Generating complete and finite test suite for ioco: is it possible?

Workshop on Model-Based Testing, 9th, 2014, Grenoble.
http://www.producao.usp.br/handle/BDPI/45462

Generating Complete and Finite Test Suite for ioco: Is It Possible?

Adenilso Simao
São Paulo University
São Carlos, São Paulo, Brazil
adenilso@icmc.usp.br

Alexandre Petrenko
Centre de recherche informatique de Montreal (CRIM)
Montreal, Quebec, Canada
petrenko@crim.ca

Testing from Input/Output Transition Systems has been intensely investigated. The conformance between the implementation and the specification is often determined by the so-called **ioco**-relation. However, generating tests for **ioco** is usually hindered by the problem of conflicts between inputs and outputs. Moreover, the generation is mainly based on nondeterministic methods, which may deliver complete test suites but require an unbounded number of executions. In this paper, we investigate whether it is possible to construct a finite test suite which is complete in a predefined fault domain for the classical **ioco** relation even in the presence of input/output conflicts. We demonstrate that it is possible under certain assumptions about the specification and implementation, by proposing a method for complete test generation, based on a traditional method developed for FSM.

1 Introduction

Testing from Input/Output Transition System (IOTS) has received great attention from academy and industry alike. The main research goal is to devise a theoretically sound testing framework when the behavior of an Implementation Under Test (IUT) is specified as the IOTS model. It is assumed that the tester controls when inputs are applied, while the IUT autonomously controls when, and if, outputs are produced. The IUT's autonomy causes issues in testing. Simply stated, the interaction between the IUT and the tester should be assumed to be asynchronous, since otherwise the tester should have the ability to block the IUT when the latter is ready to produce output but the former has input to be sent. Most approaches based on the so-called **ioco** conformance relation do not offer sound solutions to the problem of conflicts between inputs and outputs. In particular, the proposal [15] for input-enabled testers addressing the conflicts lead to uncontrollable tests, while it is widely agreed that only controllable tests, which avoid any choice between inputs or between input and output, should be used. The approaches for test purpose driven test generation from the IOTS implemented in tools such as TGV [8] and TorX [16], as well as in Uppaal Tron which also accepts the IOTS, face the same problem of treating input/output conflicts.

These issues have drawn significant attention of the testing community, e.g., [1, 6, 7, 11], and have been dealt with by allowing implicitly or explicitly the presence of channels, e.g., FIFO queues, between the IUT and tester [6, 7, 17]. However, queues impose a hard burden on the tester, since the communication is now distorted by possible delay in the transmission of messages via queues. In the extreme case, queues render some important testing problems undecidable [4, 5]. The issue is caused by the conflict between input and output enabled in the same state; while the IUT should be ready to receive input, it may choose to produce an output, blocking or ignoring incoming input. It has been shown that when all the states have either inputs or outputs, but not both, in the so-called Mealy IOTS, such problems do not arise [12].

A. Petrenko, H. Schlingloff (Eds.): Ninth Workshop on
Model-Based Testing (MBT 2014)
EPTCS 141, 2014, pp. 56–70, doi:10.4204/EPTCS.141.5

Apart for the problem of input/output conflicts, the question of generating complete and finite test suite from IOTS w.r.t. the **ioco** relation remains open. The test generation method which is most referred in the literature relies on non-deterministic choice between: (1) stopping testing; (2) applying a randomly chosen input; or (3) checking for outputs [14]. The problem with this approach is that, although completeness is guaranteed in some theoretical sense, the practical application of this method is problematic. It requires that the process be repeated an undetermined number of times, since there is no indication of when the completeness has been achieved and thus the process can stop.

On the other hand, generation methods from Finite State Machines (FSM) approach the problem of test completeness by explicitly stating a set of faulty (mutant) FSMs, called a fault domain, which model potential faults of the IUT; then, a test suite is generated that targets each faulty FSM. Its completeness implies that each IUT possessing the modelled faults will be detected by the test suite. The existing methods for complete test generation are applicable not only to minimal deterministic machines, as the early methods [2, 3, 18], but also to nondeterministic FSMs [10]. This motivated a previous attempt to rephrase FSM methods for checking experiments to the IOTS model [13]. In particular, an analogue of the Harmonized State Identifier Method (HSI-method) was elaborated there for the trace equivalence relation between the specification and implementation IOTSs. The input/output conflicts were addressed by assuming that the tester detecting (using some means) them will just try to repeatedly re-execute the expected trace to verify if it can be generated by the IUT.

In this paper, we investigate whether it is possible to construct a finite test suite for a given IOTS specification which is complete in a predefined fault domain for the classical **ioco** relation even in the presence of input/output conflicts. Our solution to the latter is based on the assumption that any IUT in a fault domain resolves each such conflict in favor of inputs; that is, we assume that the IUT is eager to process inputs and, whenever it is in a state where it can either receive an input or produce an output, it will produce an output only if no input is available. We demonstrate this by elaborating a test generation method inspired by the HSI method [19], generalizing and adapting its concepts to the realm of IOTS. We illustrate the method with a running example.

This remainder of this paper is organized as follows. In Section 2, we introduce the main concepts of IOTS and test cases. In Section 3, we present the generation method, and demonstrate that the obtained test suite is a complete for a given fault domain. Finally, in Section 4, we conclude the paper and point to future work.

2 Input/output transition system and test cases

2.1 Input/output transition system and related definitions

We use *input/output transition systems* (IOTS, a.k.a. input/output automata [9]) for modelling systems. Formally, an IOTS $\mathcal{S}$ is a quintuple $(S, s_0, I, O, h_{\mathcal{S}})$, where S is a finite set of states and $s_0 \in S$, is the initial state, I and O are disjoint sets of input and output actions, respectively, and $h_{\mathcal{S}} \subseteq S \times (I \cup O) \times S$ is the transition relation. $\mathcal{S}$ is *deterministic* if $h_{\mathcal{S}}$ is a function on a subset of $S \times (I \cup O)$, i.e., if $(s,x,s') \in h_{\mathcal{S}}$ and $(s,x,s'') \in h_{\mathcal{S}}$, then $s' = s''$. While we shall consider only deterministic IOTSs, they may have output-nondeterminism, i.e., have several outputs enabled in a state.

For IOTS $\mathcal{S}$, let $init_{\mathcal{S}}(s)$ denote the set of actions enabled at state s, i.e., $init_{\mathcal{S}}(s) = \{x \in I \cup O \mid \exists s' \in S, (s,x,s') \in h_{\mathcal{S}}\}$; let $inp_{\mathcal{S}}(s)$ and $out_{\mathcal{S}}(s)$ denote the set of inputs and outputs, respectively, enabled at state s. Thus, $inp_{\mathcal{S}}(s) = init_{\mathcal{S}}(s) \cap I$; $out_{\mathcal{S}}(s) = init_{\mathcal{S}}(s) \cap O$. We omit the subscript if it is clear which IOTS is considered.

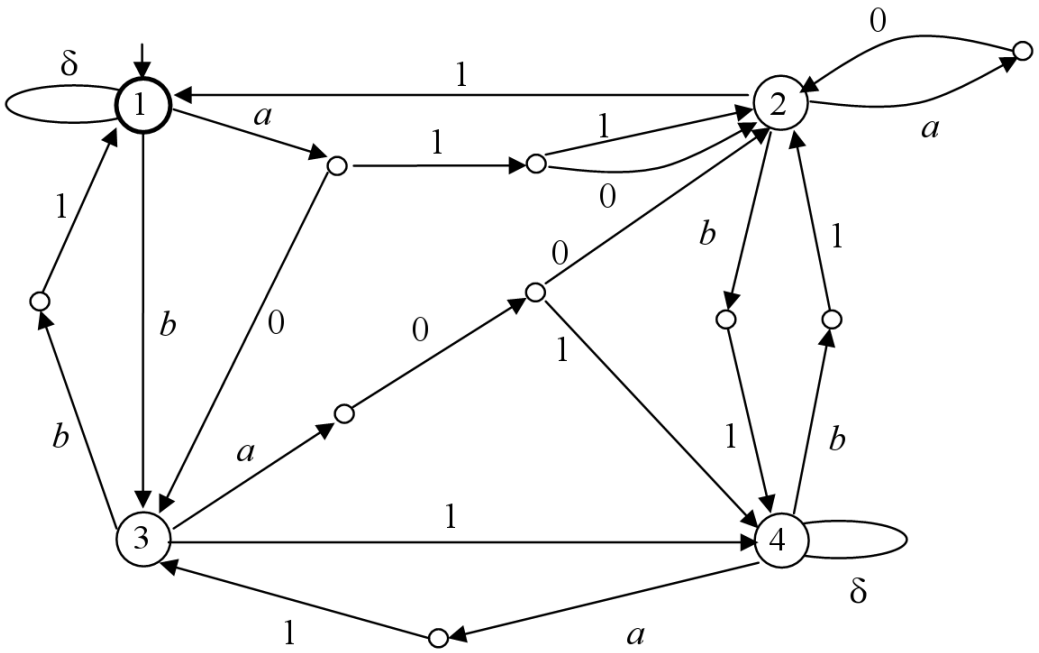


Figure 1: An IOTS.

