have **pass** as well as **fail**.

Case 3. All the states have **pass**.

These cases lead us to various relations between an implementation and the specification that can be established by the queued-quiescence testing.

In the first case, the implementation is distinguished from the specification in a single test run.

Definition 7. Given IOTS *Spec* and *Imp*, *Imp* is *queued-quiescence separable* from *Spec*, if there exists a test case (α, *out*(α)) for *Spec* such that the terminal states of the IOTS (α][*Imp*)$\downarrow_{O,\delta}$][*out*(α) are labeled with the verdict **fail**.

In the second case, the implementation can also be distinguished from the specification provided that a proper run is taken by the implementation during the test execution.

Definition 8. Given IOTS *Spec* and *Imp*, *Imp* is *queued-quiescence distinguishable* from *Spec*, if there exists a test case (α, *out*(α)) for *Spec* such that the terminal states of (α][*Imp*)$\downarrow_{O,\delta}$][*out*(α) are labeled with the verdicts **pass** and **fail**.

Consider now case 3, when for a given test case (α, *out*(α)) all the states have **pass**. In this case, the implementation does nothing illegal when the test case is executed, as it produces only valid output sequences. Two situations can yet be distinguished here. Either there exists a **pass** state of the output test process that is not included in any terminal state of

(α][*Imp*)$_{\downarrow O,\delta}$][*out*(α) or there is no such a state. The difference is that with the given test case in the former situation, the implementation could still be distinguished from its specification, while in the latter, it could not. This motivates the following definition.

Definition 9. Given IOTS *Spec* and *Imp*,

- *Imp* is said to be *queued-quiescence trace-included* in the *Spec* if for all $\alpha \in I^*$ no terminal state of the IOTS (α][*Imp*)$_{\downarrow O,\delta}$][*out*(α) is labeled with the verdict **fail**.
- *Imp* and *Spec* are *queued-quiescence trace-equivalent* if for all $\alpha \in I^*$ all the terminal states of the IOTS (α][*Imp*)$_{\downarrow O,\delta}$][*out*(α) include all the **pass** states of *out*(α) and only them.
- *Imp* that is queued-quiescence trace-included in the *Spec* but not queued-quiescence trace-equivalent to the *Spec* is said to be *queued-quiescence weakly-distinguishable* from *Spec*.

We characterize the above relations in terms of queued-quiescent traces.

Proposition 10. Given IOTS *Spec* with the initial state s_0 and *Imp* with the initial state t_0,

- *Imp* is queued-quiescence separable from *Spec* iff there exists an input sequence α such that $Qqtraces_o(t_0, \alpha) \cap Qqtraces_o(s_0, \alpha) = \varnothing$.
- *Imp* that is not queued-quiescence separable from *Spec* is queued-quiescence distinguishable from it iff there exists an input sequence α such that $Qqtraces_o(t_0, \alpha) \not\subseteq Qqtraces_o(s_0, \alpha)$.
- *Imp* that is not queued-quiescence distinguishable from *Spec* is queued-quiescence weakly-distinguishable from it iff there exists an input sequence α such that $Qqtraces_o(t_0, \alpha) \subset Qqtraces_o(s_0, \alpha)$.
- *Imp* is queued-quiescence trace-included into *Spec*, iff $Qqtraces(t_0) \subseteq Qqtraces(s_0)$.
- *Imp* and *Spec* are queued-quiescence trace-equivalent iff $Qqtraces(t_0) = Qqtraces(s_0)$.

Figure 1 provides an example of IOTS that are not quiescent trace equivalent, but are queued-quiescence trace-equivalent. Indeed, the quiescent trace $aa1\delta$ of the IOTS L_2 is not a trace of the IOTS L_1. In both, the input sequence *a* yields the queued-quiescent trace $a1\delta$, *aa* yields the queued-quiescent traces $aa1\delta$ and $aa2\delta$, any longer input sequence results in the same output sequences as *aa*.

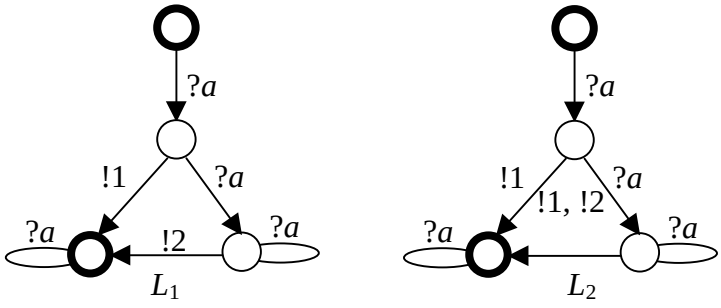


Figure 1: Two IOTS that have the different sets of quiescent traces, but are queued-quiescence trace-equivalent; inputs are decorated with "?", outputs with "!"; stable states are depicted in bold.

