In: Proc. IFIP 15th Int. Conf. Testing of Communicating Systems, TestCom'2003, Sophia Antipolis, France (2003)

Testing Transition Systems with Input and Output Testers

Alexandre Petrenko[1], Nina Yevtushenko[2], Jia Le Huo[3]

[1] CRIM, Centre de recherche informatique de Montréal,
550 Sherbrooke West, Suite 100, Montreal, Quebec, H3A 1B9, Canada
Petrenko@crim.ca
[2] Tomsk State University, 36 Lenin Street, Tomsk, 634050, Russia
Yevtushenko@elefot.tsu.ru
[3] Department of Electrical and Computer Engineering, McGill University,
3480 University Street, Room 633, Montreal, Quebec, H3A 2A7, Canada
Jiale@macs.ece.mcgill.ca

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 an implementation under test (IUT) from producing outputs, while the IUT does not block inputs from the tester, either. Thus, input from the tester and output from the IUT may occur simultaneously and should be queued in finite buffers between the tester and the IUT. A framework for so-called queued-quiescence testing is developed, based on the idea that the tester should consist of two test processes, one applying inputs via a queue to an IUT and the other reading outputs from a queue until it detects no more outputs of the IUT, i.e., the tester detects quiescence of the IUT. The testing framework is then extended with so-called queued-suspension testing by considering a tester that has several pairs of input and output processes. Test derivation procedures are elaborated with a fault model in mind.

Keywords: conformance testing, test generation, input/output transition system, fault model

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 action of a system, in other words, that the system cannot accept the 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., [6] and [1]. A system, where the next input can arrive even before an output is produced in response to a previous input, is usually modeled by the input/output automaton model [5], 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., [2], [9], [10]. In this paper, we consider the IOTS model and take a close look on some basic assumptions underlying the existing IOTS testing frameworks.

An important publication on test generation from labeled transition systems (LTS) with inputs and outputs is [12]. In this paper, it is assumed that a tester interacting with an implementation under test (IUT) is an LTS. The LTS composition operator used to formalize this interaction does not distinguish between inputs and outputs, and the tester is not input-enabled. Due to the synchronous nature of the LTS composition, the tester preempts output of the IUT any time it decides to send input to the IUT. Although this allows the tester to avoid choosing between inputs and outputs, the tester overrides the principle that "output actions can never be blocked by the environment" [12, p.106]. An IOTS "generates output and internal actions autonomously" [5], so such an IUT can be synchronously composed only with a tester that is receptive to the IUT's output.

Another assumption about the tester is taken by Tan and Petrenko [11]. In this work, it is recognized that the tester cannot block the IUT's outputs. It is only assumed that the tester can detect the situation when it offers input to the IUT, but the latter, instead of consuming it, issues an output (a so-called "exception"). An exception halts a current test run (as the tester has lost control over the test execution) and results in the verdict **inconclusive**. Notice that the tester of [12] 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 an IUT because of a context, such as queues or interfaces, between them. As pointed out in [15], to apply the test derivation algorithm of [12], 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 [14]. The proposed approach relies on a specification of a given IOTS explicitly combined with a queue context, so it is not clear how this approach could be implemented in practice. This context is also considered in [4], where a stamping mechanism is proposed to order the outputs with respect to the inputs, while quiescence is ignored. A stamping process has to be synchronously composed with an IUT as the tester in [12].

We also notice that we are aware of the only work [11] that uses fault models in test derivation from IOTS. In [12] and [14], 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 findings in attempts to elaborate such an approach. In particular, we introduce a framework for testing IOTS, assuming that a tester can never prevent an IUT from producing outputs, while the IUT 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 the IUT.

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 [5]. Section 3 presents our framework for so-called queued-quiescence testing, based on the idea that the tester should consist of two test processes: one process applies inputs to an IUT via a finite input queue and the other reads outputs that the IUT puts into a finite output queue until the second process detects no more outputs from the IUT, i.e., the tester detects quiescence of the IUT. 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 extend our testing framework with 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 the previous work and discussing further work. An earlier version of this paper is published in an INRIA preprint [7].

2 Preliminaries

A *labeled transition system* (LTS) is a 4-tuple $L = \langle S, \Sigma, \lambda, S_0 \rangle$, where S is a finite set of states with a non-empty set of initial states $S_0 \subseteq S$; Σ is a finite set of actions; $\lambda \subseteq S \times (\Sigma \cup \{\tau\}) \times S$ is a transition relation. The special symbol $\tau \notin \Sigma$ represents the internal action. We call an LTS *deterministic* if it contains no internal action, has a single initial state, and for transitions (s, a, s'), $(s, a, s'') \in \lambda$, $s' = s''$. (As opposed to the preprint [7], this paper considers LTS that might be non-deterministic.) After [12], we only consider strongly converging LTS, i.e., the LTS that contain no loop of internal actions.

Let $L_1 = \langle S, \Sigma_1, \lambda_1, S_0 \rangle$ and $L_2 = \langle T, \Sigma_2, \lambda_2, T_0 \rangle$, the *parallel composition* $L_1 \parallel L_2$ is the LTS $\langle R, \Sigma_1 \cup \Sigma_2, \lambda, R_0 \rangle$, where $R_0 = S_0 \times T_0$ is the set of initial states; the set of states $R \subseteq S \times T$ and the transition relation λ are the 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 $a \in \{\tau\} \cup \Sigma_1 \backslash \Sigma_2$, $(s, a, s') \in \lambda_1$, then $(st, a, s't) \in \lambda$;
- if $a \in \{\tau\} \cup \Sigma_2 \backslash \Sigma_1$, $(t, a, t') \in \lambda_2$, 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 it is completely up to the system when to produce an output, so that the environment cannot block the output. Formally, an *input/output transition system* (IOTS) L is an 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. We use $\langle S, I, O, \lambda, S_0 \rangle$ to represent an IOTS $\langle S, I \cup O, \lambda, S_0 \rangle$ with $I \cap O = \emptyset$. Further, we use $IOTS(I, O)$ to denote the set of all possible IOTS over the input set I and output set O.

