Testing Asynchronous Reactive Systems: Beyond the ioco framework

Adilson Luiz Bonifacio
Computing Department, University of Londrina, Londrina, Brazil
bonifacio@uel.br

and

Arnaldo Vieira Moura
Computing Institute, University of Campinas, Campinas, Brazil
arnaldo@ic.unicamp.br

Abstract

Manual testing can be rather time consuming and prone to errors specially when testing asynchronous reactive systems. Model based testing is a well-established approach to verify reactive systems specified by input output labeled transition systems (IOLTSs). One of the challenges stemming from model based testing is verifying conformance and, also, generating test suites, primarily when completeness is a required property. In order to check whether an implementation under test is in compliance with its respective specification one resorts to some form of conformance relation that guarantees the expected behavior of the implementations, given the behavior of the specification. The **ioco** relation is an example of such a conformance relation. In this work we study another conformance relation based on formal languages. We also investigate how to generate complete test suites for IOLTS models, and discuss the complexity of the test generation mechanism under this new conformance relation. We also show that **ioco** is a special case of this new conformance relation. Further, we relate our contributions to more recent works, accommodating the restrictions of their classes of fault models as special cases, and we expose the complexity of generating any complete test suite that must satisfy their restrictions.

Keywords: Test suite generation, Complete test suites, Asynchronous systems, IOLTS

1 Introduction

Software testing has been an important part in system development processes, with the goal of improving the quality of the final products. The systematic use of formal methods and techniques has taken prominence when accuracy and critical guarantees are of paramount importance to the development process, such as when failures can cause severe damages. Testing approaches based on formal models have the added advantage that test suite generation, with proven completeness guarantees, can be effectively and precisely automated, for certain classes of specification models.

Some formalisms, such as Finite State Machines (FSMs) [1, 2, 3], capture some aspects of systems' behaviors. However, in FSM models, input actions from the environment and output actions produced by the implementations under test are strongly related, and must occur synchronously. This may limit the designers' ability to model more complex asynchronous behaviors.

This work studies aspects of a more powerful formalism where the exchange of input and output stimuli can occur asynchronously, namely, the class of Input/Output Labeled Transition Systems (IOLTSs) [4, 5]. In this context, model based testing [6, 7] has been widely used as a formal framework to verify whether an implementation under test (IUT) is in conformance to a given specification, according to a given fault model and a given conformance relation [8, 9, 4]. In particular, the **ioco** relation has been proposed as a suitable conformance relation for testing IOLTS models [5]. We propose a new notion of conformance relation for testing IOLTS models. Under this new notion, it is possible to accommodate wider classes of IOLTS models,

thus removing some of the structural restrictions imposed by other approaches. Our main contributions can be summarized as follows:

— The new notion of conformance allows for the specification of arbitrary desired, as well as undesired, behaviors that an IUT must comply to. When these specifications can be cast as regular languages, we show that a certain regular language is a complete test suite that can be used to detect the presence of all desired behaviors and the absence of any undesired behaviors in any IUT.

— In a "white-box" testing scenario, when one has access to the internal structure of the IUTs, we prove the correctness of a polynomial time algorithm that can be used for checking conformance against the new relation. If the specification model is fixed, then the algorithm runs in linear time in the number of states in the implementation.

— In a "black-box" testing environment, when the tester does not have access to the internal structure of the implementations, we show that the classical **ioco** relation [5] is a special case of this new conformance relation. We also show how to generate complete test suites that can be used to verify **ioco**-conformance under the same set of fault models considered by Tretmans [5], and for implementations with any number of states, independently of the number of states in the specification. We remark that our models and definitions in Section 5 are co-extensive with those found in [5]. Further, in this setting, we prove that 1.61^m is an asymptotic worst case lower bound on the size of any **ioco**-complete test suite, where m is the number of states of the largest implementation to be put under test. We also show that our approach attains such a lower bound.