The IOTS L_1 and L_2 are considered indistinguishable in our framework, while according to the **ioco** relation [Tret96], they are distinguishable. The IOTS L_2 has the quiescent trace $aa1$ that is not a trace of L_1, therefore, to distinguish the two system, the tester has to apply two consecutive inputs a. The output 1 appearing only after the second input a indicates that the system being tested is, in fact, L_2 and not L_1. However, to make such a conclusion, the tester should be able to prevent the appearance of the output 1 after the first input a. Under our assumption, it is not possible. The tester interacts with the system via queues and has no way of knowing when the output is produced. The presence of a testing context that is a pair of finite queues in our case, makes implementation relations that could be tested via the context coarser, as is usually the case [PYBD96].

4. Deriving Queued-Quiescence Test Cases

Proposition 10 indicates the way test derivation could be performed for the IOTS *Spec* and an explicit fault model when we are given a finite set of implementations. Namely, for each *Imp* in the fault model, we may first attempt to determine an input sequence α such that $Qqtraces_o(t_0, \alpha) \cap Qqtraces_o(s_0, \alpha) = \varnothing$. If fail we could next try to find α such that $Qqtraces_o(t_0, \alpha) \not\subseteq Qqtraces_o(s_0, \alpha)$. If $Qqtraces_o(t_0, \alpha) \subseteq Qqtraces_o(s_0, \alpha)$ for each α the question is about an input sequence α such that $Qqtraces_o(t_0, \alpha) \neq Qqtraces_o(s_0, \alpha)$, thus $Qqtraces_o(t_0, \alpha) \subset Qqtraces_o(s_0, \alpha)$. Based on the found input sequence, a queued-quiescence test case for the *Imp* in hand can be constructed, as explained in the previous section. If no input sequence with this property can be determined we conclude that the IOTS *Spec* and *Imp* are queued-quiescence trace-equivalent, they cannot be distinguished by the queued-quiescence testing.

Search for an appropriate input sequence could be performed in a straightforward way by considering input sequences of increasing length. To do so, we just parameterize Definitions 7, 8 and 9 and accordingly Proposition 10 with the length of input sequences. Given a length of input sequences k, let $Qqtraces^k(s_0) = \{(\beta_{\downarrow I}\beta_{\downarrow O}\delta) \mid \beta \in qtraces(s_0) \ \& \ |\beta_{\downarrow I}| \leq k\}$. The set $Qqtraces^k(s_0)$ is finite for the IOTS L with a finite set of quiescence traces. Then, e.g., *Imp* and *Spec* are queued-quiescence k-trace-equivalent iff $Qqtraces^k(t_0) = Qqtraces^k(s_0)$. If length of α such that $Qqtraces_o(t_0, \alpha) \not\subseteq Qqtraces_o(s_0, \alpha)$ is k then *Imp* is said to be queued-quiescence k-distinguishable from *Spec*. With these parameterized definitions, we examine all the input sequences starting from an empty input action. The procedure terminates when the two IOTS are distinguished or when the value of k reaches a predefined maximum defined by the input buffer of the IUT available for queued testing.

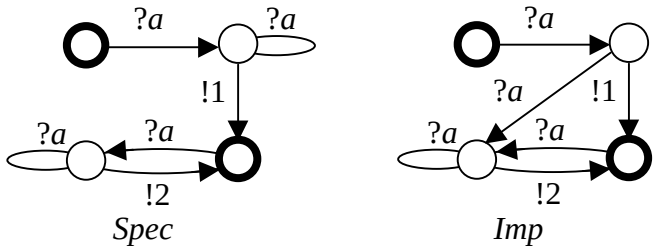
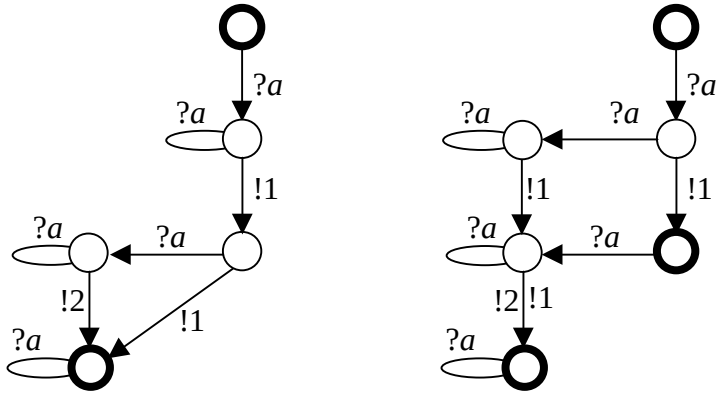


Figure 2: The IOTS that are queued-quiescence 2-distinguishable, but not queued-quiescence 1-distinguishable.