Given state s of L, we further denote $init(s)$ the set of actions defined at s, i.e., $init(s) = \{a \in (\Sigma \cup \{\tau\}) \mid \exists s' \in S \text{ s.t. } ((s, a, s') \in \lambda)\}$. The IOTS is (strongly) *input-*

enabled if each input action is enabled at any state, i.e., $I \subseteq init(s)$ for each s. 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. We notice that IOTS here corresponds to IOLTS in [4], and input-enabled IOTS to IOA in [5]. State s of the IOTS is called *unstable* if $init(s) \cap (O \cup \{\tau\}) \neq \emptyset$. Otherwise, the state is *stable*. A non-empty sequence $\alpha \in \Sigma^*$ is called a *trace* of L in state s if there exist actions $a_1, \ldots, a_k$ in $\Sigma \cup \{\tau\}$ and states $s_1, \ldots, s_{k+1}$ such that $(s_i, a_i, s_{i+1}) \in \lambda$ for all $i = 1, \ldots, k$; $s_1 = s$; and the projection of $a_1 \ldots a_k$ onto the action set Σ is the sequence α. We use $traces(s)$ to denote the set of traces of L in state s, and $traces(P)$ to denote the union of traces of L in the states in P, where P is a set of states of *Spec*. Sometimes, we use L to refer to the set of initial states of the IOTS L, e.g., $traces(L)$ denotes the union of traces of L in its initial states. We call an IOTS L *oscillating* if there exist a state s reachable from an initial state and a sequence $o_1 o_2 \ldots o_k \in O^*$ such that $(o_1 o_2 \ldots o_k)^* \subseteq traces(s)$. Following [13] and [12], we refer to a trace that takes the IOTS from a given state to a stable state as a *quiescent* trace. We use $qtraces(P)$ to denote the set of quiescent traces of *Spec* in P.

When we compose two IOTS using the parallel composition of LTS, an output action enabled in one IOTS is blocked from happening by the other IOTS if the action is not enabled in the second IOTS. Such a situation, however, cannot be justified by our assumption about the IOTS model, i.e., outputs from an IOTS are under the control of the IOTS itself. On the other hand, the composition operator for IOA defined in [5], which does not have this problem, is only applicable to input-enabled IOTS. This discussion suggests that we need to define a composition operator for IOTS that are not necessarily input-enabled. To this end, we first state compatibility conditions that define when two IOTS can be composed by relaxing the original conditions of [5]. We use $L_1 \parallel L_2$ for IOTS L_1 and L_2 to denote the parallel composition of the LTS L_1 and L_2 when the difference between their inputs and outputs is neglected.

Definition 1. Let two IOTS $L_1 = \langle S, I_1, O_1, \lambda_1, S_0 \rangle$ and $L_2 = \langle T, I_2, O_2, \lambda_2, T_0 \rangle$ be such that the set $O_1 \cap O_2 = \emptyset$. Let *st* be a state of the composition $L_1 \parallel L_2$. The IOTS 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 $a \in I_1 \cap O_2$.

L_1 and L_2 are said to be *compatible* if they are compatible in each initial state in $S_0 \times T_0$. L_1 and L_2 are *fully compatible* if they are compatible in all the states of $L_1 \parallel L_2$.

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. We notice that the parallel composition $\parallel$ of any two IOTS that are not fully compatible violates the assumption that outputs of an IOTS cannot be blocked. Therefore, we define a parallel composition of IOTS only for fully compatible ones.

Definition 2. The *parallel composition*][of two fully compatible IOTS $L_1 \in IOTS(I_1, O_1)$, and $L_2 \in IOTS(I_2, O_2)$, where the sets $I_1 \cap I_2$ and $O_1 \cap O_2$ are empty, is an IOTS defined as L_1][$L_2 = L_1 \parallel L_2$, with inputs $(I_1 \cup I_2) \setminus (O_1 \cup O_2)$ and outputs $O_1 \cup O_2$.

For fully compatible IOTS, the results of both operators, || and][, coincide. For the IOTS that are not fully compatible, the composition][is not defined.

3 Framework for Queued-Quiescence Testing

In defining a framework for testing systems modeled by IOTS, we first assume that testers are modeled by IOTS. We then require that any tester possess the following properties in addition to the usual soundness requirement. First, due to our assumption about the IOTS model, a tester should not preempt output of any IOTS. Second, a tester should always reach a verdict in finite steps, and once a verdict is reached, the tester should not change it later in the same test run. Third, a tester should be deterministic, meaning that it should have no internal actions and at most a single output action is enabled in any state. Finally, a tester should not make choice between inputs and outputs.

In a typical testing framework, it is usually assumed that a tester is a single process applying inputs to an IUT and observing outputs from the IUT. The two systems, the tester and the IUT, form a closed system. This means that if L_1 is an IOTS modeling a tester, while L_2 is an IOTS modeling the IUT, then $I_1 = O_2$, and $I_2 = O_1$. To be fully compatible with all the IOTS in $IOTS(I_2, O_2)$ the tester should be input-enabled. However, input-enabledness of testers, while making them meet the first requirement, may cause violation of the remaining ones.

An input-enabled tester may yield an infinite test run because the IOTS modeling the tester includes cycles. The test execution may never terminate when the tester interacts with an IUT with proper cycles. This, however, could simply be resolved by defining a tester whose only cycles are self-loops in the states labeled with verdicts. An IUT may continuously interact with such a tester, but the tester still reaches a verdict in a finite number of steps and remains in a state with the reached verdict. However, an arbitrary IUT may produce a wrong output after the tester has reached the verdict **pass**, which cannot be reversed because of the self-loops. To solve this problem, we require that states with the verdict **pass** only be reached when the quiescence of an IUT is detected. This feature of the tester immediately excludes oscillating specifications from further consideration, but still leaves us a wide class of specifications. Thus, we will define testers with the above stated features.

Another problem of input-enabled testers is that such a tester needs choosing between inputs and outputs. In fact, in any state where the tester has to produce an output to an IUT, all the inputs are enabled as well. So the tester has to choose between doing input or output, violating the last requirement.

It turns out that a tester processing inputs separately from outputs may resolve the problem. 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 that the specification can produce in response to the submitted input

sequence should take the output test process into a state labeled with the verdict **pass**, while any other output sequence produced by an IUT should take the output test process to a 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 $pref(\alpha)$ denote the set of all the prefixes of a sequence $\alpha \in \Sigma^*$ over the set Σ. The set $pref(\alpha)$ has the empty sequence ε. Also given a set $\Gamma \subseteq \Sigma^*$, let $\{\beta \in pref(\gamma) \mid \gamma \in \Gamma\} = pref(\Gamma)$.