A state s is a *sink* state if $init(s) = \emptyset$; s is an *input state* if $inp(s) \neq \emptyset$. We denote the set of input states by S_{in}. An input state s is *stable* (quiescent) if $init(s) \subseteq I$. An input state s is a *quasi-stable* state if $out(s) \neq \emptyset$. In a quasi-stable state, there is an input/output conflict (note that the IOTS itself does not provide any mechanism for resolving such conflicts). A state is an *output state* if it is neither sink nor input state. Figure 1 shows an example of an IOTS, where $I = \{a,b\}$ and $O = \{0,1\}$. Input states are numbered; states 1 and 4 are stable, whereas states 2 and 3 are quasi-stable.

For IOTS $\mathcal{S}$, a *path* from state s_1 to state s_{n+1} is a sequence of transitions $p = (s_1,a_1,s_2)(s_2,a_2,s_3)\dots(s_n,a_n,s_{n+1})$, where $(s_i,a_i,s_{i+1}) \in h_{\mathcal{S}}$ for $i = 1,\dots,n$. Let ε denote the empty sequence of actions. We say that s_{n+1} is *reachable* from s_1. IOTS $\mathcal{S}$ is *initially-connected* if each state is reachable from the initial state. A sequence $u \in (I \cup O)^*$ is called a *trace* of $\mathcal{S}$ from state $s_1 \in S$ if there exists path $(s_1,a_1,s_2)(s_2,a_2,s_3)\dots(s_n,a_n,s_{n+1})$, such that $u = a_1 \dots a_n$. We use the usual operator *after* to denote the state reached after the sequence of actions (we consider only deterministic IOTS), i.e., s_1-after-$u = s_{n+1}$; if u is not a trace of s_1, then s_1-after-$u = \emptyset$. Let also $Tr(T)$ denote the set of traces from states in $T \subseteq S$. For simplicity, we denote $Tr(\{s\})$ as $Tr(s)$ and use $Tr(\mathcal{S})$ to denote $Tr(s_0)$. A trace u of IOTS $\mathcal{S}$ is *completed*, if s_0-after-u is a sink state. A trace u of IOTS $\mathcal{S}$ is a *bridge trace from input state s*, if s-after-$u \in S_{in}$ and for each proper prefix w of u, s-after-$w \notin S_{in}$.

Given an IOTS $\mathcal{S} = (S,s_0,I,O,h_{\mathcal{S}})$ and a state $s \in S$, let $\mathcal{S}/s$ denote the IOTS that differs from $\mathcal{S}$ in the initial state changed to s, removing states and transitions which are unreachable from s.

We use a designated symbol δ to indicate quiescence in $\mathcal{S}$, that is, the absence of outputs. Quiescence can be encoded by adding self-looping δ transitions to the stable states; the resulting IOTS has the output action set $O \cup \{\delta\}$. Traces of this IOTS which end with δ are quiescent traces and traces containing δ are *suspension* traces. In the rest of the paper, we assume that $Tr(\mathcal{S})$ includes all kinds of traces.

An IOTS $\mathcal{T} = (T,t_0,I,O,h_{\mathcal{T}})$ is a *submachine* of the IOTS $\mathcal{S} = (S,s_0,I,O,h_{\mathcal{S}})$, if $T \subseteq S$ and $h_{\mathcal{T}} \subseteq h_{\mathcal{S}}$. A state $s \in T$ of a submachine $\mathcal{T}$ of $\mathcal{S}$ is *output-preserving* if for each $x \in O$ such that $(s,x,s') \in h_{\mathcal{S}}$, we

have that $(s,x,s') \in h_{\mathcal{T}}$. The submachine $\mathcal{T}$ of $\mathcal{S}$ is *output-preserving* if each state which is not a sink state is output-preserving. The submachine is *trivial* if T is a singleton and $h_{\mathcal{T}} = \emptyset$.

The IOTS $\mathcal{S}$ is *progressive* if it has no sink state and each cycle contains a transition labeled with input, i.e., there is no output divergence. The IOTS $\mathcal{S}$ is *input-complete* if all inputs are enabled in input states, i.e., $inp(s) \neq \emptyset$ implies that $inp(s) = I$, for each state s. The IOTS $\mathcal{S}$ is *single-input* if $|inp(s)| = 1$, for each input state s; it is *output-complete* if $out(s) = O$, for each output state s.

In this paper, we assume that specifications and implementations are input-complete progressive deterministic initially-connected IOTS; we let $IOTS(I,O)$ denote the set of such IOTSs with input set I and output set O.

To characterize the common behavior of two IOTSs in $IOTS(I,O)$ we use the intersection operation. The *intersection* $\mathcal{S} \cap \mathcal{P}$ of IOTSs $\mathcal{S} = (S,s_0,I,O,h_{\mathcal{S}})$ and $\mathcal{P} = (P,p_0,I,O,h_{\mathcal{P}})$ is an IOTS $(Q,q_0,I,O,h_{\mathcal{S}\cap\mathcal{P}})$ with the state set $Q \subseteq S \times P$, the initial state $q_0 = (s_0,p_0)$, and the transition relation $h_{\mathcal{S}\cap\mathcal{P}}$, such that Q is the smallest state set obtained by using the rule $((s,p),x,(s',p')) \in h_{\mathcal{S}\cap\mathcal{P}} \iff (s,x,s') \in h_{\mathcal{S}}$ and $(p,x,p') \in h_{\mathcal{P}}$. The intersection $\mathcal{S} \cap \mathcal{P}$ preserves only common traces of both machines; in other words, for each state (s,p) of $\mathcal{S} \cap \mathcal{P}$ we have $Tr((s,p)) = Tr(s) \cap Tr(p)$; moreover, $out((s,p)) = out(s) \cap out(p)$. Thus, $Tr(\mathcal{S} \cap \mathcal{P}) = Tr(\mathcal{S}) \cap Tr(\mathcal{P})$.

Given two IOTSs $\mathcal{S}$ and $\mathcal{T}$, such that $\mathcal{S}$ has at least one sink state $s \in S$, the IOTS obtained by merging the initial state of $\mathcal{T}$ with a sink state s is called the *chaining* of $\mathcal{S}$ and $\mathcal{T}$ in the sink state s, denoted $\mathcal{S}@_s\mathcal{T}$.

For conformance testing, we consider a usual **ioco** relation.

Definition 1 *Given two IOTSs $\mathcal{P},\mathcal{S} \in IOTS(I,O)$, $\mathcal{S} = (S,s_0,I,O,h_{\mathcal{S}})$ and $\mathcal{P} = (P,p_0,I,O,h_{\mathcal{P}})$, we write $\mathcal{P}$ **ioco** $\mathcal{S}$ if for each trace $\alpha \in Tr(\mathcal{S})$, we have that $out(\mathcal{P}\text{-after-}\alpha) \subseteq out(\mathcal{S}\text{-after-}\alpha)$. If $\mathcal{P}$ **ioco** $\mathcal{S}$ then we say that state p_0 is a* reduction *of state s_0. The reduction relation between states is also defined for states of the same IOTS $\mathcal{S} \in IOTS(I,O)$, namely, s_1 is a* reduction *of s_2, if $\mathcal{S}/s_1$ **ioco** $\mathcal{S}/s_2$.*

We write $\mathcal{P}$ ~~**ioco**~~ $\mathcal{S}$, if not $\mathcal{P}$ **ioco** $\mathcal{S}$. We notice that if the specification IOTS $\mathcal{S}$ contains some state that is a reduction of another state then there exist an implementation $\mathcal{P} \in IOTS(I,O)$ and state $p \in P$, that is a reduction of both states of $\mathcal{S}$. Intuitively, the two states are "merged" into a single state in the implementation. As a result, a conforming implementation may have fewer states than its specification. This observation motivates the following definitions and statements.

Definition 2 *Two states of $\mathcal{S} \in IOTS(I,O)$ are* compatible*, if there exists a state of an IOTS $\mathcal{P} \in IOTS(I,O)$ that is a reduction of both states; otherwise, i.e., if for any $\mathcal{P} \in IOTS(I,O)$, no state of $\mathcal{P}$ is a reduction of both states, they are* distinguishable*.*

According to this definition, compatible states can be "merged" in an implementation IOTS into a single state and it can still be a reduction of the specification IOTS, however, any reduction of the specification IOTS cannot have a state that is a reduction of distinguishable states.

The compatibility of states can be easily determined by the intersection of IOTSs, a simple and inexpensive operation. By definition, if two states of a given IOTS are compatible, there exists a state of some input-complete, progressive IOTS which is a reduction of both states. Such a state is the initial state of the intersection of two instances of a given machine initialized in different states, since the intersection represents all the common traces of the two states. On the other hand, if the two states are distinguishable, the intersection is not a progressive IOTS. This fact is stated in the following lemma.