— In a recent work, Simão and Petrenko [10] discussed how to generate complete test suites for some classes of restricted IOLTSs. We prove that our method can construct a complete test suite for such classes of models, when the same restrictions must be satisfied. Also, in [10] the complexity of the generated test suites was not studied. Here we prove an asymptotic exponential worst case lower bound for any **ioco**-complete test suite that must satisfy the same restrictions as in [10] and, further, we prove that our method also attains this lower bound.

We briefly comment on works that are more closely related to our study. See Section 7 for a more expanded view. Tretmans [5] proposed the **ioco**-conformance relation for IOLTS models, and developed the foundations of an **ioco**-based testing theory, where IUTs are treated as "black-boxes". In this testing architecture the tester is seen as an artificial environment that drives the exchange of input and output symbols with the IUT during test runs. Some restrictions must be observed by the specification, implementation and tester models, such as input-completeness and output-determinism. Simão and Petrenko [10] also described an approach to generate finite complete test suites for IOLTSs. They, however, also imposed a number of restrictions upon the specification and the implementation models in order to obtain finite complete test suites. They assumed test purposes to be single-input and also output-complete. Moreover, specifications and implementations must be input-complete, progressive, and initially-connected, so further restricting the class of IOLTS models that can be tested according to their fault model. Here, we remove many of such restrictions.

Section 2 establishes notations and preliminary results. A new notion of conformance relation is defined in Section 3. Generating complete test suites for verifying adherence to the new conformance relation, and its complexity, is described in Section 4. Section 5 visits the **ioco**-conformance relation and establishes exponential lower bounds on the size of **ioco**-complete test suites. Section 6 looks at another class of IOLTS models and show how to obtain complete test suites for this class, and discusses complexity issues when working with models of this class. Section 7 comments on related works, and Section 8 offers concluding remarks.

2 Preliminaries and Notation

In this section we define Labeled Transition Systems (LTSs) and Input/Output Labeled Transition Systems (IOLTSs). For completeness, we also include standard definitions and properties of regular languages and finite state automata (FSA). Some preliminary results associating LTSs and FSA are given.

2.1 Basic Notation

Let X and Y be sets. Then $\mathcal{P}(X) = \{Z \mid Z \subseteq X\}$ is the power set of X, and $X - Y = \{z \mid z \in X \text{ and } z \notin Y\}$ is the set difference. We let $X_Y = X \cup Y$. When no confusion can arise, we write X_y instead of $X_{\{y\}}$. If X is a finite set, the size of X will be indicated by $|X|$.

An alphabet A is any non-empty set of symbols. A word over A is any finite sequence $\sigma = x_1 \dots x_n$ where $n \geq 0$ and $x_i \in A$, for $i = 1, 2, \dots, n$. When $n = 0$ we have the empty sequence ε. The set of all finite words over A is denoted by $A^\star$. When we write $x_1 x_2 \dots x_n \in A^\star$, it is assumed that $n \geq 0$ and that $x_i \in A$, $1 \leq i \leq n$, unless noted otherwise. The length of $\alpha \in A^\star$ is indicated by $|\alpha|$. Hence, $|\varepsilon| = 0$. Let $\sigma = \sigma_1 \dots \sigma_n$ and $\rho = \rho_1 \dots \rho_m$ be words over A. The concatenation $\sigma\rho$ of σ and ρ is the word $\sigma_1 \dots \sigma_n \rho_1 \dots \rho_m$. Clearly, $|\sigma\rho| = |\sigma| + |\rho|$. A language G over A is a set $G \subseteq A^\star$. Let G_1, $G_2 \subseteq A^\star$. The product $G_1 G_2$ is $\{\sigma\rho \,|\, \sigma \in G_1, \rho \in G_2\}$, and the complement is $\overline{G_1} = A^\star - G_1$.