Definition 3. Given an input word $\alpha \in I^*$, the *input test process* with α for $L \in$ IOTS(I, O) is an (deterministic) IOTS $\alpha = \langle pref(\alpha), \emptyset, I, \lambda_\alpha, \{\varepsilon\}\rangle$, where the state set is $pref(\alpha)$ with ε as the only initial state, the set of inputs is empty, the set of outputs is I, and the transition relation $\lambda_\alpha = \{(\beta, a, \beta a) \mid \beta a \in pref(\alpha)\}$.

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 in IOTS(I, O) that is input-enabled.

To define an output test process that complements the input test process α, we have first to determine all the output sequences, valid and invalid, the output test process has to expect from the IUT. The number of valid output sequences is finite, as the specification does not oscillate by our assumption. Thus, in response to α, the IOTS *Spec* $\in$ IOTS(I, O) can execute any trace that is a completed trace [3] of the IOTS α][*Spec* leading into a terminal state, i.e., into state g, where $init(g) = \emptyset$. 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 *Spec* in response to the input sequence α.

Given a quiescent trace $\beta \in qtraces(P)$, where P is a set of states of *Spec*, the sequence $\beta_{\downarrow I}\beta_{\downarrow O}\delta$ is said to be a *queued-quiescent trace* of *Spec* in P, where $\delta \notin \Sigma$ is a designated symbol indicating that no more outputs follows, in other words, that *Spec* becomes quiescent as it has reached a stable state. We use $Qqtraces(P)$ to denote the set of queued-quiescent traces of P $\{(\beta_{\downarrow I}\beta_{\downarrow O}\delta) \mid \beta \in qtraces(P)\}$ and $Qqtraces_o(P, \alpha)$ to denote the set $\{\beta_{\downarrow O}\delta \mid \beta \in qtraces(P)$ & $\beta_{\downarrow I} = \alpha\}$. Next, we define the output test process and the test case.

Given the input test process α and the set $Qqtraces_o(Spec, \alpha)$, we define a set of output sequences $out(\alpha)$ that the output test process can receive from an IUT. It is sufficient to consider all the shortest invalid output sequences along with all the 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 valid. The set $out(\alpha)$ is defined as follows. For each $\beta \in pref(Qqtraces_o(Spec, \alpha))$ the sequence $\beta \in out(\alpha)$ if $\beta \in Qqtraces_o(Spec, \alpha)$, otherwise $\beta a \in out(\alpha)$ for all $a \in O \cup \{\delta\}$ such that $\beta a \notin pref(Qqtraces_o(Spec, \alpha))$.

Definition 5. The *output test process* for the IOTS *Spec* and the input test process α is an (deterministic) IOTS $<pref(out(\alpha)), O \cup \{\delta\}, \varnothing, \lambda_{out(\alpha)}, \{\varepsilon\}>$, where certain states are labeled with verdicts **pass** or **fail**. and the state set is $pref(out(\alpha))$ with ε as the only initial state, the input set is $O \cup \{\delta\}$, and the output set is empty. State $\beta \in pref(out(\alpha))$ is labeled with the verdict **pass** if $\beta \in Qqtraces_o(Spec, \alpha)$ or with the verdict **fail** if $\beta \in out(\alpha) \backslash Qqtraces_o(Spec, \alpha)$. The transition relation $\lambda_{out(\alpha)} = \{(\beta, a, \beta a) \mid \beta a \in pref(out(\alpha))\} \cup \{(\beta, \delta, \beta) \mid \beta$ is labeled **pass**$\} \cup \{(\beta, a, \beta) \mid a \in O \cup \{\delta\}$ & β is labeled **fail**$\}$.

For a given input test process α, where $\alpha \in I^*$, we reuse $out(\alpha)$ to denote the output test process that complements the input test process α. The pair $(\alpha, out(\alpha))$ is called a *queued-quiescence tester* or simply a *test case* for the IOTS *Spec*.

The self-looping transitions at the states labeled **pass** and **fail** are added to make the output test process fully compatible with any IUT in the set IOTS(*I*, *O*). These self-loops are the only cycles of the output test process, so verdicts **pass** or **fail** can be reached in finite steps. Once verdicts are reached, they are not changed. Therefore, the states with verdicts indicate the end of the test execution. We assume that once the output test process detects the quiescence, the IUT cannot produce any visible output later, which justifies why the **pass** states have only a self-loop on quiescence.

To describe the execution of a queued-quiescence test case, we define a new operator $\delta\Downarrow O$. For IOTS *L*, $L_{\delta\Downarrow O}$ is an IOTS obtained by first augmenting all the stable states of *L* by self-looping transitions labeled with δ, then projecting the augmented automaton onto the alphabet $O \cup \delta$, and finally determinizing the obtained automaton. The execution of a queued-quiescence test case $(\alpha, out(\alpha))$ against an IOTS $Imp \in IOTS(I, O)$ is described by the IOTS $(\alpha \,][\, Imp)_{\delta\Downarrow O} \,][\, out(\alpha)$. Each trace leading this IOTS into a state, where the output test process is in a state labeled with **pass** or **fail**, is a *test run*. Notice that we treat the symbol δ as an input of the output test process, assuming that the tester executing δ just detects the fact that its buffer has no more symbols to read. Since the outputs of an IUT are stored in a finite queue, any implementation that, in response to the input sequence α, can produce an output sequence longer than the queue length may overflow the queue. To solve this problem, we should determine a lower bound of the output queue length so that the buffer is not overflowing until the tester reaches a verdict. The bound depends on the input sequence α and *Spec*, and it is finite because *Spec* does not oscillate.

The queued-quiescence tester $(\alpha, out(\alpha))$ meets all the requirements stated above. We use the term *verdict* state to refer to a state of the IOTS $(\alpha \,][\, Imp)_{\delta\Downarrow O} \,][\, out(\alpha)$ such that the IOTS $out(\alpha)$ is in a state with a verdict.