Lemma 1 *Two states $s_1,s_2 \in S$ of $\mathcal{S} = (S,s_0,I,O,h_{\mathcal{S}})$, $\mathcal{S} \in IOTS(I,O)$ are compatible if and only if $\mathcal{S}/s_1 \cap \mathcal{S}/s_2 \in IOTS(I,O)$.*

Proof. Suppose that s_1 and s_2 are compatible. We show that $\mathcal{S}/s_1 \cap \mathcal{S}/s_2 \in IOTS(I,O)$, that is, $\mathcal{S}/s_1 \cap \mathcal{S}/s_2$ is input-complete, progressive, deterministic and initially-connected. Let $\alpha \in Tr(\mathcal{S}/s_1 \cap \mathcal{S}/s_2)$. Thus, $\alpha \in Tr(\mathcal{S}/s_1) \cap Tr(\mathcal{S}/s_2)$. We have that $s'_1 = \mathcal{S}/s_1$-after-α and $s'_2 = \mathcal{S}/s_2$-after-α are also compatible. Hence, by Definition 1, there exists a state p of $\mathcal{P} \in IOTS(I,O)$, with $\mathcal{P} = (P, p_0, I, O, h_{\mathcal{P}})$ that is a reduction of s'_1 and s'_2. It holds that $out(p) \subseteq out(s'_1)$ and $out(p) \subseteq out(s'_2)$. As $\mathcal{P}$ is progressive, we have that $init(p) \neq \emptyset$, and thus there exists $x \in out(p)$; hence, $x \in init(s'_2) \cap init(s'_2)$. It follows that $(\mathcal{S}/s_1 \cap \mathcal{S}/s_2)$-after-$\alpha$ is not a sink state, since it is followed by x, at least. Thus, $\mathcal{S}/s_1 \cap \mathcal{S}/s_2$ has no sink state. If x is an input, then $I \subseteq init(s'_1)$ and $I \subseteq init(s'_2)$, since $\mathcal{S}$ is input-complete. Therefore, $I \subseteq init((\mathcal{S}/s_1 \cap \mathcal{S}/s_2)$-after-$\alpha)$, and $\mathcal{S}/s_1 \cap \mathcal{S}/s_2$ is input-complete. As $\mathcal{S}$ is progressive, it does not have cycles with transitions labeled only with outputs. Hence, neither $\mathcal{S}/s_1 \cap \mathcal{S}/s_2$ has such cycles, i.e., $\mathcal{S}/s_1 \cap \mathcal{S}/s_2$ is also progressive. As $\mathcal{S}$ is deterministic and initially-connected, so are $\mathcal{S}/s_1, \mathcal{S}/s_2$ and, consequently, $\mathcal{S}/s_1 \cap \mathcal{S}/s_2$. It follows then that $\mathcal{S}/s_1 \cap \mathcal{S}/s_2 \in IOTS(I,O)$.

Suppose now that the intersection $\mathcal{S}/s_1 \cap \mathcal{S}/s_2 \in IOTS(I,O)$, i.e., it is input-complete, progressive, deterministic and initially-connected. We show that s_1 and s_2 are compatible, demonstrating that the initial state of $\mathcal{S}/s_1 \cap \mathcal{S}/s_2$ is a reduction of s_1 and s_2. For each trace $\alpha \in Tr(\mathcal{S}/s_1 \cap \mathcal{S}/s_2)$, we have that $init((\mathcal{S}/s_1 \cap \mathcal{S}/s_2)$-after-$\alpha) \subseteq init(\mathcal{S}/s_1$-after-$\alpha) = init(s_1$-after-$\alpha)$; thus, $init((\mathcal{S}/s_1 \cap \mathcal{S}/s_2)$-after-$\alpha) \cap O = out((\mathcal{S}/s_1 \cap \mathcal{S}/s_2)$-after-$\alpha) \subseteq init(\mathcal{S}/s_1$-after-$\alpha) \cap O = out(\mathcal{S}/s_1$-after-$\alpha) = out(s_1$-after-$\alpha)$. Therefore, the initial state of $\mathcal{S}/s_1 \cap \mathcal{S}/s_2$ is a reduction of s_1. Analogously, $\mathcal{S}/s_1 \cap \mathcal{S}/s_2$ is a reduction of s_2 and the result thus follows.$\diamond$

Corollary 1 *States s_1 and s_2 of $\mathcal{S}$ are distinguishable if and only if $\mathcal{S}/s_1 \cap \mathcal{S}/s_2 \notin IOTS(I,O)$, i.e., the IOTS $\mathcal{S}/s_1 \cap \mathcal{S}/s_2$ has a sink state.*

An IOTS in $IOTS(I,O)$ is *input-state-minimal* if every two input states are distinguishable. In the following, we assume that IOTSs which are not input-state-minimal are excluded from $IOTS(I,O)$.

The next lemma states when one state of an IOTS is a reduction of another. The outputs enabled in each state reached in the intersection IOTS, initialized with the respective states, are exactly the outputs enabled in one of the states.

Lemma 2 *Given two states $s_1, s_2 \in S$ of $\mathcal{S} = (S, s_0, I, O, h_{\mathcal{S}})$, s_1 is a reduction of s_2 if and only if $out((s,s')) = out(s)$ for each state (s,s') of $\mathcal{S}/s_1 \cap \mathcal{S}/s_2$.*

Proof. Assume that s_1 is a reduction of s_2; thus, $\mathcal{S}/s_1$ **ioco** $\mathcal{S}/s_2$. We have that for each trace $\alpha \in Tr(\mathcal{S}/s_2)$, $out(s_1$-after-$\alpha) \subseteq out(s_2$-after-$\alpha)$. Let (s,s') be a state of $\mathcal{S}/s_1 \cap \mathcal{S}/s_2$. Thus, there exists a trace $\beta \in Tr(\mathcal{S}/s_1 \cap \mathcal{S}/s_2)$, such that$(\mathcal{S}/s_1 \cap \mathcal{S}/s_2)$-after-$\beta = (s,s')$ and, therefore, $\mathcal{S}/s_1$-after-$\beta = s$ and $\mathcal{S}/s_2$-after-$\beta = s'$. It holds that $\beta \in Tr(\mathcal{S}/s_2)$ and $out(s_1$-after-$\beta) \subseteq out(s_2$-after-$\beta)$; thus, $out(s) \subseteq out(s')$. We have that $out((s,s')) = out(s) \cap out(s')$. The result then follows, since $out(s) \subseteq out(s')$ and $out((s,s')) = out(s) \cap out(s')$ implies that $out((s,s')) = out(s)$. Assume now that $out((s,s')) = out(s)$ for each state (s,s') of $\mathcal{S}/s_1 \cap \mathcal{S}/s_2$. Let $\alpha \in Tr(\mathcal{S}/s_2)$. We have that $\alpha \in Tr(\mathcal{S}/s_1)$ if and only if $\alpha \in Tr(\mathcal{S}/s_1 \cap \mathcal{S}/s_2)$. If $\alpha \notin Tr(\mathcal{S}/s_1)$, then $out(\mathcal{S}/s_1$-after-$\alpha) = \emptyset$ and the result follows, since $out(\mathcal{S}/s_1$-after-$\alpha) \subseteq out(\mathcal{S}/s_2$-after-$\alpha)$. If $\alpha \in Tr(\mathcal{S}/s_1)$, let $(s,s') = \mathcal{S}/s_1$-after-$\alpha \cap \mathcal{S}/s_2$-after-$a$; thus, $s = \mathcal{S}/s_1$-after-a and $s' = \mathcal{S}/s_2$-after-a. We have that $out(\mathcal{S}/s_1$-after-$\alpha \cap \mathcal{S}/s_2$-after-$\alpha) = out(\mathcal{S}/s_1$-after-$\alpha)$. Let $x \in out(\mathcal{S}/s_1$-after-$\alpha)$. As $x \in out(\mathcal{S}/s_1$-after-$\alpha \cap \mathcal{S}/s_2$-after-$\alpha) = out(\mathcal{S}/s_1$-after-$\alpha) \cap out(\mathcal{S}/s_2$-after-$\alpha)$, it holds that $x \in out(\mathcal{S}/s_2$-after-$\alpha)$. The result then follows, since $out(\mathcal{S}/s_1$-after-$\alpha) \subseteq out(\mathcal{S}/s_2$-after-$\alpha)$, implying that $\mathcal{S}/s_1$ **ioco** $\mathcal{S}/s_2$, i.e., s_1 is a reduction of s_2.$\diamond$

2.2 Test definitions and problem statement

To simplify the discussion, we refer to inputs and outputs always taking the view of the implementation, IUT; thus, we say, for instance, that the tester sends an input to the IUT and receives outputs from it, and define test cases accordingly preserving the input and output sets of the specification IOTS $\mathcal{S} = (S, s_0, I, O, h_{\mathcal{S}})$. Recall that δ is included into O; in particular, the output δ of a test case is interpreted as the fact that the tester executing the test case detects quiescence of the IUT.

Definition 3 *A test case over input set I and output set O is an acyclic single-input output-complete IOTS $\mathcal{U} = (U, u_0, I, O, h_{\mathcal{U}})$, where U has a designated sink state* fail. *A test case is* controllable *if it has no quasi-stable states, otherwise it is* uncontrollable. *A* test suite *is a finite set of test cases.*

Let $Tr_{fail}(\mathcal{U})$ be the traces which lead to the sink state $fail$, i.e., $Tr_{fail}(\mathcal{U}) = \{\alpha \in Tr(\mathcal{U}) \mid \mathcal{U}\text{-after-}\alpha = fail\}$. Let $Tr_{pass}(\mathcal{U})$ be the traces which do not lead to $fail$, i.e., $Tr_{pass}(\mathcal{U}) = Tr(\mathcal{U}) \backslash Tr_{fail}(\mathcal{U})$.