Definition 1 *Let A, B be alphabets. A* homomorphism *from A to B is any function $h : A \to B^\star$.*

A homomorphism $h : A \to B^\star$ can be extended to a function $\widehat{h} : A^\star \to B^\star$ where $\widehat{h}(\varepsilon) = \varepsilon$, and $\widehat{h}(a\sigma) = h(a)\widehat{h}(\sigma)$ where $a \in A$. We can further lift $\widehat{h}$ to a function $\widetilde{h} : \mathcal{P}(A^\star) \to \mathcal{P}(B^\star)$, by letting $\widetilde{h}(G) = \cup_{\sigma \in G} \widehat{h}(\sigma)$, for all $G \subseteq A^\star$. We often write h in place of $\widehat{h}$, or of $\widetilde{h}$, when no confusion can arise. When $a \in A$ is any symbol, we define the simple homomorphism $h_a : A \to (A - \{a\})^\star$ by letting $h_a(a) = \varepsilon$, and $h_a(x) = x$ when $x \neq a$. So, $h_a(\sigma)$ erases all occurrences of a in σ.

2.2 Labeled Transition Systems

A Labeled Transition System (LTS) is a formal model that is convenient to express asynchronous exchange of messages between participating entities, in the sense that outputs do not have to occur synchronously with inputs, but are generated as separate events. It consists of a set of states and a transition relation between states. Each transition is guarded by an action symbol. We first give the finite syntactic description of LTSs. A semantic structure that attributes meaning to a syntactic description will be given shortly in the sequel.

Definition 2 *A* Labeled Transition System *is a tuple $\mathcal{S} = \langle S, s_0, L, T \rangle$:*

1. *S is a finite set of* states *or* locations*;*
2. *s_0 is the* initial state*, or* initial location*;*
3. *L is a finite set of* labels*, or* actions*; $\tau \notin L$ is the* internal action symbol*;*
4. *$T \subseteq S \times L_\tau \times S$ is a set of* transitions*.*

The symbol τ is used to model any actions that are not exchanged as messages, that is, actions that cause only an internal change of states. The class of all LTSs over an alphabet L will be denoted by $\mathcal{L}(L)$.

Example 1 *For the LTS $\mathcal{S} = \langle S, s_0, L, T \rangle$, depicted in Figure 1, we have $S = \{s_0, s_1, s_2, s_3, s_4\}$ and $L = \{b, c, t\}$. Arrows indicate transitions, so that*

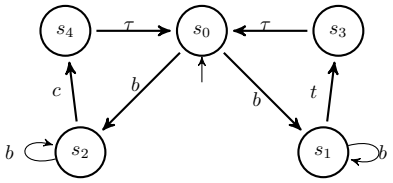


Figure 1: An LTS with 5 states and 8 transitions.

$$\begin{aligned} T = \{&(s_0, b, s_1), (s_0, b, s_2), (s_1, b, s_1), (s_1, t, s_3), \\ &(s_2, b, s_2), (s_2, c, s_4), (s_3, \tau, s_0), (s_4, \tau, s_0)\}. \end{aligned}$$

The semantics of an LTS is given by its traces, or behaviors. But first, we need the notion of paths in an LTS.

Definition 3 *Let $\mathcal{S} = \langle S, s_0, L, T \rangle$ be an LTS and $p, q \in S$. Let $\sigma = \sigma_1, \dots, \sigma_n \in L_\tau^\star$. We say that σ is:*

1. *a* path *from p to q if there are states $r_i \in S$, $0 \leq i \leq n$ and, additionally, we have $(r_{i-1}, \sigma_i, r_i) \in T$, $1 \leq i \leq n$, $r_0 = p$ and $r_n = q$;*
2. *an* observable path *from p to q if μ is a path from p to q and $\sigma = h_\tau(\mu)$.*

We say that the paths start at p and end at q.

A path σ from p to q is just a sequence of symbols that allows one to move from state p to state q. Note that a path may include transitions over the internal symbol τ. An observable path arises from any ordinary path from which internal symbols have been erased. An external observer will not see the internal transitions, as the model moves from state p to state q. It is also clear that an observable path is not necessarily a path.