Consider the example in Figure 2. By direct inspection, one can assure *Imp* is not queued-quiescence 1-distinguishable from *Spec*, for both produce the output 1 in response to the

input *a*. However, it is queued-quiescence 2-distinguishable from *Spec*. Indeed, in response to the sequence ?*a*?*a* the *Spec* can produce the output 1 or 12. While the *Imp* - 2 or 12.
It is interesting to notice that the notion of *k*-distinguishability applied to the IOTS and FSM models exhibits different properties. In particular, two *k*-distinguishable FSM are also *k*+1-distinguishable. This does not always hold for IOTS. The system *Imp* in Figure 3 is queued-quiescence 1-distinguished from *Spec*; however, it is not queued-quiescence *k*-distinguished from *Spec* for any $k > 1$.



Spec *Imp*

Figure 3: The IOTS that are queued-quiescence 1-distinguishable, but not queued-quiescence *k*-distinguishable for $k > 1$.

This indicates that a special care has to be taken when one attempts to adapt FSM-based methods to the queued testing of IOTS.

There is at least one special case when distinguishability can be decided without determining the sets of queued-quiescence *k*-traces for various *k*.

Let $Imp_{\downarrow O}$ denote the set of output projections of all traces of *Imp*.

Proposition 11. Given two IOTS *Spec* and *Imp*, if the set $Imp_{\downarrow O}$ is not a subset of $Spec_{\downarrow O}$ then *Imp* is queued-quiescence distinguishable from *Spec*, moreover, any quiescence trace $\beta \in qtraces(Spec)$ such that $\beta_{\downarrow O} \in Imp_{\downarrow O} \backslash Spec_{\downarrow O}$ yields a queued-quiescence test case $(\beta_{\downarrow I}, out(\beta_{\downarrow I}))$ that when executed against the *Imp* produces the verdict **fail**.

The statement suggests a procedure for deriving test cases.

Procedure for deriving a test case that distinguishes the IOTS *Imp* from *Spec*.
Input: IOTS *Spec* and *Imp* such that the set $Imp_{\downarrow O}$ is not a subset of $Spec_{\downarrow O}$.
Output: A queued-quiescence test case $(\alpha, out(\alpha))$ such that $Qqtraces_o(t_0, \alpha) \not\subseteq Qqtraces_o(s_0, \alpha)$.

Step 1. By use of a subset construction, project *Imp* and *Spec* onto the output set *O*.

Step 2. Using the direct product of the obtained projections, determine a trace ρ that is a trace of the output projection of *Imp* while not being a trace of that of *Spec*.

Step 3. Compose LTS $<pref(\rho), O, \lambda_\rho, \varepsilon>$ with the *Imp* and obtain the LTS, where each trace is a trace of the *Imp* with the output projection ρ. Determine the input projection α of any trace of the obtained LTS and the queued-quiescence test case $(\alpha, out(\alpha))$.

Proposition 12. Given two IOTS *Spec* and *Imp*, let (α, *out*(α)) be the queued-quiescence test case derived by the above procedure. Then the queued-quiescence test case executed against the *Imp* produces the verdict **fail**. Moreover, if no **pass** verdict can be produced then *Imp* is queued-quiescence separable from *Spec*.

5. Queued-Suspension Testing of IOTS

In the previous sections, we explored the possibilities for distinguishing IOTS based on their queued-quiescent traces. The latter are pairs of input and output projections of quiescent traces. If systems with different quiescent traces have the same set of queued-quiescent traces, no queued-quiescence test case can differentiate them. However, sometimes such IOTS can still be distinguished by a queued testing, as we demonstrate below.

Consider the example in Figure 4. Here the two IOTS have different sets of quiescent traces, however, the have the same set of queued-quiescent traces {$a1\delta$, $aa1\delta$, $aa12\delta$, $aaa1\delta$, $aaa12\delta$, … }. In the testing framework presented in Section 3, they are not distinguishable. Indeed, we cannot tell them apart when a single input is applied to their initial states. Moreover, in response to the input sequence *aa* and to any longer sequence, they produce the same output sequence 12. The difference is that IOTS *Imp*, while producing the output sequence 12, becomes quiescent just before the output 2 and the IOTS *Spec* does not. The problem is that this quiescence is not visible through the output queue by the output test process that expects either 1 or 12 in response to *aa*. The queued-quiescence tester can detect the quiescence after reading the output sequence 12 as an empty queue, but it cannot detect an "intermediate" quiescence of the system. It has no way of knowing whether the system becomes quiescent before a subsequent input is applied. Both inputs are in the input buffer and it is completely up to the system when to read the second input.