Definition 4 *Given the specification IOTS $\mathcal{S}$, a test case $\mathcal{U} = (U, u_0, I, O, h_{\mathcal{U}})$, and an implementation IOTS $\mathcal{B} \in IOTS(I, O)$,*

- *$\mathcal{B}$ passes the test case $\mathcal{U}$, if the intersection $\mathcal{B} \cap \mathcal{U}$ has no state, where the test $\mathcal{U}$ is in the state fail.*
- *$\mathcal{B}$ fails $\mathcal{U}$, if the intersection $\mathcal{B} \cap \mathcal{U}$ has a state, where the test $\mathcal{U}$ is in the state fail.*

A test suite T is

- sound for IOTS $\mathcal{S}$ in $IOTS(I, O)$, *if each $\mathcal{B} \in IOTS(I, O)$, such that $\mathcal{B}$ **ioco** $\mathcal{S}$, passes each test in T.*
- exhaustive for IOTS $\mathcal{S}$ in $IOTS(I, O)$, *if each IOTS $\mathcal{B} \in IOTS(I, O)$, such that $\mathcal{B}$ ~~**ioco**~~ $\mathcal{S}$, fails some test in T.*
- complete for IOTS $\mathcal{S}$ in $IOTS(I, O)$ *w.r.t. the **ioco** relation, if T is sound and exhaustive for $\mathcal{S}$ in $IOTS(I, O)$.*

Notice that $\mathcal{B}$ passes the test case $\mathcal{U}$, if and only if $Tr(\mathcal{B}) \cap Tr_{fail}(\mathcal{U}) = \emptyset$ and $Tr_{pass}(\mathcal{U}) \subseteq Tr(\mathcal{B})$.

The problem of complete test suite generation for a given IOTS was addressed in [14, 15]. To generate such a test suite a simple algorithm is suggested, which, however, should be executed an indeterminate number of times to achieve the test completeness w.r.t. the **ioco** relation. In the first work [14], only controllable test cases are generated; the problem with that solution is that the tester must be able to somehow preempt any output each time a test case prescribes sending some input to the IUT. In the second work [15], "the most important technical change with respect to [14] is the input enabledness of test cases, which was inspired by [11]". In terms of our definitions, test cases are uncontrollable; they contain quasi-stable states, where both inputs and outputs are enabled. The intention behind this is to address input/output conflict present in the specification IOTS, since the specification itself provides no clue how an implementation resolves input/output conflict. The behavior of the tester executing uncontrollable test cases may become nondeterministic (the tester has to execute one of the two mutually exclusive actions) and the test results may not always be reproducible. The approaches to generation of controllable tests that tolerate input/output conflicts based on the use of queues are elaborated in several work [4, 6, 7, 11, 12, 17]. The problem is that one needs to know the size of queues to obtain a finite complete test suite.

In this paper, we demonstrate, first, that controllable tests that tolerate input/output conflicts can be constructed without knowing the size of queues, and second, that it is possible to obtain in a systematic

way a finite set of controllable test cases which is a complete test suite in a finite set of IOTSs. The key assumption we make about the implementation IOTSs in the fault domain is that each implementation when it is a quasi-stable state with the input/output conflict, it does not produce any output if its input queue contains an input. We call such implementations *input-eager*. A subset of $IOTS(I,O)$ that contains input-eager IOTSs is denoted $IEIOTS(I,O)$. Finiteness of complete test suites results from further constraining this set by the number of its input states, as we demonstrate later.

Testing any input-eager IOTS allows one to use two controllable test cases dealing with input/output conflict; in a quasi-stable state one test case does not send any input and only observes output sequence concluded by quiescence and another one just sends input. In the latter case, the tester does not need to preempt IUT outputs, as an input-eager IOTS will not produce them since the input queue is not empty and contains the input from the tester.

3 Generating complete test suites for IOTS

In this section, we investigate whether a classical method for constructing a complete test suite for the FSM model can be reworked to achieve the same result for the IOTS model even with input/output conflicts, namely a test suite with controllable test cases complete in a finite fault domain, without transforming IOTS into Mealy machine. To demonstrate that it is in fact possible, we develop here a counter-part of the HSI-method [19] for the simplest case, when the FSM is completely specified, minimal, and the fault domain contains FSMs with the number of states not exceeding that of the specification machine.

The HSI-method for FSMs uses sets of distinguishing input sequences, so-called harmonized state identifiers, one per state, such that any two identifiers share an input sequence which distinguishes the two states. These input sequences are appended to state and transition covers in order to check that every state of the implementation corresponds to some state of the specification and every transition of the implementation corresponds to a transition of the specification.

Accordingly, we need first to define state and transition covers, as well as harmonized state identifiers for a given IOTS.

3.1 State and transition covers for IOTS

We first turn our attention to the notion of state cover, needed in tests to eventually establish a mapping from states of the specification to states of the IUT. We focus only on input states of the specification IOTS. First, to check the IUTs reaction to some input it is in fact sufficient to apply the input to a given input state, observe an output sequence, and if it is correct then check whether a proper input state is reached. Output state identification can thus be avoided. However, even considering only input states, some input state of the specification may not be mapped to any state of the IUT even if the latter is a reduction of the specification. Therefore, we should define a state cover targeting only those input states of the specification which have a corresponding state in any **ioco**-conforming implementation.

Definition 5 *Given an initially connected IOTS* $\mathbb{S}$ *and an input state s, s is* certainly reachable (c-reachable)*, if any* $\mathcal{P} \in IOTS(I,O)$*, such that* $\mathcal{P}$ ***ioco*** $\mathbb{S}$*, contains an input state that is a reduction of s.*

It turns out that the certainly reachable states can be determined by considering a submachine of $\mathbb{S}$, similarly to the FSM case [10].

Lemma 3 *An input state s of an IOTS* $\mathbb{S}$ *is c-reachable if* $\mathbb{S}$ *contains a single-input acyclic output-preserving submachine of* $\mathbb{S}$ *which has s as the only sink state.*

Proof. Let $\mathcal{C}_s$ be a single-input acyclic output-preserving submachine of $\mathcal{S}$, which has s as the sink state. The input state s is the only sink state in the submachine; hence all its completed traces converge in s. The submachine is output-preserving, this means that for each $\alpha \in Tr(\mathcal{C}_s)$, if $\mathcal{C}_s$-after-$\alpha \neq s$ then $out(\mathcal{C}_s\text{-after-}\alpha) = out(\mathcal{S}\text{-after-}\alpha)$. Hence for any IOTS $\mathcal{P} \in IOTS(I,O)$, such that $\mathcal{P}$ **ioco** $\mathcal{S}$, it also holds that $out(\mathcal{P}\text{-after-}\alpha) \subseteq out(\mathcal{S}\text{-after-}\alpha)$, thus $out(\mathcal{P}\text{-after-}\alpha) \subseteq out(\mathcal{C}_s\text{-after-}\alpha)$. This implies that $\mathcal{P}$ should have at least one of the completed traces of $\mathcal{C}_s$; let β be such a completed trace. It is easy to see that $\mathcal{P}$ **ioco** $\mathcal{S}$ implies that for any $\gamma \in Tr(\mathcal{P})$, $\mathcal{P}$-after-γ is a reduction of $\mathcal{S}$-after-γ. Hence in any IOTS $\mathcal{P} \in IOTS(I,O)$, such that $\mathcal{P}$ **ioco** $\mathcal{S}$, the state $\mathcal{P}$-after-β is a reduction of $\mathcal{S}$-after-β. The result follows, since β is a completed trace of $\mathcal{C}_s$ and, thus, $\mathcal{S}$-after-$\beta = s$.$\diamond$

Definition 6 *Given a c-reachable input state s of an IOTS $\mathcal{S}$, a single-input acyclic output-preserving submachine $\mathcal{C}_s$, which has s as the only sink state, is a* preamble *for state s.*

Preambles for states can be determined by Algorithm 1, adapted from [10].

Algorithm 1 for constructing a preamble for a given input state.
Input: An IOTS $\mathcal{S}$ and input state $s \in S$.
Output: a preamble if the state s is c-reachable.
Construct an IOTS $\mathcal{R} = (R, r_0, I, O, h_{\mathcal{R}})$ as follows
$R := \{s\}$;
$h_{\mathcal{R}} := \emptyset$;
While $s_0 \notin R$ and there exist an input state $s' \notin R$ and nonempty $A \subseteq I$, such that for each $x \in A$, $(s', x, s'') \in h_{\mathcal{S}}$, and for each trace $\gamma \in Tr(s'')$, where $\gamma \in O^*$, there exists a prefix γ' such that s''-after-$\gamma' \in R$.
$R := R \cup \{s'\} \cup \{s''\text{-after-}\alpha \mid \gamma \in O*, \gamma \in Tr(s''), \alpha \in pref(\gamma')\}$;
$h_{\mathcal{R}} := h_{\mathcal{R}} \cup \{(s', x, s'') \in h_{\mathcal{S}} \mid x \in A\} \cup \{(s''\text{-after-}\alpha, o, s''\text{-after-}\alpha o) \mid \gamma \in O^*, \gamma \in Tr(s''), \alpha o \in pref(\gamma')\}$;
End While;
If $s_0 \notin R$ then return the message "the state s is not c-reachable" and stop;
Else let $\mathcal{R} = (R, r_0, I, O, h_{\mathcal{R}})$, where $r_0 := s_0$, be the obtained IOTS;
Starting from the initial state, remove in each state all input transitions, but one, to obtain a single-input submachine with the only sink state s;
Delete states which are unreachable from the initial state;
Return the obtained machine as a preamble for the state s and stop. $\diamond$

A preamble can be used to transfer from the initial state to c-reachable input states. For the initial state itself, the preamble is simply the trivial IOTS, which contains only the initial state. Figures 2.a, 2.b and 3 show the preambles for states 2, 3 and 4, respectively, of the IOTS in Figure 1.