Example 2 *Consider the LTS of Figure 1. The following are paths starting at s_2: ε, b, bcτ, cτbbbtτb. The following are observable paths starting at s_2: ε, b, bc, cbbbtb. There are observable paths of any length from s_2 to s_1. Note that, starting at s_2, the path cτb may lead to either s_1 or back to s_2.* □

If σ is a path from p to q we write $p \xrightarrow{\sigma} q$. We write $p \xrightarrow{\sigma}$ when there is some $q \in S$ and $p \xrightarrow{\sigma} q$; likewise, $p \to q$ means that there is some $\sigma \in L_\tau^\star$ such that $p \xrightarrow{\sigma} q$. Also $p \to$ means $p \xrightarrow{\sigma} q$ for some $q \in S$ and some $\sigma \in L_\tau^\star$. To emphasize the underlying LTS we write $p \xrightarrow[\mathcal{S}]{\sigma} q$. The symbol $\Rightarrow$ will indicate observable paths, with the same abbreviations used with the $\to$ relation. Note that $s \Rightarrow p$ if and only if $s \to p$.

Paths starting at a state p are traces of p. The semantics of an LTS are the traces of the initial state.

Definition 4 *Let $\mathcal{S} = \langle S, s_0, L, T\rangle$ be an LTS and let $p \in S$. The set of* traces *of p is $tr(p) = \{\sigma \,|\, p \xrightarrow{\sigma}\}$, and the set of* observable traces *of p is $otr(p) = \{\sigma \,|\, p \stackrel{\sigma}{\Rightarrow}\}$. The* semantics *of $\mathcal{S}$ is the set $tr(s_0)$, and the* observable semantics *of $\mathcal{S}$ is the set $otr(s_0)$.*

We may write $tr(\mathcal{S})$ for $tr(s_0)$ and $otr(\mathcal{S})$ for $otr(s_0)$. If $\mathcal{S}$ has no τ-labeled transitions, then $otr(\mathcal{S}) = tr(\mathcal{S})$.

Example 3 *Consider the LTS of Figure 1. the sequences ε, bbc, btτbcτbbb are in the semantics of $\mathcal{S}$. The observable semantics of $\mathcal{S}$ includes: ε, bcbt, bbbtbcbtb.* □

We can restrict the class of LTS models somewhat, with no loss of descriptive capability. First, we assume that all τ-moves change state, and we do away with states that are not reachable from the initial state.

Remark 1 *In Definition 2 we will always assume that $(s, \tau, s) \notin T$ and that $s_0 \to s$ holds, for any $s \in S$.*

Internal actions can facilitate the specification of formal models. For example, specifying that "after delivering money, the ATM returns to the initial state", can be done by a τ-move back to the initial state, if there is no need to exchange messages with a user. Note that Remark 1 does not prevent the occurrence of cycles labeled by τ-moves, which would lead a livelock. Sometimes, however, we may not want such behaviors, or we might want that no observable behavior leads to two distinct states. This motivates the next definition.

Definition 5 *We say that $\mathcal{S} = \langle S, s_0, L, T\rangle$ is* deterministic *if $s_0 \stackrel{\sigma}{\Rightarrow} s_1$ and $s_0 \stackrel{\sigma}{\Rightarrow} s_2$ imply $s_1 = s_2$, for all s_1, $s_2 \in S$, and all $\sigma \in L^\star$.*

The following result is immediate.

Proposition 1 *A deterministic model has no τ-moves.*

2.3 Finite State Automata

An LTS induces a finite automaton with simple properties. In particular, τ-labeled transitions correspond to nondeterminism induced by ε-moves in the automaton. For completeness, we give here the basic definitions.