Proposition 6. For a queued-quiescence test case $(\alpha, out(\alpha))$ of *Spec* and any IOTS $Imp \in IOTS(I, O)$

- the IOTS $(\alpha \,][\, Imp)_{\delta\Downarrow O}$ and $out(\alpha)$ are fully compatible;
- at least one verdict state is reachable from every state in the IOTS $(\alpha \,][\, Imp)_{\delta\Downarrow O} \,][\, out(\alpha)$ and every cycle in the IOTS involves only verdict states, in other words, the tester always reaches a verdict in finite steps;
- both α and $out(\alpha)$ are deterministic;
- there is no state in α or $out(\alpha)$ where both inputs and outputs are enabled;

- if a verdict state is reached in the IOTS $(\alpha][\ Spec)_{\delta\Downarrow O}][\ out(\alpha)$, the output tester $out(\alpha)$ is in a state with the verdict **pass**, i.e., the test case $(\alpha, out(\alpha))$ is sound.

The composition $(\alpha][\ Imp)_{\delta\Downarrow O}][\ out(\alpha)$ has one or several verdict 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 verdict states of the composition, three 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 $(\alpha, out(\alpha))$ for *Spec* such that all the verdict states of the IOTS $(\alpha][\ Imp)_{\delta\Downarrow O}][\ out(\alpha)$ are labeled with the verdict **fail**.

In the second case, the implementation can also be distinguished from the specification if 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 $(\alpha, out(\alpha))$ for *Spec* such that at least one verdict state of $(\alpha][\ Imp)_{\delta\Downarrow O}][\ out(\alpha)$ is labeled with the verdict **fail**.

Clearly, *Imp* queued-quiescence separable from *Spec* is also queued-quiescence distinguishable from it. Consider now case 3, when for a given test case $(\alpha, out(\alpha))$ 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 verdict state of $(\alpha][\ Imp)_{\delta\Downarrow O}][\ out(\alpha)$ 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 *queued-quiescence weakly-distinguishable* from *Spec* if there exists a test case $(\alpha, out(\alpha))$ for *Spec* such that the verdict states of $(\alpha][\ Imp)_{\delta\Downarrow O}][\ out(\alpha)$ does not include all the **pass** states of $out(\alpha)$.
- *Imp* is *queued-quiescent trace-included* in *Spec* if for all $\alpha \in I^*$ all the verdict state of the IOTS $(\alpha][\ Imp)_{\delta\Downarrow O}][\ out(\alpha)$ are labeled with the verdict **pass**.
- *Imp* and *Spec* are *queued-quiescent trace-equivalent* if for all $\alpha \in I^*$ all the verdict states of the IOTS $(\alpha][\ Imp)_{\delta\Downarrow O}][\ out(\alpha)$ are labeled with the verdict **pass** and include all the **pass** states of $out(\alpha)$.

By definition, *Imp* is queued-quiescence weakly-distinguishable from *Spec* if *Imp* is queued-quiescence distinguishable from *Spec*, or if *Imp* is queued-quiescent trace-

included in but not queued-quiescent trace-equivalent to *Spec*. We characterize the above relations in terms of traces and queued-quiescent traces.

Proposition 10. Given IOTS *Spec* and *Imp*,

1. *Imp* is queued-quiescence separable from *Spec* iff there exists an input sequence α such that $Qqtraces_o(Imp, \alpha) \cap Qqtraces_o(Spec, \alpha) = \varnothing$.
2. *Imp* is queued-quiescence distinguishable from it iff there exists an input sequence α such that $traces(\alpha][\ Imp)_{\delta\Downarrow O} \not\subseteq traces(\alpha][\ Spec)_{\delta\Downarrow O}$.
3. *Imp* is queued-quiescence weakly-distinguishable from it iff there exists an input sequence α such that $traces(\alpha][\ Imp)_{\delta\Downarrow O} \neq traces(\alpha][\ Spec)_{\delta\Downarrow O}$.
4. *Imp* is queued-quiescent trace-included into *Spec*, iff *Imp* does not oscillate and $Qqtraces(Imp) \subseteq Qqtraces(Spec)$.
5. *Imp* and *Spec* are queued-quiescent trace-equivalent iff *Imp* does not oscillate and $Qqtraces(Imp) = Qqtraces(Spec)$.

$traces(\alpha][\ Imp)_{\delta\Downarrow O}$ is the set of traces of $(\alpha][\ Imp)_{\delta\Downarrow O}$ in the initial states of the IOTS. $traces(\alpha][\ Imp)_{\delta\Downarrow O} = traces(\alpha][\ Imp)_{\downarrow O} \cup Qqtraces_o(Imp, \alpha)\delta^*$. The relation between $traces(\alpha][\ Imp)_{\downarrow O}$ and $traces(\alpha][\ Spec)_{\downarrow O}$ is used to deal correctly with oscillating implementations [10]. Fig. 1 provides an example of the systems that are not quiescent trace equivalent, but are queued-quiescent 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.

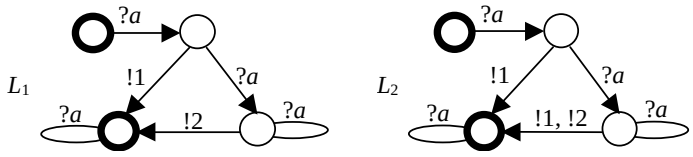

Fig. 1. Two IOTS that have different sets of quiescent traces, but are queued-quiescent trace-equivalent. Inputs are decorated with "?", outputs with "!"; and stable states are drawn in bold

The IOTS L_1 and L_2 are considered indistinguishable in our framework, while according to the **ioco** relation [12], 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 systems, the tester has to apply two consecutive inputs a. The output 1 appearing after the second input a indicates that the system being tested is, in fact, L_2 and not L_1. However, to reach such a conclusion, the tester should be able to prevent the appearance of the output 1 after the first input a. This is not possible under our assumption that the tester cannot block outputs of the IUT. The tester interacts with the IUT via queues and has no way of knowing in which state the output is actually produced. The presence of a testing context, which 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 [8].