We assume that each input state of the specification IOTS $\mathcal{S}$ is c-reachable and the initial state is a stable state. An *input state cover* Z of $\mathcal{S}$ is a set of preambles, one for each input state, i.e., $Z = \{\mathcal{C}_s \mid s \in S_{in}\}$.

In FSM-based testing, a state cover is extended to a transition cover, by adding all inputs to each transfer sequence of the state cover. In an IOTS, an input applied in an input state may be followed by a number of output sequences leading to various stable states, creating quiescent traces of IOTS. The set of all possible quiescent traces created by $x \in I$ in input state $s \in S_{in}$ is $\{x\gamma\delta \in Tr(s) \mid \gamma \in O^*\}$. We use $Cov(s,x)$, called (s,x)-*cover*, to refer to an IOTS, such that $Tr(Cov(s,x)) = \{x\gamma\delta \in Tr(s) \mid \gamma \in O^*\}$ and the set of sink states is $\{s\text{-after-}x\gamma \mid \gamma \in O^*\}$. For instance, $Cov(2,a)$ for state 2 and input a of

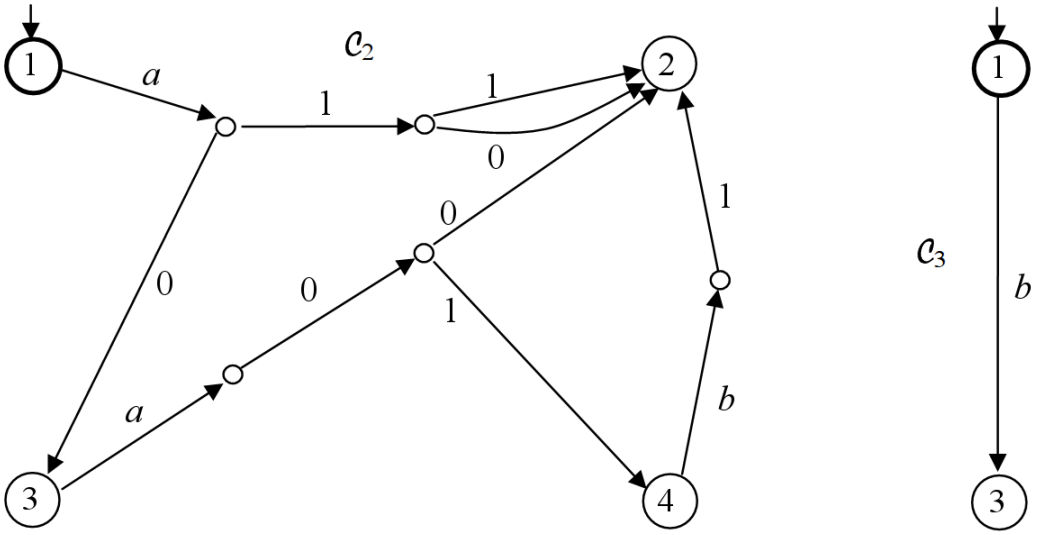


Figure 2: Preambles $\mathcal{C}_2$ and $\mathcal{C}_3$.

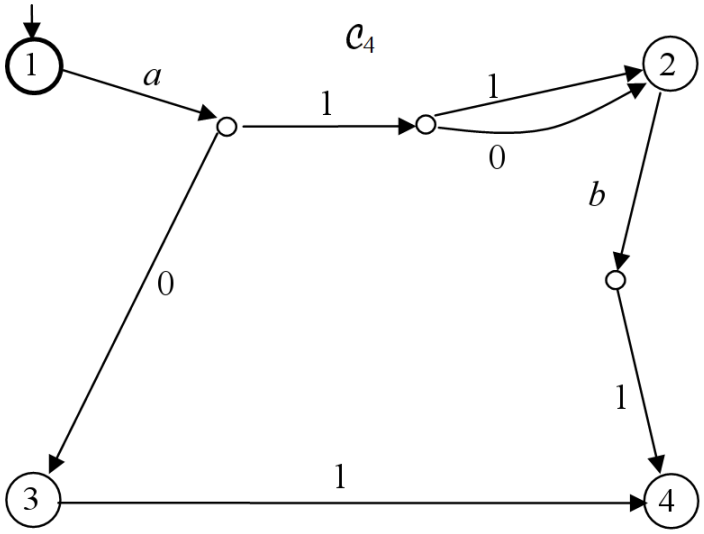


Figure 3: Preamble $\mathcal{C}_4$.

the IOTS in Figure 1 has the trace $a01\delta$, whereas $Cov(1,a)$ has the traces $a01\delta$, $a111\delta$ and $a101\delta$. A *transition cover V* of $\mathcal{S}$ is the set of preambles of an input state cover chained with (s,x)-covers, i.e., $V = \{\mathcal{C}_s @_s Cov(s,x) \mid s \in S_{in}, x \in I\}$. Notice that each bridge trace starting from a quasi-stable state $s \in S_{in}$ is covered by $Cov(s',x)$, for some input state s' and input x. More generally, we state the following lemma.

Lemma 4 *Given an IOTS $\mathcal{S} \in IOTS(I,O)$ and a bridge trace β from an input state $s \in S_{in}$, there exist input state $s' \in S_{in}$ and input x, such that $\gamma\beta\gamma'\delta \in Tr(Cov(s',x))$, for some traces $\gamma \in Tr(s')$ and $\gamma' \in Tr(s'\text{-after-}\gamma\beta)$.*

Proof. If β starts with an input, then the results follows directly, since with γ as the empty sequence $\beta\gamma' d$ is a quiescent trace starting at state s. If β starts with an output, then, $\beta \in O^*$ and s is a quasi-stable state. Notice that there exists $\gamma' \in O^*$, such that $\beta\gamma'\delta \in Tr(s)$, since $\mathcal{S}$ is progressive. Moreover, there exist an input state s', a trace γ starting with x and followed by outputs, such that s'-after-$\gamma = s$. Thus, $\gamma\beta\gamma'\delta \in Tr(Cov(s',x))$.$\diamond$

3.2 State identifiers for IOTS

The notion of a separator for two states of a given IOTS can be considered as the generalization of the notion of separating sequence used for FSM.

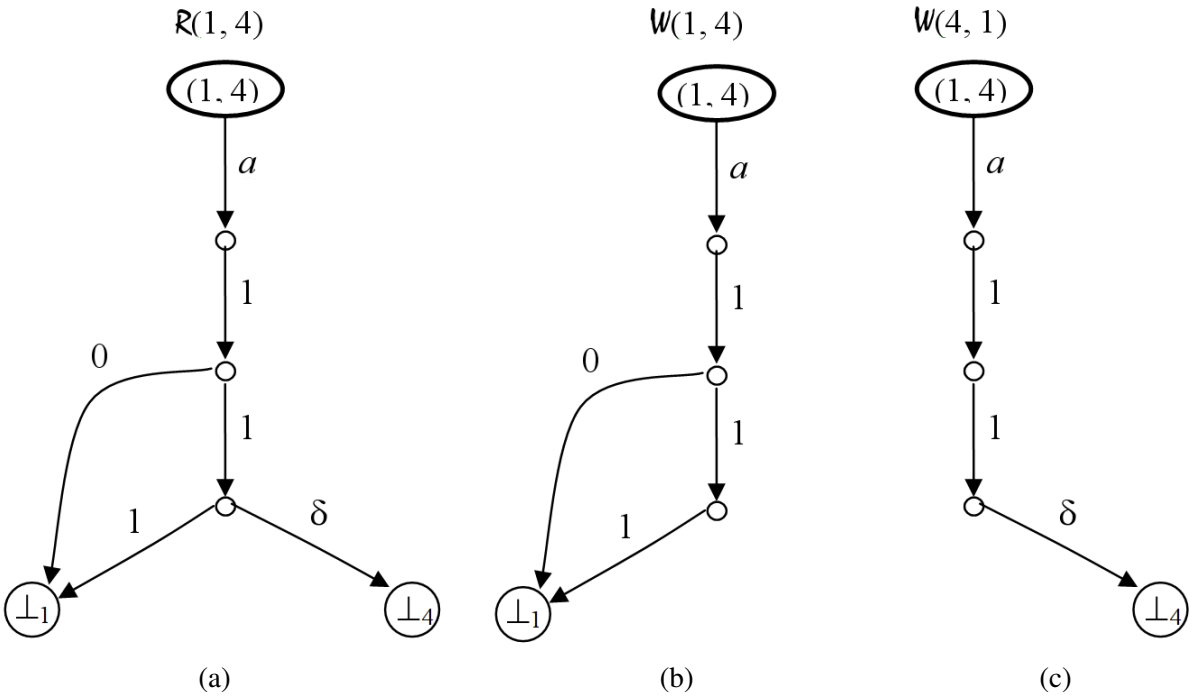
Definition 7 *Given distinguishable states s_1 and s_2 of an IOTS $\mathcal{S} \in IOTS(I,O)$, a single-input acyclic IOTS $\mathcal{R}(s_1,s_2) = (R, r_0, I, O, h_{\mathcal{R}})$ with the sink states $\perp_{s_1}$ and $\perp_{s_2}$ is a* separator *of states s_1 and s_2 if the following two conditions hold:*

- *r_0-after-$\alpha = \perp_{s_1}$ implies $\alpha \in Tr(s_1) \backslash Tr(s_2)$ and r_0-after-$\alpha = \perp_{s_2}$ implies $\alpha \in Tr(s_2) \backslash Tr(s_1)$;*
- *for each trace α of $\mathcal{R}(s_1,s_2)$ and input x defined in r_0-after-α, $out(r_0\text{-after-}\alpha x) = out(s_1\text{-after-}\alpha x) \cup out(s_2\text{-after-}\alpha x)$.*

The IOTS, obtained by removing from $\mathcal{R}(s_1,s_2)$ the sink state $\perp_{s_2}$ and all transitions leading to it, is called a distinguisher of s_1 from s_2 and is denoted by $\mathcal{W}(s_1,s_2)$.