Definition 6 *$\mathcal{A} = \langle S, s_0, A, \rho, F\rangle$ is a* Finite State Automaton *(FSA) where:*

1. *S is a finite set of* states*;*
2. *$s_0 \in S$ is the* initial state*;*
3. *A is a finite non-empty* alphabet*;*
4. *$\rho \subseteq S \times (A \cup \{\varepsilon\}) \times S$ is the* transition relation*; and*
5. *$F \subseteq S$ is the set of* final states*.*

A transition $(p, \varepsilon, q) \in \rho$ is called an ε-move *of $\mathcal{A}$.*

The semantics of a FSA is the set accepted words.

Definition 7 *Let $\mathcal{A} = \langle S, s_0, A, \rho, F\rangle$ be a FSA. Define the relation $\mapsto$ as*

1. $p \overset{\varepsilon}{\mapsto} p$, *and*
2. $p \overset{x\mu}{\mapsto} q$ *if* $(p, x, r) \in \rho$ *and* $r \overset{\mu}{\mapsto} q$ *with* $x \in \Sigma \cup \{\varepsilon\}$.

$L(\mathcal{A}) = \{\sigma \mid s_0 \overset{\sigma}{\mapsto} p \text{ and } p \in F\}$ *is the set* accepted by $\mathcal{A}$. *And* $G \subseteq A^\star$ *is* regular *if it is accepted by an FSA.*

Determinism requires no ε-moves and the transition relation is a function. Completeness means that from any state there is an outgoing transition on any symbol.

Definition 8 $\mathcal{A} = \langle S, s_0, A, \rho, F\rangle$ *is* deterministic *if* ρ *is a function* $S \times A \to S$, *and* $\mathcal{A}$ *is* complete *if for all* $s \in S$, $a \in A$ *we have* $(s, a, p) \in \rho$ *for some* $p \in S$.

The subset construction [11, 12] gives:

Proposition 2 *Let* $\mathcal{A}$ *be FSA. We can effectively construct a deterministic FSA* $\mathcal{B}$ *such that* $L(\mathcal{A}) = L(\mathcal{B})$.

Note that $\mathcal{B}$ may have up to 2^k states if $\mathcal{A}$ has k states.

The next constructions require simple algorithms.

Proposition 3 *Let* $\mathcal{A}$ *and* $\mathcal{B}$ *be FSAs. We can easily obtain and FSA* $\mathcal{C}$ *such that*

1. $L(\mathcal{C}) = L(A) \cup L(\mathcal{B})$, $L(\mathcal{C}) = L(\mathcal{A}) \cap L(\mathcal{B})$, *or* $L(\mathcal{C}) = \overline{L(\mathcal{A})}$.
2. $L(\mathcal{C}) = L(\mathcal{A})$ *and* $\mathcal{C}$ *is complete.*

Also, if $\mathcal{A}$ *and* $\mathcal{B}$ *are deterministic, then so is* $\mathcal{C}$.

Proof See [11].

We can convert τ-moves of an LTS into ε-moves of a FSA, and turn all states of the LTS into final states of the FSA, and vice-versa. Then algorithmic constructions for FSAs can be used to transform the LTSs.

Definition 9 *We have the following two associations:*

1. *Let* $\mathcal{S} = \langle S, s_0, L, T\rangle$ *be an LTS. The* induced *FSA is* $\mathcal{A}_\mathcal{S} = \langle S, s_0, L, \rho, S\rangle$ *where*
   $(p, \ell, q) \in \rho$ *if and only if* $(p, \ell, q) \in T$,
   $(p, \varepsilon, q) \in \rho$ *if and only if* $(p, \tau, q) \in T$.
2. *Let* $\mathcal{A} = \langle S, s_0, A, \rho, S\rangle$ *be a FSA. The* induced *LTS is* $\mathcal{S}_\mathcal{A} = \langle S, s_0, A, T\rangle$ *where*
   $(p, a, q) \in T$ *if and only if* $(p, a, q) \in \rho$,
   $(p, \tau, q) \in T$ *if and only if* $(p, \varepsilon, q) \in \rho$.