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 that includes 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 $Qqtraces_o(Imp, \alpha) \cap Qqtraces_o(Spec, \alpha) = \emptyset$. If fail we could next try to find α such that $traces(\alpha][\ Imp)_{\delta\Downarrow O} \not\subseteq traces(\alpha][\ Spec)_{\delta\Downarrow O}$. If $traces(\alpha][\ Imp)_{\delta\Downarrow O} \subseteq traces(\alpha][\ Spec)_{\delta\Downarrow O}$ for each α the question is about an input sequence α such that $traces(\alpha][\ Imp)_{\delta\Downarrow O} \neq traces(\alpha][\ Spec)_{\delta\Downarrow O}$, thus $traces(\alpha][\ Imp)_{\delta\Downarrow O} \subset traces(\alpha][\ Spec)_{\delta\Downarrow O}$. Based on the found input sequence, a queued-quiescence test case for *Imp* at 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-quiescent trace-equivalent, they cannot be distinguished by the queued-quiescence testing.

Search for an appropriate distinguishing 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*, then, e.g., *Imp* and *Spec* are queued-quiescent *k*-trace-equivalent iff $traces(\alpha][\ Imp)_{\delta\Downarrow O} = traces(\alpha][\ Spec)_{\delta\Downarrow O}$ for all $\alpha \in I^{\leq k}$, where $I^{\leq k}$ denotes the set of all input sequences of length equal or less than *k*. If the length of α such that $traces(\alpha][\ Imp)_{\delta\Downarrow O} \not\subseteq traces(\alpha][\ Spec)_{\delta\Downarrow O}$ 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 sequence. 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.

Consider the example in Fig. 2. *Imp* is not queued-quiescence 1-distinguishable from *Spec*, for both produce the output 1 in response to the input *a*. However, *Imp* is queued-quiescence 2-distinguishable from *Spec*. Indeed, in response to the sequence *aa* the *Spec* can produce the output 1 or 12. While *Imp* - 2 or 12.

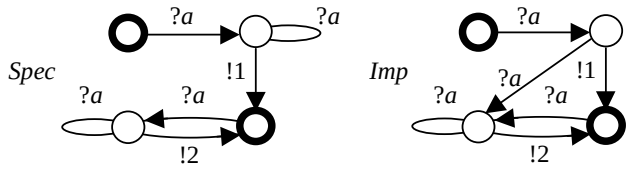

Fig. 2. The IOTS that are queued-quiescence 2-distinguishable, but not queued-quiescence 1-distinguishable

The search for a distinguishing input sequence relies on a procedure that verifies whether a given input sequence α satisfies $traces(\alpha][\ Imp)_{\delta\Downarrow O} \subseteq traces(\alpha][\ Spec)_{\delta\Downarrow O}$. Instead of elaborating this procedure, we give a more general procedure that accepts a regular language defined over the input set. Let *E* denote such a language, $E \subseteq I^*$, following the definition of the input test process, we also use *E* to denote the

(deterministic) IOTS whose trace set is *pref*(*E*). Since *E* is regular, such an IOTS exists. In the following proposition, we generalize item 2 in Proposition 10 by considering the inclusion relation between $traces(E \,][\, Imp)_{\delta\Downarrow O}$ and $traces(E \,][\, Spec)_{\delta\Downarrow O}$.

Proposition 11. Given two IOTS *Spec* and *Imp*, *Imp* is queued-quiescence distinguishable from *Spec* iff there exists a regular language $E \subseteq I^*$ such that $traces(E \,][\, Imp)_{\delta\Downarrow O} \not\subseteq traces(E \,][\, Spec)_{\delta\Downarrow O}$. Moreover, any trace $\beta \in traces(E \,][\, Imp)$ such that $\beta_{\downarrow O}\delta^* \cap (traces(E \,][\, Imp)_{\delta\Downarrow O} \setminus traces(E \,][\, Spec)_{\delta\Downarrow O}) \neq \emptyset$ yields a queued-quiescence test case $(\beta_{\downarrow I}, out(\beta_{\downarrow I}))$ that, when executed against *Imp*, produces the verdict **fail**.

Proof: (If) If there exists a regular language $E \subseteq I^*$ such that $traces(E \,][\, Imp)_{\delta\Downarrow O} \not\subseteq traces(E \,][\, Spec)_{\delta\Downarrow O}$, the part after "moreover" in the proposition indicates how a corresponding test case can be derived.

(Only if) If *Imp* is queued-quiescence distinguishable from *Spec*, according to the definition of the queued-quiescence distinguishability, there exists an input sequence α, which is a regular language with a single word, such that $traces(\alpha \,][\, Imp)_{\delta\Downarrow O} \not\subseteq traces(\alpha \,][\, Spec)_{\delta\Downarrow O}$. QED.

We call a language *E* satisfying the properties in Proposition 11 a *distinguishing input set* of *Spec* and *Imp*. The proposition suggests a test case derivation procedure.

Procedure 12. For deriving a test case of *Spec* that *Imp* fails if a given language *E* is a distinguishing input set.
Input: IOTS *Spec* and *Imp*, and a regular language *E*.
Output: *E* is not a distinguishing input set or a test case $(\alpha, out(\alpha))$.

Step 1. Construct the deterministic automata that accept $traces(E \,][\, Imp)_{\delta\Downarrow O}$ and $traces(E \,][\, Spec)_{\delta\Downarrow O}$, respectively.

Step 2. Using the direct products of the obtained automata, determine a sequence $\rho \in traces(E \,][\, Imp)_{\delta\Downarrow O} \setminus traces(E \,][\, Spec)_{\delta\Downarrow O}$. If such a sequence exists, go to Step 3; otherwise, return the result that *E* is not a distinguishing input set.

Step 3. Construct a deterministic automaton by composing *Imp* with the LTS $<pref(\rho_{\downarrow O}), O, \lambda_\rho, \{\varepsilon\}>$, where $\lambda_\rho = \{(\beta, a, \beta a) \mid \beta a \in pref(\rho_{\downarrow O})\}$. Determine a trace γ of the obtained LTS with $\gamma_{\downarrow I} \in E$ and $\gamma_{\downarrow O} = \rho_{\downarrow O}$ and the queued-quiescence test case $(\gamma_{\downarrow I}, out(\gamma_{\downarrow I}))$.

Proposition 13. Given two IOTS *Spec* and *Imp*, and a distinguishing input set *E*, let $(\alpha, out(\alpha))$ be the queued-quiescence test case derived by the above procedure. Then the queued-quiescence test case executed against *Imp* produces the verdict **fail**.