Separator $\mathcal{R}(s_1,s_2)$ can be obtained from the intersection $\mathcal{S}/s_1 \cap \mathcal{S}/s_2 = (Q,(s_1,s_2),I,O,h_{\mathcal{S}}/s_1 \cap \mathcal{S}/s_2)$, similar to the case of FSM [10], as follows (Algorithm 2). First we determine the intersection $\mathcal{S}/s_1 \cap \mathcal{S}/s_2$ and identify the states where the two IOTSs $\mathcal{S}/s_1$ and $\mathcal{S}/s_2$ disagree on outputs. For each such state, we add transitions leading to sink states $\perp_{s_1}$ and $\perp_{s_2}$. In the final step, we determine a separator as a single-input output-preserving acyclic submachine of the obtained IOTS by removing inputs, as in Algorithm 1.

Algorithm 2 for constructing a separator for two input states.
Input: An IOTS $\mathcal{S}$ and distinguishable input states $s_1, s_2 \in S_{in}$.
Output: a separator $\mathcal{R}(s_1,s_2)$.
Construct the IOTS $\mathcal{S}/s_1 \cap \mathcal{S}/s_2 = (Q,(s_1,s_2),I,O,h_{\mathcal{S}/s_1 \cap \mathcal{S}/s_2})$
Let $Q_{dis} = \{(s,s') \in Q \mid out(s) \neq out(s')\}$
$h_{dis} = \{((s,s'),o,\perp_{s_1}) \mid (s,s') \in Q_{dis}, o \in out(s) \backslash out(s')\} \cup \{((s,s'),o,\perp_{s_2}) \mid (s,s') \in Q_{dis}, o \in out(s') \backslash out(s)\}$
$h_{\mathcal{P}} = h_{\mathcal{S}/s_1 \cap \mathcal{S}/s_2} \cup h_{dis}$
Let $\mathcal{P} = (Q \cup \{\perp_{s_1}, \perp_{s_2}\}, (s_1,s_2), I, O, h_{\mathcal{P}})$
Starting from the initial state, remove in each state all input transitions, but one, to obtain a single-input submachine with the only sink states $\perp_{s_1}$ and $\perp_{s_2}$;



(a) (b) (c)

Figure 4: (a) Separator $\mathcal{R}(1,4)$, (b) Distinguisher $\mathcal{W}(1,4)$ and (c) Distinguisher $\mathcal{W}(4,1)$.

Delete states which are unreachable from the initial state;

Return the obtained machine as a separator for the states s_1 and s_2, and stop. $\diamond$

Notice that a separator of states s_1 and s_2 is obviously a separator of s_2 and s_1, i.e., $\mathcal{R}(s_1,s_2) = \mathcal{R}(s_2,s_1)$, whereas a distinguisher of s_1 from s_2 is different from a distinguisher of s_2 from s_1, i.e., $\mathcal{W}(s_1,s_2) \neq \mathcal{W}(s_2,s_1)$. Figure 4 shows a separator $\mathcal{R}(1,4)$ obtained by Algorithm 2, as well as the corresponding distinguishers $\mathcal{W}(1,4)$ and $\mathcal{W}(4,1)$.

We consider only input-state-minimal specification IOTS, so we are interested in distinguishers of only input states. If s_1 is a stable state and s_2 is a quasi-stable state then the separator $\mathcal{R}(s_1,s_2)$ is simple; it has a transition with δ leading from the state (s_1,s_2) to state s_1 and a transition for each $o \in out(s_2)$, leading to s_2. Thus, a distinguisher of each stable state from any quasi-stable state has a single δ-transition, we call it a *quiescence distinguisher* of a stable state s, denoted $\mathcal{W}^{\delta}(s)$. It should be included into a stable state identifier of the state s.

Definition 8 *A* state identifier *of input state s, denoted $\mathcal{ID}(s)$, is a set of distinguishers $\mathcal{W}(s,s')$ for each input state s' distinguishable from s, including $\mathcal{W}^{\delta}(s)$ if state s is stable. A set of input state identifiers $\{\mathcal{ID}(s) \mid s \in S_{in}\}$, is* harmonized*, if for each pair of input states s_1 and s_2, such that both are either stable or quasi-stable states, there exists a separator $\mathcal{R}(s_1,s_2)$, such that $\mathcal{W}(s_1,s_2) \in \mathcal{ID}(s_1)$ and $\mathcal{W}(s_2,s_1) \in \mathcal{ID}(s_2)$.*

For the IOTS in Figure 1, we have that $\mathcal{ID}(1)$ includes $\mathcal{W}^{\delta}(1)$ as well as $\mathcal{W}(1,4)$ in Figure 4.

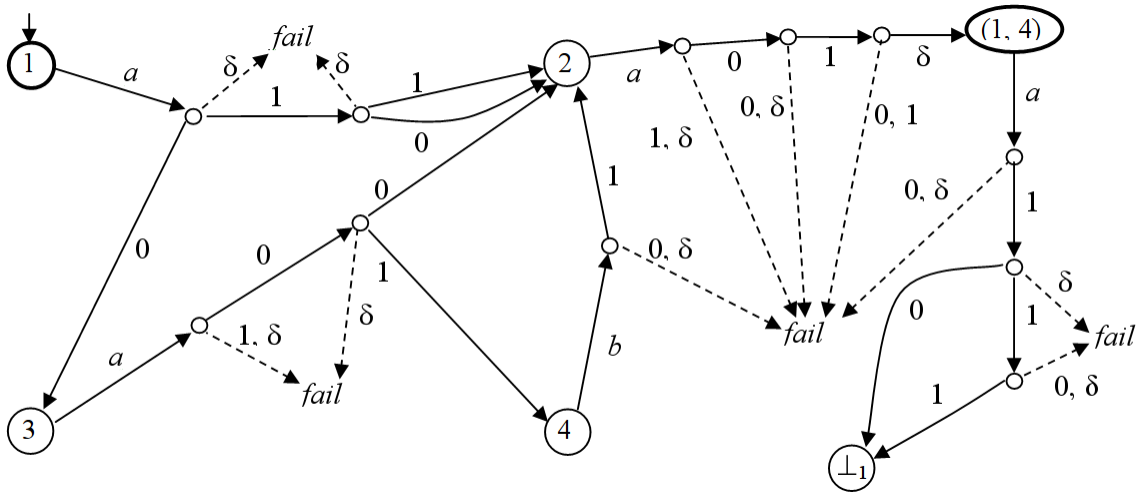


Figure 5: Test Case $TC(\mathcal{C}_2@_2Cov(2,a)@_1\mathcal{W}(1,4))$.

3.3 Complete test suite

Given the specification IOTS $\mathcal{S} = (S, s_0, I, O, h_\mathcal{S})$, $\mathcal{S} \in IOTS(I,O)$, let Z be an input state cover, V be a transition cover of $\mathcal{S}$, and $\{\mathcal{ID}(s) \mid s \in S_{in}\}$ be a set of harmonized identifiers for input states. Consider the set of IOTSs obtained by chaining each IOTS from the input state cover and transition cover with a corresponding harmonized state identifier, namely $D = \{\mathcal{T}@_s\mathcal{R} \mid s \in sink(\mathcal{T}), \mathcal{T} \in (Z \cup V), \mathcal{R} \in \mathcal{ID}(s)\}$, where $sink(\mathcal{T})$ is the set of sink states of $\mathcal{T}$. Each IOTS $\mathcal{U} \in D$ is an acyclic single-input IOTS, since it is obtained by chaining IOTSs with these properties. Moreover, it has no quasi-stable states. If the IOTS $\mathcal{U}$ happens to be also output-complete then it satisfies Definition 3 and is already a test case. The IOTSs in this set can easily be completed with the state $fail$ as follows. Given a single-input acyclic IOTS $\mathcal{U} = (U, u_0, I, O, h_\mathcal{U})$, let $TC(\mathcal{U})$ be the IOTS $(T \cup \{fail\}, u_0, I, O, h_\mathcal{U} \cup h_f)$, where $h_f = \{(s, o, fail) \mid s \in U, out(s) \neq \emptyset, o \in O \backslash out(s)\}$, which is a test case. Figure 5 shows the example of a test case, obtained by chaining the preamble $\mathcal{C}_2$, $Cov(2,a)$ with the quiescent trace $a01\delta$, and distinguisher $\mathcal{W}(1,4)$. Notice that the quiescence distinguisher $\mathcal{W}^\delta(4)$ of a stable state 4 is also used to identify this state, since the quiescent trace $a01\delta$ has it as a suffix. The $fail$ state is replicated to reduce the clutter.