An easy induction establishes the following

Proposition 4 *The associations in Definition 9 give* $otr(\mathcal{S}) = L(\mathcal{A}_\mathcal{S})$ *and* $otr(\mathcal{S}_\mathcal{A}) = L(\mathcal{A})$.

If $\mathcal{S}$ is an LTS, Definition 9(1) gives a FSA $\mathcal{A}$ where all states are final and $L(\mathcal{A}) = otr(\mathcal{S})$. Using Proposition 2 we can get a deterministic FSA $\mathcal{B}$ with $L(\mathcal{B}) = L(\mathcal{A})$. The subset construction [11, 12] says that all states in $\mathcal{B}$ are final, except possibly for the state representing the empty set. Removing this state we get a FSA $\mathcal{C}$ in which all states are final and such that $L(\mathcal{C}) = L(\mathcal{B})$. Now we can use Definition 9(2) and get a LTS $\mathcal{T}$ with $otr(\mathcal{T}) = L(\mathcal{C})$. Moreover, since $\mathcal{B}$ was deterministic, it is easy to see that $\mathcal{T}$ is also deterministic.

Proposition 5 *Let* $\mathcal{S}$ *be an LTS. We can construct a deterministic LTS* $\mathcal{T}$ *such that* $otr(\mathcal{S}) = tr(\mathcal{T}) = otr(\mathcal{T})$.

Note that in the worst case $\mathcal{T}$ can grow exponentially.

2.4 Input Output Labeled Transition Systems

Sometimes we wish to say that symbols are "received" from the environment as inputs, while others are "sent back" as outputs. An Input/Output Labeled Transition System (IOLTS) allows for this.

Definition 10 *An Input Output Labeled Transition System (IOLTS) is a tuple* $\mathcal{I} = \langle S, s_0, L_I, L_U, T\rangle$, *where:*

1. L_I *is a finite non-empty set of* input actions*;*
2. L_U *is a finite non-empty set of* output actions*;*
3. $L_I \cap L_U = \emptyset$, *and* $L = L_I \cup L_U$ *is the set of* actions*;*

4. $\mathcal{S}_{\mathcal{I}} = \langle S, s_0, L, T\rangle$ *is the* underlying LTS *of* $\mathcal{I}$.

$\mathcal{IO}(L_I, L_U)$ is the class of all IOLTSs with input alphabet L_I and output alphabet L_U. Other works impose additional restrictions to the basic IOLTS model, but we do not need any restrictions at this point. We look at more restricted variations in Sections 5 and 6.

Several notions involving IOLTSs will be defined by a direct reference to their underlying LTSs.

Definition 11 *The semantics of an IOLTS* $\mathcal{I}$ *is the set* $otr(\mathcal{I}) = otr(\mathcal{S}_{\mathcal{I}})$.

When $\mathcal{I}$ is an IOLTS the notation $\underset{\mathcal{I}}{\rightarrow}$ and $\underset{\mathcal{I}}{\Rightarrow}$ are to be understood as $\underset{\mathcal{S}}{\rightarrow}$ and $\underset{\mathcal{S}}{\Rightarrow}$, respectively, where $\mathcal{S}$ is the underlying LTS for $\mathcal{I}$. IOLTSs generalize the simpler formalism of Mealy machines [13], where communication is synchronous. In an IOLTS, inputs and outputs happen asynchronously, which facilitates the specification of more complex behaviors, like in reactive systems.

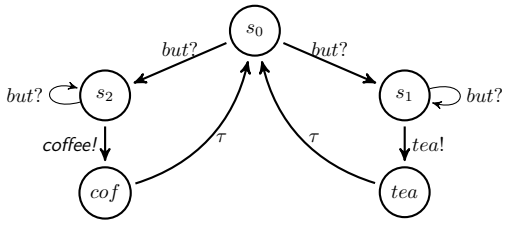


Figure 2: An IOTS for a drink dispensing machine.