If we consider every $E \in \{\{\alpha\} \mid \alpha \in I^{\leq k}\}$, the test cases that queued-quiescence *k*-distinguish *Imp* from *Spec* are derived. We notice that if the set *E* is I^*, Procedure 12 reduces to the test case derivation procedure reported earlier [7].

It is interesting to know 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 Fig. 3 is queued-quiescence 1-distinguished from *Spec*; however, it is not queued-quiescence *k*-distinguished from *Spec* for any $k > 1$. This indicates that a

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

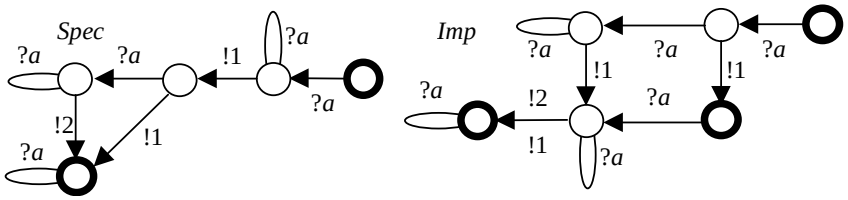

Fig. 3. The IOTS that are queued-quiescence 1-distinguishable, but not queued-quiescence *k*-distinguishable for $k > 1$

5 Queued-Suspension Testing

In the previous sections, we explored the possibilities for distinguishing IOTS based on their traces and queued-quiescent traces. The latter are pairs of input and output projections of quiescent traces. If two non-oscillating systems with different quiescent traces have the same sets of queued-quiescent traces, queued-quiescence testing may not differentiate them. However, sometimes such IOTS can still be distinguished by a queued testing, as we demonstrate below.

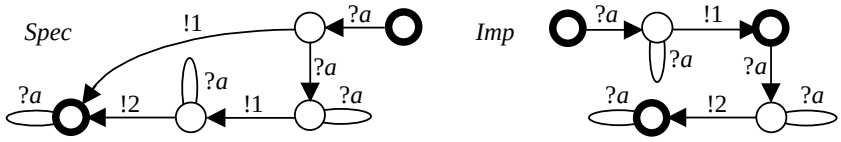

Fig. 4. Two queued-quiescent trace equivalent IOTS

Consider the example in Fig. 4. Here the two IOTS have different sets of quiescent traces, however, they have the same set of queued-quiescent traces $\{a1\delta, aa1\delta, aa12\delta, aaa1\delta, aaa12\delta, \ldots\}$. 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.

Further decomposing the tester for *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*. As opposed to a queued-quiescence tester, such a tester can detect intermediate quiescence of the IUT.

The example motivates the definition of a new type of testers. Such a tester is defined for a given sequence of input words $\alpha_1 \ldots \alpha_p$, in which $\alpha_i \neq \varepsilon$ for $i = 2, \ldots, p$. The tester is a finite tree with queued-quiescence test cases as nodes connected by transfer of control. The root node $(\alpha_1, out(\alpha_1))$ is a queued-quiescence test case of *Spec*, and is executed first. If the IUT passes the queued-quiescence test case, one of the node's children is selected based on the output of the IUT $\beta_1 \delta \in Qqtraces_o(Spec, \alpha_1)$ and control is transferred to this node; otherwise, the IUT fails the tester and the test execution is terminated. 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. If we also use *Spec*-**after**-(α, β) to denote the IOTS obtained from *Spec* by initializing it in these states, the selected child node is a queued-quiescence test case of *Spec*-**after**-(α_1, β_1). The input test process of the child node executes α_2. The process continues until the IUT fails or a verdict of a leaf node is reached.

We define a sequence of output words $\beta_1 \ldots \beta_i$ to be *consistent* with the corresponding sequence of input words $\alpha_1 \ldots \alpha_i$ if $\beta_1 \in Qqtraces_o(Spec, \alpha_1)$ and $\beta_j \in Qqtraces_o(Spec\text{-}\mathbf{after}\text{-}(\alpha_1 \ldots \alpha_{j-1}, \beta_1 \ldots \beta_{j-1}), \alpha_j)$ for each $j = 2, \ldots, i$. Every node in the tree is identified by a consistent output sequence that leads the tester to the node. Given α_i, we use $(\alpha_i, out(\alpha_i, \beta_1 \ldots \beta_{i-1}))$ to denote the queued-quiescence test case of *Spec*-**after**-$(\alpha_1 \ldots \alpha_{i-1}, \beta_1 \ldots \beta_{i-1})$. We have the definition of the tester based on the discussions above.

Definition 14. Given a finite sequence of input words $\alpha_1 \ldots \alpha_p$, a *queued-suspension tester* or a *queued-suspension test case* $(\alpha_1 \ldots \alpha_p, Out(\alpha_1 \ldots \alpha_p))$ is a tree $(N, P, (\alpha_1, out(\alpha_1)))$, in which

- N is the set of nodes, $N = \{(\alpha_1, out(\alpha_1))\} \cup \{(\alpha_i, out(\alpha_i, \beta_1 \ldots \beta_{i-1})) \mid \beta_1 \ldots \beta_{i-1}$ is an output sequence consistent with $\alpha_1 \ldots \alpha_{i-1}, i = 2, \ldots, p\}$;
- P is the transition relation, $P = \{(\alpha_1, out(\alpha_1)) \rightarrow (\alpha_2, out(\alpha_2, \beta_1)) \mid \beta_1$ is an output sequence consistent with $\alpha_1\} \cup \{(\alpha_i, out(\alpha_i, \beta_1 \ldots \beta_{i-1})) \rightarrow (\alpha_{i+1}, out(\alpha_{i+1}, \beta_1 \ldots \beta_i)) \mid \beta_1 \ldots \beta_i$ is an output sequence consistent with $\alpha_1 \ldots \alpha_i, i = 2, \ldots, p\text{-}1\}$;
- $(\alpha_1, out(\alpha_1))$ is the root node.

It is clear that for a single input word, a queued-suspension tester reduces to a queued-quiescence tester. The queued-suspension testing is more discriminative than queued-quiescence testing, as Fig. 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 $(\alpha_1\beta_1\delta)\ldots(\alpha_p\beta_p\delta)$ is called a *queued-suspension trace* of *Spec* if $\beta_1 \ldots \beta_p$ is an output sequence consistent with $\alpha_1 \ldots \alpha_p$. We use *Qstraces*(*Spec*) to denote the set of queued-suspension traces of *Spec* in the initial states.