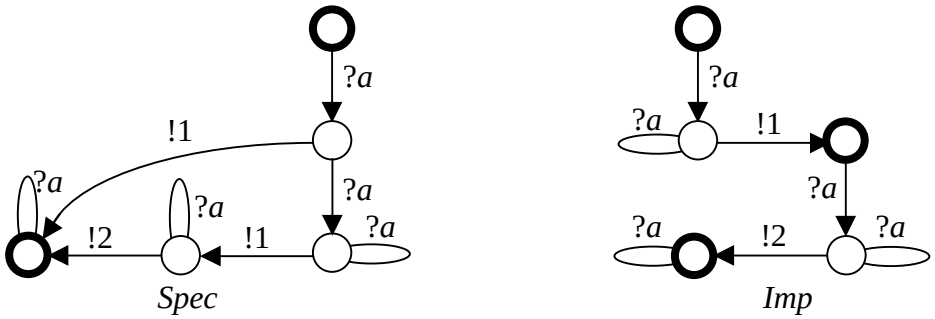


Figure 4: The queued-quiescence equivalent IOTS.

Intuitively, further decomposing the tester for the *Spec* into two input and two output test processes could solve the problem. In this case, testing is performed as follows. The first input test process issues the input *a*. The first output test process expects the output 1 followed by a quiescence δ, when the quiescence is detected, the control is transferred to the second input test process that does the final *a*. Then the second output test process expects quiescence. If, instead, it detects the output 2 it produces the verdict **fail** which indicates that the IUT is *Imp* and not *Spec*. Opposed to a queued-quiescence tester, such a tester can detect an intermediate quiescence of the system. The example motivates the following definitions.

Let $\alpha_1 \ldots \alpha_p$ be a finite sequence of input words such that $\alpha_i \in I^*$, $i = 1, \ldots, p$, and $\alpha_i \neq \varepsilon$ for $i \neq 1$. Each word α_i defines the input test process α_i (see Definition 3). To define output test processes that complement input test processes $\alpha_1 \ldots \alpha_p$, we first notice that the first

output test process is designed based on the fact that the *Spec* starts in the only state that is its initial state. This is no longer true for any subsequent output test process; the *Spec* can start in one of several stable states, depending on an output sequence it has produced in response to the stimuli from the previous input test processes. Let $\alpha\beta\delta \in Qqtraces(s_0)$, then we use *Spec*-**after**-(α, β) to denote the set of stable states that are reached by *Spec* when it executes all possible quiescent traces with the input projection α and output projection β. Consider the input test process α_2, before it starts, the *Spec* has produced one of $|Qqtraces_o(s_0, \alpha_1)|$ output sequences, each of which defines the set of stable states *Spec*-**after**-(α_1, β), where $\beta \in Qqtraces_o(s_0, \alpha_1)$, these are the starting states for the second input test process α_2. Thus, we have to define $|Qqtraces_o(s_0, \alpha_1)|$ output test processes that complement the input test process α_2. Each valid output sequence produced by the IUT so far is used to decide which pair of input and output test processes should execute next, while each invalid sequence terminates the test execution. Thus, testing becomes adaptive. For an output sequence $\beta \in Qqtraces_o(s_0, \alpha_1)$, the *Spec* can produce in response to α_2 any output sequence in the set $Qqtraces_o$(*Spec*-**after**-(α_1, β), α_2). Generalizing this to α_{i+1}, we have the following. The set of output sequences the *Spec* can produce in response to α_{i+1} is $Qqtraces_o$(*Spec*-**after**-($\alpha_1 \ldots \alpha_i$, $\beta_1 \ldots \beta_i$), α_{i+1}), where for the output projection $\beta_1 \ldots \beta_i$ it holds that $\beta_1 \in Qqtraces_o(s_0, \alpha_1)$ and $\beta_j \in Qqtraces_o$(*Spec*-**after**-($\alpha_1 \ldots \alpha_{j-1}$, $\beta_1 \ldots \beta_{j-1}$), α_j) for each $j = 2, \ldots, i$. A sequence of output words $\beta_1 \ldots \beta_i$ with such a property is said to be *consistent* (with the corresponding sequence of input words $\alpha_1 \ldots \alpha_i$).