Completing each IOTS in the set D, we finally obtain a test suite $TS = \{TC(\mathcal{U}) \mid \mathcal{U} \in D\}$. Consider now the subset of $IEIOTS(I,O)$ restricted by the number of input states less or equal to that of the specification IOTS $\mathcal{S}$; we denote it by $IEIOTS(I,O,k)$, where k is the number of input states in $\mathcal{S}$. We state the main result of the paper.

Theorem 1 *Given an IOTS $\mathcal{S} \in IOTS(I,O)$ with k input states, the test suite TS is a complete test suite for $\mathcal{S}$ in $IEIOTS(I,O,k)$ w.r.t.* ***ioco*** *relation.*

Before proving Theorem 1, we state some auxiliary results.

Lemma 5 *Given two IOTSs $\mathcal{P}$, $\mathcal{S} \in IOTS(I,O)$, if $\mathcal{P}$ is an initially connected submachine of $\mathcal{S}$ with the same initial state s_0, then $\mathcal{P}$* ***ioco*** *$\mathcal{S}$.*

Proof. Let α be a trace of $\mathcal{S}$. We show that $out(\mathcal{P}\text{-after-}\alpha) \subseteq out(\mathcal{S}\text{-after-}\alpha)$. Let $s = \mathcal{S}$-after-a. If $s \notin P$, where P is the set of states of $\mathcal{P}$, then $out(\mathcal{P}\text{-after-}\alpha) = \emptyset$, and the result follows. If $s \in P$, we have that $out_\mathcal{P}(s) \subseteq out_\mathcal{S}(s)$. As $s = \mathcal{P}$-after-α, the result also follows. Thus, $\mathcal{P}$ **ioco** $\mathcal{S}$.$\diamond$

Definition 9 *Given two IOTSs $\mathcal{P}$, $\mathcal{S} \in IOTS(I,O)$, $\mathcal{P} = (P, p_0, I, O, h_{\mathcal{P}})$ and $\mathcal{S} = (S, s_0, I, O, h_{\mathcal{S}})$, $\mathcal{P}$ is* input-state homeomorphic *to $\mathcal{S}$, if there exists a bijective map φ from P_{in} to S_{in} such that for every state $p \in P_{in}$, each bridge trace $\gamma \in Tr(p)$, it holds that $\varphi(p)$-after-$\gamma = \varphi(p$-after-$\gamma)$.*

$\mathcal{P}$ and $\mathcal{S}$ are input-state isomorphic*, if $\mathcal{P}$ is input-state homeomorphic to $\mathcal{S}$ and $\mathcal{S}$ is input-state homeomorphic to $\mathcal{P}$.*

Notice that for output-deterministic IOTSs, input-state isomorphic IOTSs are also input-state homeomorphic. An output-nondeterministic IOTS $\mathcal{S}$ that is input-state homeomorphic to $\mathcal{P}$ differs from $\mathcal{P}$ in state names, as well as in the set of bridge traces in some states, since it may have fewer bridge traces, while input-state isomorphic IOTSs differ just in state names.

Corollary 2 *Given two IOTSs $\mathcal{P}, \mathcal{S} \in IOTS(I,O)$, if $\mathcal{P}$ is input-state homeomorphic to $\mathcal{S}$, then $\mathcal{P}$ is input-state isomorphic to an initially connected submachine of $\mathcal{S}$ with k input states and the same initial state.*

Lemma 6 *Given an IOTS $\mathcal{S} \in IOTS(I,O)$, let $\mathcal{N} \in IEIOTS(I,O,k)$ be an IEIOTS which passes TS. Then $\mathcal{N}$ is input-state homeomorphic to $\mathcal{S}$.*

Proof. Let $\mathcal{N} \in IEIOTS(I,O,k)$, such that $\mathcal{N}$ passes TS. TS contains test cases where preambles of an input state cover are chained with harmonized identifiers to the respective states. Thus, for input states s and s', TS contains the test cases $TC(\mathcal{C}_s @_s \mathcal{W}(s,s'))$ and $TC(\mathcal{C}_{s'} @_{s'} \mathcal{W}(s',s))$. Let α be a completed trace of $\mathcal{C}_s$ and α' be a completed trace of $\mathcal{C}_{s'}$, such that $\alpha, \alpha' \in Tr(\mathcal{N})$. As $\mathcal{N}$ passes TS, no $fail$ state is reached when the distinguishers $\mathcal{W}(s,s')$ and $\mathcal{W}(s',s)$ are applied after α and α', respectively. Since no state can reach sink state in both distinguishers (see Definition 7), we have that the states $\mathcal{N}$-after-α and $\mathcal{N}$-after-α' are different, i.e., $\mathcal{N}$-after-$\alpha \neq \mathcal{N}$-after-α'. These are input states, thus, for each pair of input states of $\mathcal{S}$ there exist a pair of distinct states in $\mathcal{N}$; consequently, $\mathcal{N}$ has at least k input states. As $\mathcal{N} \in IEIOTS(I,O,k)$, $\mathcal{N}$ has exactly k input states.

Let $\mathcal{T} \in (Z \cup V)$, $t \in sink(\mathcal{T})$, $\alpha \in Tr(\mathcal{T}) \cap Tr(\mathcal{N})$, such that $\mathcal{T}$-after-$\alpha = t$, $\mathcal{N}$-after-$\alpha \in N_{in}$. Similarly, let $\mathcal{T}' \in (Z \cup V)$, $t' \in sink(\mathcal{T}')$, $\alpha' \in Tr(\mathcal{T}') \cap Tr(\mathcal{N})$, such that $\mathcal{T}'$-after-$\alpha' = t'$. Notice that α and α' are completed traces of IOTSs in the state or transition cover, which are also traces of $\mathcal{N}$. We prove that $\mathcal{S}$-after-$\alpha' = \mathcal{S}$-after-α if and only if $\mathcal{N}$-after-$\alpha' = \mathcal{N}$-after-α. Let $s = \mathcal{S}$-after-α and $s' = \mathcal{S}$-after-α'. Suppose first that $\mathcal{S}$-after-$\alpha' \neq \mathcal{S}$-after-α. Thus, TS contains $TC(\mathcal{T} @_s \mathcal{W}(s,s'))$ and $TC(\mathcal{C}_{s'} @_{s'} \mathcal{W}(s',s))$, and as $\mathcal{N}$ passes TS, no $fail$ state is reached when the distinguishers $\mathcal{W}(s,s')$ and $\mathcal{W}(s',s)$ are applied after α and α', respectively. Since no state can reach sink state in both distinguishers, we have that $\mathcal{N}$-after-$\alpha \neq \mathcal{N}$-after-α'. Suppose now that $\mathcal{S}$-after-$\alpha' = \mathcal{S}$-after-α. We prove by contradiction that $\mathcal{N}$-after-$\alpha' = \mathcal{N}$-after-α. Assume that $\mathcal{N}$-after-$\alpha' \neq \mathcal{N}$-after-α. Thus, let s'' be an input state, different from $s = \mathcal{S}$-after-α. Let $\beta \in Tr(\mathcal{C}_{s''})$, such that $\beta \in Tr(\mathcal{N})$. As TS contains $TC(\mathcal{T} @_s \mathcal{W}(s,s''))$ and $TC(\mathcal{C}_{s''} @_{s''} \mathcal{W}(s'',s))$ and $\mathcal{N}$ passes TS, we have that $\mathcal{N}$-after-$\alpha \neq \mathcal{N}$-after-β. Analogously, we can show that we have that $\mathcal{N}$-after-$\alpha' \neq \mathcal{N}$-after-β. Thus, $\mathcal{N}$-after-α is distinct from $k-1$ distinct input states of $\mathcal{N}$ and $\mathcal{N}$-after-α' is also distinct from $k-1$ distinct input states of $\mathcal{N}$. As $\mathcal{N}$-after-$\alpha' \neq \mathcal{N}$-after-α, $\mathcal{N}$ has $k+1$ states, which contradicts the fact that $\mathcal{N} \in IEIOTS(I,O,k)$ and has at most k input states. Therefore, $\mathcal{N}$-after-$\alpha' = \mathcal{N}$-after-α. Thus, let φ be a bijection from the input states N_{in} of $\mathcal{N}$ to the input states S_{in} of $\mathcal{S}$, such that for each completed trace χ of an IOTS in the state cover Z or transition cover V, which is also a trace of $\mathcal{N}$, we have that $\varphi(\mathcal{N}$-after-$\chi) = \mathcal{S}$-after-χ. Let p be an input state of $\mathcal{N}$. There exists a completed trace α of an IOTS in the input state cover Z, such that α is also a trace of $\mathcal{N}$ and $\mathcal{N}$-after-$\alpha = p$. Thus, it holds that $\varphi(\mathcal{N}$-after-$\alpha) = \varphi(p) = \mathcal{S}$-after-$\alpha$. Let $\gamma \in Tr(p)$ be a bridge trace, such that $\alpha\gamma$ is a completed trace of an IOTS in the transition cover V. Thus, it follows that $\varphi(p)$-after-$\gamma = \varphi(\mathcal{N}$-after-$\alpha)$-after-$\gamma = (\mathcal{S}$-after-$\alpha)$-after-$\gamma = \mathcal{S}$-after-$\alpha\gamma = \varphi(\mathcal{N}$-after-$\alpha\gamma) =$

$\varphi((\mathcal{N}\text{-after-}\alpha)\text{-after-}\gamma) = \varphi(p\text{-after-}\gamma)$, i.e., $\varphi(p)\text{-after-}\gamma = \varphi(p\text{-after-}\gamma)$. Therefore, we have that $\mathcal{N}$ is input-state homeomorphic to $\mathcal{S}$.$\diamond$

We can now prove Theorem 1.