We define the relations that can be established by queued-suspension testing similar to Definitions 7, 8, and 9.

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

- *Imp* is *queued-suspension separable* from *Spec*, if there exists a test case $(\alpha_1 \ldots \alpha_p, Out(\alpha_1 \ldots \alpha_p))$ for *Spec* such that for any consistent output sequence $\beta_1 \ldots \beta_{p-1}$ all the verdict states of the IOTS $(\alpha_p \,][\, Imp\text{-}\mathbf{after}\text{-}(\alpha_1 \ldots \alpha_{p-1}, \beta_1 \ldots \beta_{p-1}))_{\delta\Downarrow O} \,][\, out(\alpha_p, \beta_1 \ldots \beta_{p-1})$ are labeled with the verdict **fail**.
- *Imp* is *queued-suspension distinguishable* from *Spec*, if there exist a test case $(\alpha_1 \ldots \alpha_p, Out(\alpha_1 \ldots \alpha_p))$ for *Spec* and a consistent output sequence $\beta_1 \ldots \beta_{p-1}$ such that at least one verdict state of the IOTS $(\alpha_p \,][\, Imp\text{-}\mathbf{after}\text{-}(\alpha_1 \ldots \alpha_{p-1}, \beta_1 \ldots \beta_{p-1}))_{\delta\Downarrow O} \,][\, out(\alpha_p, \beta_1 \ldots \beta_{p-1})$ is labeled with the verdict **fail**.
- *Imp* is *queued-suspension weakly-distinguishable* from *Spec* if there exist a test case $(\alpha_1 \ldots \alpha_p, Out(\alpha_1 \ldots \alpha_p))$ for *Spec* and a consistent output sequence $\beta_1 \ldots \beta_{p-1}$ such that the verdict states of the IOTS $(\alpha_p \,][\, Imp\text{-}\mathbf{after}\text{-}(\alpha_1 \ldots \alpha_{p-1}, \beta_1 \ldots \beta_{p-1}))_{\delta\Downarrow O} \,][\, out(\alpha_p, \beta_1 \ldots \beta_{p-1})$ does not include all the **pass** states of $out(\alpha_p, \beta_1 \ldots \beta_{p-1})$.
- *Imp* is said to be *queued-suspension trace-included* in *Spec* if for all $\alpha \in I^*$ and all possible partitions of α into words $\alpha_1, \ldots, \alpha_p$, all the verdict states of IOTS $(\alpha_p \,][\, Imp\text{-}\mathbf{after}\text{-}(\alpha_1 \ldots \alpha_{p-1}, \beta_1 \ldots \beta_{p-1}))_{\delta\Downarrow O} \,][\, out(\alpha_p, \beta_1 \ldots \beta_{p-1})$ are labeled with the verdict **pass**.
- *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 verdict states of the IOTS $(\alpha_p \,][\, Imp\text{-}\mathbf{after}\text{-}(\alpha_1 \ldots \alpha_{p-1}, \beta_1 \ldots \beta_{p-1}))_{\delta\Downarrow O} \,][\, out(\alpha_p, \beta_1 \ldots \beta_{p-1})$ are labeled with the verdict **pass** and include all the **pass** states of $out(\alpha_p, \beta_1 \ldots \beta_{p-1})$.

Accordingly, the following is a generalization of Proposition 10.

Proposition 16. 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 $Qqtraces_o(Imp\text{-}\mathbf{after}\text{-}(\alpha_1 \ldots \alpha_{i-1}, \gamma_1 \ldots \gamma_{i-1}), \alpha_i) \cap Qqtraces_o(Spec\text{-}\mathbf{after}\text{-}(\alpha_1 \ldots \alpha_{i-1}, \gamma_1 \ldots \gamma_{i-1}), \alpha_i) = \varnothing$ for any consistent $\gamma_1 \ldots \gamma_{i-1}$.
- *Imp* 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 $traces(\alpha_i \,][\, Imp\text{-}\mathbf{after}\text{-}(\alpha_1 \ldots \alpha_{i-1}, \gamma_1 \ldots \gamma_{i-1}))_{\delta\Downarrow O} \not\subseteq traces(\alpha_i \,][\, Spec\text{-}\mathbf{after}\text{-}(\alpha_1 \ldots \alpha_{i-1}, \gamma_1 \ldots \gamma_{i-1}))_{\delta\Downarrow O}$.
- *Imp* 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 $traces(\alpha_i \,][\, Imp\text{-}\mathbf{after}\text{-}(\alpha_1 \ldots \alpha_{i-1}, \gamma_1 \ldots \gamma_{i-1}))_{\delta\Downarrow O} \neq traces(\alpha_i \,][\, Spec\text{-}\mathbf{after}\text{-}(\alpha_1 \ldots \alpha_{i-1}, \gamma_1 \ldots \gamma_{i-1}))_{\delta\Downarrow O}$.

- *Imp* is queued-suspension trace-included into *Spec*, iff *Imp* does not oscillate and $Qstraces(Imp) \subseteq Qstraces(Spec)$.
- *Imp* and *Spec* are queued-suspension trace-equivalent iff *Imp* does not oscillate and $Qstraces(Imp) = Qstraces(Spec)$.

The queued-suspension testing 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 $(\alpha_1 \dots \alpha_p, Out(\alpha_1 \dots \alpha_p))$, 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 15 obtaining appropriate notions of k-distinguishability. In particular, a queued-suspension trace of *Spec* $\alpha_1\beta_1\delta \dots \alpha_p\beta_p\delta \in Qstraces(Spec)$ is called a queued-suspension k-trace of *Spec* if $|\alpha_i| \le k$ for all $i = 1, \dots, p$. The set of all these traces $Qstraces^{\le k}(Spec)$ has a finite representation.