Each output projection $\beta_1 \ldots \beta_i$ defines, therefore, a distinct test output process for the (i+1)-*th* input test process. The set of output sequences $out(\alpha_{i+1}, \beta_1 \ldots \beta_i)$ the output test process for the given sequence $\beta_1 \ldots \beta_i$ can receive from an IUT is defined as follows. For each $\gamma \in pref(Qqtraces_o$(*Spec*-**after**-($\alpha_1 \ldots \alpha_i$, $\beta_1 \ldots \beta_i$), $\alpha_{i+1})$ the sequence $\gamma \in out(\alpha_{i+1}, \beta_1 \ldots \beta_i)$ if $\gamma \in Qqtraces_o$(*Spec*-**after**-($\alpha_1 \ldots \alpha_i$, $\beta_1 \ldots \beta_i$), α_{i+1}), otherwise $\gamma a \in out(\alpha_{i+1}, \beta_1 \ldots \beta_i)$ for all $a \in O \cup \{\delta\}$ such that $\gamma a \notin pref(Qqtraces_o$(*Spec*-**after**-($\alpha_1 \ldots \alpha_i$, $\beta_1 \ldots \beta_i$), $\alpha_{i+1})$.

Let $\beta_1 \ldots \beta_i \cdot pref(out(\alpha_{i+1}, \beta_1 \ldots \beta_i))$ denote the set $\{\beta_1 \ldots \beta_i \alpha \mid \alpha \in pref(out(\alpha_{i+1}, \beta_1 \ldots \beta_i))\}$. Now we are ready to generalize the definition of output test processes (Definition 5), taking into account a valid output sequence produced by an IUT with the preceding test processes.

Definition 13. Let $\alpha_1 \ \ldots \ \alpha_{i+1}$ be a sequence of input words and $\beta_1 \ldots \beta_i$ be a consistent sequence of output words. An *output test process* for the IOTS *Spec*, sequence $\beta_1 \ldots \beta_i$ and the input test process α_{i+1} is a tuple $<\beta_1 \ldots \beta_i \cdot pref(out(\alpha_{i+1}, \beta_1 \ldots \beta_i)), O \cup \{\delta\}, \varnothing, \lambda_{\alpha i+1, \beta 1 \ldots \beta i}, \beta_1 \ldots \beta_i>$, where $\beta_1 \ldots \beta_i \cdot pref(out(\alpha_{i+1}, \beta_1 \ldots \beta_i))$ is the state set, $O \cup \{\delta\}$ is the input set, the output set is empty, $\lambda_{\alpha i+1, \beta 1 \ldots \beta i} = \{(\beta_1 \ldots \beta_i \gamma, a, \beta_1 \ldots \beta_i \gamma a) \mid \gamma a \in pref(out(\alpha_{i+1}, \beta_1 \ldots \beta_i))\}$ and $\beta_1 \ldots \beta_i$ is the initial state. Each state $\beta_1 \ldots \beta_i \gamma$ is labeled with the verdict **pass** if $\gamma \in Qqtraces_o$(*Spec*-**after**-($\alpha_1 \ldots \alpha_i$, $\beta_1 \ldots \beta_i$), α_{i+1}) or with the verdict **fail** if $\gamma \in out(\alpha_{i+1}, \beta_1 \ldots \beta_i) \backslash Qqtraces_o$(*Spec*-**after**-($\alpha_1 \ldots \alpha_i$, $\beta_1 \ldots \beta_i$), α_{i+1}).

We use $out(\alpha_{i+1}, \beta_1 \ldots \beta_i)$ to denote an output test process that complements the input test process α_{i+1} and the output sequence $\beta_1 \ldots \beta_i$. For a given sequence of input words $\alpha_1 \ldots \alpha_p$, the set of the tuples of pairs $(\alpha_1, out(\alpha_1)), \ldots, (\alpha_p, out(\alpha_p, \beta_1 \ldots \beta_{p-1}))$ for all consistent output sequences $\beta_1 \ldots \beta_{p-1}$ is called a *queued-suspension tester* or a *queued-suspension test case* and is denoted $(\alpha_1 \ldots \alpha_p, Out(\alpha_1 \ldots \alpha_p))$.

It is clear that for a single input test process, a queued-suspension tester reduces to a queued-quiescence tester. The queued-suspension testing is more discriminative than

queued-quiescence testing, as Figure 4 illustrates. In fact, consider a queued-quiescence tester derived from a single sequence $\alpha_1 \ldots \alpha_p$ and a queued-suspension tester derived from the sequence of p words $\alpha_1, \ldots, \alpha_p$, the former uses just the output projection of quiescent traces that have the input projection $\alpha_1 \ldots \alpha_p$ while the latter additionally partitions the quiescent traces into p quiescent sub-traces. Then the two systems that cannot be distinguished by the queued-suspension testing have to produce the same output projection, moreover, the output projections have to coincide up to the partition defined by the partition of the input sequence. This leads us to the notion of queued-suspension traces.