Proof of Theorem 1. We first prove that TS is sound for $\mathcal{S}$ in $IEIOTS(I,O,k)$. Let $\mathcal{N} \in IEIOTS(I,O,k)$, such that $\mathcal{N}$ **ioco** $\mathcal{S}$. We have that for each test $\mathcal{U} \in TS$, $Tr_{pass}(\mathcal{U}) \subseteq Tr(\mathcal{S})$. Thus, $Tr_{pass}(\mathcal{U} \cap \mathcal{S}) = Tr_{pass}(\mathcal{U}) \cap Tr(\mathcal{S}) = Tr_{pass}(\mathcal{U})$. Since $\mathcal{N}$ **ioco** $\mathcal{S}$, we have, for each $\alpha \in Tr(\mathcal{S})$, $out(\mathcal{N}\text{-after-}\alpha) \subseteq out(\mathcal{S}\text{-after-}\alpha)$. Let $\beta \in Tr_{pass}(\mathcal{U} \cap \mathcal{N})$; hence, $\beta \in Tr_{pass}(\mathcal{U})$ and $\beta \in Tr(\mathcal{N})$. As $Tr_{pass}(\mathcal{U}) \subseteq Tr(\mathcal{S})$, we have that $\beta \in Tr(\mathcal{S})$. It follows that $Tr_{pass}(\mathcal{U} \cap \mathcal{N}) = Tr_{pass}(\mathcal{U}) \cap Tr(\mathcal{N}) \subseteq Tr_{pass}(\mathcal{U}) \cap Tr(\mathcal{S}) = Tr_{pass}(\mathcal{U} \cap \mathcal{S}) = Tr_{pass}(\mathcal{U})$. Hence, $Tr_{pass}(\mathcal{U} \cap \mathcal{N}) \subseteq Tr_{pass}(\mathcal{U})$. As a result, $\mathcal{N}$ passes each test of TS, and TS is thus sound for $\mathcal{S}$ in $IEIOTS(I,O,k)$ for the **ioco** relation.

We now prove by contradiction that TS is exhaustive for $\mathcal{S}$ in $IEIOTS(I,O,k)$. Assume that TS is not exhaustive $\mathcal{S}$ in $IEIOTS(I,O,k)$; thus, there exists $\mathcal{N} \in IEIOTS(I,O,k)$, such that $\mathcal{N}$ ~~**ioco**~~ $\mathcal{S}$ and $\mathcal{N}$ passes TS. As $\mathcal{N}$ passes TS, by Lemma 6, we have that $\mathcal{N}$ is input-state homeomorphic to $\mathcal{S}$; thus, by Corollary 2, $\mathcal{N}$ is input-state isomorphic to an initially connected submachine of $\mathcal{S}$ with k input states; hence, by Lemma 5, $\mathcal{N}$ **ioco** $\mathcal{S}$, a contradiction. We conclude then that TS is exhaustive for $\mathcal{S}$ in $IEIOTS(I,O,k)$.

Therefore, TS is complete for $\mathcal{S}$ in $IEIOTS(I,O,k)$ w.r.t. the **ioco** relation.$\diamond$

4 Concluding Remarks

In this paper, we have investigated whether it is possible to construct a finite test suite for a given IOTS specification which is complete in a predefined fault domain for the classical **ioco** relation even in the presence of input/output conflicts. Our conclusion is that it is in fact possible; however, under a number of assumptions about the implementations and the specifications. We have proposed a generation method which produces a finite test suite, which is complete for a given fault domain. The issue of conflicts between inputs and outputs is tackled by assuming that the implementation is "eager" to read inputs and thus such conflict is solved in favor of input, i.e., outputs are produced only if no input is presented to the implementation.

The proposed generation method is based on a classical FSM method. Thus, we rephrased the notions related to FSM generation methods, such as state cover, transition cover, state identifier, to the IOTS model. The method applies to IOTS that is minimal in the sense defined in the paper and each input state is reachable in any **ioco**-conforming implementation. A remarkable feature of the method is that it requires no assumption about distinguishability of output states or about their number in the specification and any implementation. Also no bound on the buffer's length in the implementation is required to generate a complete test suite.

Our future work will focus on extending the class of IOTSs for which the approach is applicable by relaxing the mentioned constraints.

Acknowledgment

The first author would like to thank Brazilian Funding Agency FAPESP for its partial financial support (Grant 12/02232-3). We would like to thank the anonymous reviewers for the suggestions that helped improving the paper.

References

[1] I. Bourdonov, A. Kossatchev & V. Kuliamin (2006): *Formal conformance testing of systems with refused inputs and forbidden actions*. Electronic Notes in Theoretical Computer Science 164(4), pp. 83–96, doi:10.1016/j.entcs.2006.09.008.

[2] T. Chow (1978): *Testing software design modeled by finite-state machines*. IEEE Transactions on Software Engineering 4(3), pp. 178–187, doi:10.1109/TSE.1978.231496.

[3] F. C. Hennie (1964): *Fault-detecting experiments for sequential circuits*. In: Proceedings of the 5th Annual Symposium on Switching Circuit Theory and Logical Design, Princeton, New Jersey, pp. 95–110, doi:10.1109/SWCT.1964.8.

[4] R. Hierons (2012): *The complexity of asynchronous model based testing*. Theor. Comput. Sci. 451, pp. 70–82, doi:10.1016/j.tcs.2012.05.038.

[5] R. Hierons (2013): *Implementation relations for testing through asynchronous channels*. Comput. J. 56(11), pp. 1305–1319, doi:10.1093/comjnl/bxs107.

[6] J. Huo & A. Petrenko (2004): *On testing partially specified iots through lossless queues*. In: Proc. Testing of Communicating Systems, pp. 76–94, doi:10.1007/978-3-540-24704-3_6.

[7] J. Huo & A. Petrenko (2009): *Transition covering tests for systems with queues*. Software Testing Verification and Reliability 19, pp. 55–83, doi:10.1002/stvr.396.

[8] C. Jard & T. Jeron (2005): *TGV: Theory, principles and algorithms: A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems*. Software Tools for Technology Transfer 7(4), pp. 297–315, doi:10.1007/s10009-004-0153-x.

[9] N. Lynch & M. R. Tuttle (1989): *An introduction to input/output automata*. CWI Quarterly 2(3), pp. 219–246.

[10] A. Petrenko & N. Yevtushenko (2011): *Adaptive testing of deterministic implementations specified by nondeterministic fsms*. In: International Conference on Testing Software and Systems, pp. 162–178, doi:10.1007/978-3-642-24580-0_12.

[11] A. Petrenko, N. Yevtushenko & J. Huo (2003): *Testing transition systems with input and output testers*. In: TestCom 2003, LNCS 2644, pp. 129–145, doi:10.1007/3-540-44830-6_11.

[12] A. Simao & A. Petrenko (2011): *Generating asynchronous test cases from test purposes*. Information & Software Technology 53(11), pp. 1252–1262, doi:10.1016/j.infsof.2011.06.006.

[13] Q. Tan & A. Petrenko (1998): *Test generation for specifications modeled by input/output automata*. In: Proceedings of the 11th International Workshop on Testing of Communicating Systems (IWTCS'98), pp. 83–99, doi:10.1007/978-0-387-35381-4_6.

[14] J. Tretmans (1996): *Test generation with inputs, outputs and repetitive quiescence*. Software Concepts and Tools 17(3), pp. 103–120.

[15] J. Tretmans (2008): *Model based testing with labelled transition systems*. In: Formal Methods and Testing, pp. 1–38, doi:10.1007/978-3-540-78917-8_1.

[16] J. Tretmans & E. Brinksma (2003): *TorX: automated model based testing*. In: First European Conference on Model-Driven Software Engineering, pp. 31–43.

[17] J. Tretmans & L. Verhaard (1992): *A queue model relating synchronous and asynchronous communication*. In: Proc. International Symposium Protocol Specification, Testing and Verification, pp. 131–145, doi:10.1016/B978-0-444-89874-6.50015-5.

[18] M. P. Vasilevskii (1973): *Failure diagnosis of automata*. Cybernetics 4, pp. 653–665, doi:10.1007/BF01068590.

[19] N. Yevtushenko & A. Petrenko (1990): *Synthesis of test experiments in some classes of automata*. Automatic Control and Computer Sciences 24(4), pp. 50–55.