Definition 17. Let S_{stable} be the set of all stable states of an IOTS $Spec = \langle S, I, O, \lambda, S_0 \rangle$. A *queued-suspension k-machine* for *Spec* is a tuple $\langle R, I^{\le k}O^*\delta, \lambda^k_{stable}, r_0 \rangle$, denoted $Spec^k_{susp}$, where the starting state $r_0 = \{S_0\}$ and the set of states $R \subseteq \boldsymbol{P}(S_{stable}) \cup \{S_0\}$ ($\boldsymbol{P}(S_{stable})$ is a powerset of S_{stable}), and the transition relation λ^k_{stable} are the smallest sets obtained by application of the following rules:

- $(r, \alpha\beta\delta, r') \in \lambda^k_{stable}$ if $\alpha \in I^{\le k}$, $\beta \in O^*$ and r' is the set of states r-**after**-(α, β) in *Spec*.
- In case that some initial state of *Spec* is unstable $(r_0, \varepsilon\beta\delta, r') \in \lambda^k_{stable}$ if $\beta \in O^*$ and $\beta \ne \varepsilon$, and $r' = S_0$-**after**-(ε, β).

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

Proposition 18. The set of traces of $Spec^k_{susp}$ coincides with the set of queued-suspension k-traces of *Spec*.

Corollary 19. A non-oscillating IOTS *Imp* is queued-suspension k-distinguishable from *Spec* iff Imp^k_{stable} has a trace that is not a trace of $Spec^k_{susp}$.

Fig. 1 depicts the 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 $I^{\le k}$ and output set $O^{\le m}$ 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 applies inputs to an IUT via a finite input queue and output test process reads outputs that the IUT puts into a finite output queue until it detects no more outputs from the IUT, i.e., the tester detects quiescence of the IUT. In such a testing architecture, input from the tester and output from the

IUT may occur simultaneously. We call such a testing scenario queued testing. We elaborated 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 by 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 an IUT, namely that the IUT can issue output at any time and the tester cannot determine exactly the stimulus that causes the output. We believe that this assumption is less restrictive than any other assumption known in the testing literature [2], [6]. Testing with this assumption requires buffers between the IUT and tester. To make our approach practical, these buffers are considered finite, opposed to the case of infinite queues considered earlier [14]. We demonstrated that the implementation relations that can be verified by the queued testing are coarser than those previously considered. The test derivation procedures were elaborated with a fault model in mind. The resulting test suite becomes finite and related to the assumptions about potential faults, as opposed to the approach of [12], 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., [12].

Concerning future work, we believe that this paper may trigger research in various directions. It is interesting, for example, to see to which extent one could adapt FSM-based test derivation methods driven by fault models, as is done in [11] with a more restrictive assumption about a tester in mind.

Acknowledgment

The first author acknowledges fruitful discussions with Andreas Ulrich about testing IOTS. This work was in part supported by the NSERC grant OGP0194381. The second author acknowledges a partial support of the program “Russian Universities”. The third author acknowledges a partial support of ReSMiQ.

References

1. Bochmann, G. v., Petrenko, A.: Protocol Testing: Review of Methods and Relevance for Software Testing. In: The Proceedings of the ACM International Symposium on Software Testing and Analysis, ISSTA'94. USA (1994)
2. Brinksma, E., Tretmans, J.: Testing Transition Systems: An Annotated Bibliography. In: Cassez, F., Jard, C., Rozoy, B., Ryan, M. (eds.): Modeling and Verification of Parallel Processes. Lecture Notes in Computer Science, Vol. 2067. Springer-Verlag, Berlin Heidelberg New York (2001)
3. van Glabbeek, R. J.: The Linear Time-Branching Time Spectrum. In: The Proceedings of CONCUR’90. Lecture Notes In Computer Science, Vol. 458. Springer-Verlag, Berlin Heidelberg New York (1990)

4. Jard, C., Jéron, T., Tanguy, L., Viho, C.: Remote Testing Can Be as Powerful as Local Testing. In: The Proceedings of the IFIP Joint International Conference, Methods for Protocol Engineering and Distributed Systems, FORTE XII/PSTV XIX. China (1999)
5. Lynch, N., Tuttle, M. R.: An Introduction to Input/Output Automata. In: CWI Quarterly, Vol. 2, Issue 3 (1989)
6. Petrenko, A.: Fault Model-Driven Test Derivation from Finite State Models: Annotated Bibliography. In: Cassez, F., Jard, C., Rozoy, B., Ryan, M. (eds.): Modeling and Verification of Parallel Processes. Lecture Notes in Computer Science, Vol. 2067. Springer-Verlag, Berlin Heidelberg New York (2001)
7. Petrenko, A., Yevtushenko, N.: Queued Testing of Transition Systems with Inputs and Outputs. In: Hierons, R., Jeron, T. (eds.): INRIA preprint, the Proceedings of the Workshop on Formal Approaches to Testing of Software, FATES'02, A Satellite Workshop of CONCUR'02. Czech Republic (2002)
8. Petrenko, A., Yevtushenko, N., Bochmann, G. v., Dssouli, R.: Testing in Context: Framework and Test Derivation. In: Computer Communications, Vol. 19 (1996)
9. Phalippou, M.: Executable Testers. In: The Proceedings of the IFIP Sixth International Workshop on Protocol Test Systems, IWPTS'93. France (1993)
10. Segala, R.: Quiescence, Fairness, Testing and the Notion of Implementation. In: The Proceedings of CONCUR'93. Lecture Notes in Computer Science, Vol. 715. Springer-Verlag, Berlin Heidelberg New York (1993)
11. Tan, Q. M., Petrenko, A.: Test Generation for Specifications Modeled by Input/Output Automata. In: The Proceedings of the IFIP 11th International Workshop on Testing of Communicating Systems, IWTCS'98. Russia (1998)
12. Tretmans, J.: Test Generation with Inputs, Outputs and Repetitive Quiescence. In: Software-Concepts and Tools, Vol. 17, Issue 3 (1996)
13. Vaandrager, F.: On the Relationship between Process Algebra and Input/Output Automata. In: The Proceedings of Sixth Annual IEEE Symposium on Logic in Computer Science (1991)
14. Verhaard, L., Tretmans, J., Kim, P., Brinksma, E.: On Asynchronous Testing. In: The Proceedings of the IFIP 5th International Workshop on Protocol Test Systems, IWPTS'92. Canada (1992)
15. de Vries, R. G., Belinfante, A., Feenstra, J.: Automated Testing in Practice: The Highway Tolling System. In: The Proceedings of the IFIP 14th International Conference on Testing of Communicating Systems, TestCom'2002. Berlin, Germany (2002)