Given a finite sequence of finite input words $\alpha_1 \ldots \alpha_p$, a sequence of queued-quiescence traces $(\alpha_1\beta_1\delta)\ldots(\alpha_p\beta_p\delta)$ is called a *queued-suspension trace* of *Spec* if $\alpha_1\beta_1\delta \in Qqtraces(s_0)$ and for each $i = 2, \ldots, p$ it holds that $\beta_i\delta \in Qqtraces_o(Spec\text{-}\textbf{\textit{after}}\text{-}(\alpha_1 \ldots \alpha_{i-1}, \beta_1 \ldots \beta_{i-1}), \alpha_i)$. We use $Qstraces(s)$ to denote the set of queued-suspension traces of *Spec* in state s.

We define the relations that can be characterized with queued-suspension testing by adapting Definitions 7, 8, and 9.

Definition 14. Given IOTS *Spec* and *Imp*,

- *Imp* is *queued-suspension separable* from *Spec*, if there exist a test case $(\alpha_1 \ldots \alpha_p, Out(\alpha_1 \ldots \alpha_p))$ for *Spec* such that for any consistent output sequence $\beta_1 \ldots \beta_{p-1}$ the terminal states of the IOTS $(\alpha_p \,][\, Imp\ \textbf{after}(\alpha_1 \ldots \alpha_{p-1}, \beta_1 \ldots \beta_{p-1}))\downarrow_{O,\delta} \,][\, out(\alpha_p, \beta_1 \ldots \beta_{p-1})$ are labeled with the verdict **fail**.
- *Imp* is *queued-suspension distinguishable* from *Spec*, if there exist test case $(\alpha_1 \ldots \alpha_p, Out(\alpha_1 \ldots \alpha_p))$ for *Spec* and consistent output sequence $\beta_1 \ldots \beta_{p-1}$ such that the terminal states of the IOTS $(\alpha_p \,][\, Imp\ \textbf{after}(\alpha_1 \ldots \alpha_{p-1}, \beta_1 \ldots \beta_{p-1}))\downarrow_{O,\delta} \,][\, out(\alpha_p, \beta_1 \ldots \beta_{p-1})$ are labeled with the verdicts **pass** and **fail**.
- *Imp* is said to be *queued-suspension trace-included* in the *Spec* if for all $\alpha \in I^*$ and all possible partitions of α into words $\alpha_1, \ldots, \alpha_p$, no terminal state of IOTS $(\alpha_p \,][\, (\alpha_p \,][\, Imp\ \textbf{after}(\alpha_1 \ldots \alpha_{p-1}, \beta_1 \ldots \beta_{p-1}))\downarrow_{O,\delta} \,][\, out(\alpha_p, \beta_1 \ldots \beta_{p-1})$ is labeled with the verdict **fail**.
- *Imp* and *Spec* are *queued-suspension trace-equivalent* if for all $\alpha \in I^*$, all possible partitions of α into words $\alpha_1, \ldots, \alpha_p$, and all consistent output sequence $\beta_1 \ldots \beta_{p-1}$, all the terminal states of the IOTS $(\alpha_p \,][\, Imp\ \textbf{after}(\alpha_1 \ldots \alpha_{p-1}, \beta_1 \ldots \beta_{p-1}))\downarrow_{O,\delta} \,][\, out(\alpha_p, \beta_1 \ldots \beta_{p-1})$ include all the **pass** states of $out(\alpha_i)$ and only them.
- *Imp* that is queued-suspension trace-included in the *Spec* but not queued-suspension trace-equivalent to the *Spec* is said to be *queued-suspension weakly-distinguishable* from *Spec*.

Accordingly, the following is a generalization of Proposition 10.

Proposition 15. Given IOTS *Spec* and *Imp*,

- *Imp* is queued-suspension separable from *Spec* iff there exists a finite sequence of input words $\alpha_1 \ldots \alpha_i$ such that $Qstraces_o(Imp\text{-}\textbf{\textit{after}}\text{-}(\alpha_1 \ldots \alpha_{i-1}, \gamma_1 \ldots \gamma_{i-1}), \alpha_i) \cap Qstraces_o(Spec\text{-}\textbf{\textit{after}}\text{-}(\alpha_1 \ldots \alpha_{i-1}, \gamma_1 \ldots \gamma_{i-1}), \alpha_i) = \varnothing$ for any consistent $\gamma_1 \ldots \gamma_{i-1}$.
- *Imp* that is not queued-suspension separable from *Spec* is queued-suspension distinguishable from it iff there exist a finite sequence of input words $\alpha_1 \ldots \alpha_i$ and consistent $\gamma_1 \ldots \gamma_{i-1}$ such that $Qstraces_o(Imp\text{-}\textbf{\textit{after}}\text{-}(\alpha_1 \ldots \alpha_{i-1}, \gamma_1 \ldots \gamma_{i-1}), \alpha_i) \not\subseteq Qstraces_o(Spec\text{-}\textbf{\textit{after}}\text{-}(\alpha_1 \ldots \alpha_{i-1}, \gamma_1 \ldots \gamma_{i-1}), \alpha_i)$.

- *Imp* that is not queued-suspension distinguishable from *Spec* is queued-suspension weakly-distinguishable from it iff there exist a finite sequence of input words $\alpha_1 \ldots \alpha_i$ and consistent $\gamma_1 \ldots \gamma_{i-1}$ such that $Qstraces_o(Imp\text{-}\mathbf{after}\text{-}(\alpha_1 \ldots \alpha_{i-1}, \gamma_1 \ldots \gamma_{i-1}), \alpha_i) \subset Qstraces_o(Spec\text{-}\mathbf{after}\text{-}(\alpha_1 \ldots \alpha_{i-1}, \gamma_1 \ldots \gamma_{i-1}), \alpha_i)$.
- *Imp* is queued-suspension trace-included into *Spec*, iff $Qstraces(t_0) \subseteq Qstraces(s_0)$.
- *Imp* and *Spec* are queued-suspension trace-equivalent iff $Qstraces(t_0) = Qstraces(s_0)$.

The queued-suspension testing also needs input and output buffers as the queued-quiescence testing. The size of the input buffer is defined by the longest input word in a chosen test case $(\alpha_1 \ldots \alpha_p, Out(\alpha_1 \ldots \alpha_p))$, while that of the output buffer by the longest output sequence produced in response to any input word. We assume the size of the input buffer k is given and use it to define queued-suspension k-traces and accordingly, to parameterize Definition 14 obtaining appropriate notions of k-distinguishability. In particular, a queued-suspension trace of *Spec* $\alpha_1\beta_1\delta \ldots \alpha_p\beta_p\delta \in Qstraces(s_0)$ is called a queued-suspension k-trace of *Spec* if $|\alpha_i| \le k$ for all $i = 1, \ldots, p$. The set of all these traces $Qstraces^k(s_0)$ has a finite representation.

Definition 16. Let S_{stable} be the set of all stable states of an IOTS $Spec = \langle S, I \cup O, \lambda, s_0 \rangle$ and I^k denote the set of all words of at most k inputs. A *queued-suspension k-machine* for *Spec* is a tuple $\langle R, I^kO^*\delta, \lambda^k_{stable}, s_0 \rangle$, denoted $Spec^k_{susp}$, where the set of states $R \subseteq \mathbf{P}(S_{stable}) \cup \{s_0\}$, ($\mathbf{P}(S_{stable})$ is a powerset of S_{stable}), and the transition relation λ^k_{stable} are the smallest sets obtained by application of the following rules:

- $(r, \alpha\beta, r') \in \lambda^k_{stable}$ if $\alpha\beta \in I^kO^*\delta$ and r' is the union of sets s-**after**-(α, β) for all $s \in r$.
- In case the initial state s_0 is unstable $(s_0, \alpha\beta, r') \in \lambda^k_{stable}$ if $\alpha\beta \in I^kO^*\delta$, $\alpha = \varepsilon$ and $r' = s_0$-**after**-(ε, β).

Notice that each system that does not oscillate has at least one stable state.

Proposition 17. The set of traces of $Spec^k_{susp}$ coincides with the set of queued-suspension k-traces of *Spec*.

Corollary 18. *Imp* is queued-suspension k-distinguishable from *Spec* iff the Imp^k_{stable} has a trace that is not a trace of $Spec^k_{susp}$.

Figure 1 gives the example of IOTS that are queued-suspension trace equivalent, recall that they are also queued-quiescent trace-equivalent, but not quiescent trace equivalent.

We notice that a queued-suspension k-machine can be viewed as an FSM with the input set I^k and output set O^m for an appropriate integer m, so that FSM-based methods could be adapted to derive queued-suspension test cases.

6. Conclusion

We addressed the problem of testing from transition systems with inputs and outputs and elaborated a testing framework based on the idea of decomposing a tester into input and output processes. Input test process is applying inputs to a system under test via a finite input queue and output test process is reading outputs that the system puts into a finite output queue until it detects no more outputs from the system, i.e., the tester detects

quiescence in the system. In such a testing architecture, input from the tester and output from the system under test may occur simultaneously. We call such a testing scenario a queued testing. We analyzed two types of queued testers, the first consisting of single input and single output test processes, a so-called queued-quiescence tester, and the second consisting of several such pairs of processes, a so-called queued-suspension tester. We defined implementation relations that can be checked in the queued testing with both types of testers and proposed test derivation procedures.

Our work differs from the previous work in several important aspects. First of all, we make a liberal assumption on the way the tester interacts with a system under test, namely that the system can issue output at any time and the tester cannot determine exactly its stimulus after which an output occurs. We believe this assumption is less restrictive than any other assumption known in the testing literature [BrTr01], [Petr01]. Testing with this assumption requires buffers between the system and tester. These buffers are finite, opposed to the case of infinite queues considered in a previous work [VTKB92]. We demonstrated that in a queued testing, the implementation relations that can be verified are coarser than those previously considered. Appropriate implementation relations were defined and test derivation procedures were elaborated with a fault model in mind. Thus, the resulting test suite becomes finite and related to the assumptions about potential faults, opposed to the approach of [Tret96], where the number of test cases is, in fact, uncontrollable and not driven by any assumption about faults. The finiteness of test cases allows us, in addition, to check equivalence relations and not only preorder relations as in, e.g., [Tret96].

Concerning future work, we believe that this paper may trigger research in various directions. One possible extension could be to consider non-rigid transition systems, allowing non-observable actions. Procedures for test derivation proposed in this paper could be improved, as our purpose here was just to demonstrate that the new testing problem with finite queues could be solved in a straightforward way. It is also interesting to see to which extent one could adapt FSM-based test derivation methods driven by fault models, as it is done in [TaPe98] with a more restrictive assumption about a tester in mind.

Acknowledgment

This work was in part supported by the NSERC grant OGP0194381. The first author acknowledges fruitful discussions with Andreas Ulrich about testing IOTS. Comments of Jia Le Huo are appreciated.

References

[BoPe94] G. v. Bochmann and A. Petrenko, Protocol Testing: Review of Methods and Relevance for Software Testing, the proceedings of the ACM International Symposium on Software Testing and Analysis, ISSTA'94, USA, 1994.

[BrTr01] E. Brinksma and J. Tretmans, Testing Transition Systems: An Annotated Bibliography, LNCS Tutorials, LNCS 2067, Modeling and Verification of Parallel Processes, edited by F. Cassez, C. Jard, B. Rozoy and M. Ryan, 2001.

[dVBF02] R. G. de Vries, A. Belinfante and J. Feenstra, Automated Testing in Practice: The Highway Tolling System, the proceedings of the IFIP 14th International Conference on Testing of Communicating Systems, TestCom'2002, Berlin, Germany, 2002.

[Glab90] R. J. van Glabbeek, The Liear Time-Branching Time Spectrum, the proceedings of CONCUR'90, LNCS 458, 1990.

[JJTV99] C. Jard, T. Jéron, L. Tanguy and C. Viho, Remote Testing Can Be as Powerful as Local Testing, the proceedings of the IFIP Joint International Conference, Methods for Protocol Engineering and Distributed Systems, FORTE XII/PSTV XIX, China, 1999.

[LyTu89] N. Lynch and M. R. Tuttle, An Introduction to Input/Output Automata, CWI Quaterly, 2(3), 1989.

[Petr01] A. Petrenko, Fault Model-Driven Test Derivation from Finite State Models: Annotated Bibliography, LNCS Tutorials, LNCS 2067, Modeling and Verification of Parallel Processes, edited by F. Cassez, C. Jard, B. Rozoy and M. Ryan, 2001.

[Phal93] M. Phalippou, Executable Testers, the proceedings of the IFIP Sixth International Workshop on Protocol Test Systems, IWPTS'93, France, 1993.

[PYBD96] A. Petrenko, N. Yevtushenko, G. v. Bochmann, R. Dssouli, Testing in Context: Framework and Test Derivation, Computer Communications, 19, 1996.

[Sega93] R. Segala, Quiescence, Fairness, Testing and the Notion of Implementation, the proceedings of CONCUR'93, LNCS 715, 1993.

[TaPe98] Q. M. Tan, A. Petrenko, Test Generation for Specifications Modeled by Input/Output Automata, the proceedings of the 11th International Workshop on Testing of Communicating Systems, IWTCS'98, Russia, 1998.

[Tret96] J. Tretmans, Test Generation with Inputs, Outputs and Repetitive Quiescence, Software-Concepts and Tools, 17(3), 1996.

[Vaan91] F. Vaandrager, On the Relationship between Process Algebra and Input/Output Automata, the proceedings of Sixth Annual IEEE Symposium on Logic in Computer Science, 1991.

[VTKB92] L. Verhaard, J. Tretmans, P. Kim, and E. Brinksma, On Asynchronous Testing, the proceedings of the IFIP 5th International Workshop on Protocol Test Systems, IWPTS'92, Canada, 1992.