Example 4 *Figure 2 was adapted from [5]. It describes a strange coffee machine. When the user hits the start button, indicated by the input label* but?*, the machine chooses to go either to* s_1 *or to* s_2*. If it goes to* s_1*, no matter how many extra times* but? *is hit the loop at* s_1 *keeps the machine at* s_1*; then, asynchronously, the machine dispenses a cup of tea, signaled by the output label* tea!*, reaching state* tea*. Next, the machine performs an internal action, and returns to the start state. The left branch indicates a similar behavior, but now it dispenses a cup of coffee, and by another internal action returns it to the start state.* □

3 Conformance testing

We define a new and generalized notion of conformance, allowing for sets of desired and undesired behaviors in IOLTS models. We study the relationship of this notion to **ioco**-conformance [5], and show that the latter is a special case of this new conformance relation.

Remark 2 *We deal with the notion of quiescent states in Section 5. For now we note that the treatment of quiescent states will require the addition of a new symbol* δ *to the output alphabet, and some* δ*-transition to the set of transitions. Since the results obtained in this section are valid for* any *LTS model, they will remain valid for those variations that treat quiescent states explicitly.*

3.1 The New Conformance Relation

We consider a language D, the set of "desirable" behaviors, and a language F, the set of "undesirable", behaviors. We say that an implementation $\mathcal{I}$ *conforms* to a specification $\mathcal{S}$ according to (D, F) if no undesirable behavior observable in $\mathcal{I}$ is specified in $\mathcal{S}$, and all desirable behaviors observable in $\mathcal{I}$ are specified in $\mathcal{S}$.

Definition 12 *Let* $D, F \subseteq L^\star$, $\mathcal{S}$ *and* $\mathcal{I}$ *be LTSs over* L*. Then* $\mathcal{I}$ (D, F)-conforms to $\mathcal{S}$*, written* $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$*, if and only if* $\sigma \in otr(\mathcal{I}) \cap F$ *gives* $\sigma \notin otr(\mathcal{S})$*, and* $\sigma \in otr(\mathcal{I}) \cap D$ *gives* $\sigma \in otr(\mathcal{S})$.

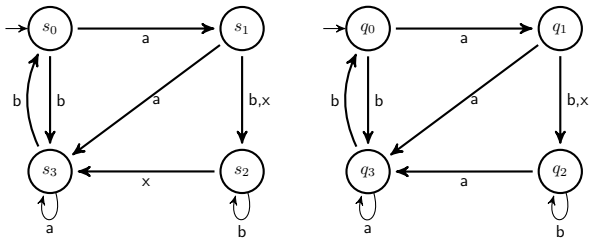
For an equivalent way of expressing these conditions write $\overline{otr}(\mathcal{S})$ for the complement of $otr(\mathcal{S})$.

Proposition 6 *Let* $\mathcal{S}$ *and* $\mathcal{I}$ *be LTSs over* L*, and* $D, F \subseteq L^\star$*. Then* $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ *if and only if*

$$otr(\mathcal{I}) \cap \left[(D \cap \overline{otr}(\mathcal{S})) \cup (F \cap otr(\mathcal{S}))\right] = \emptyset.$$

Example 5 *Let* $\mathcal{S}$ *be as in Figure 3(a) and* $\mathcal{I}$ *as shown in Figure 3(b). Let* $D = (a+b)^\star ax$ *and* $F = ba^\star b$*. We want to check if* $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ *holds.*

Since $baab \in F \cap otr(\mathcal{S})$*we get* $F \cap otr(\mathcal{S}) \neq \emptyset$*. Because* $baab \in otr(\mathcal{I})$*, we conclude that* $\mathbf{conf}_{D,F}\,\mathcal{S}$ *does not hold. Alternatively, it is easy to see that* $ababax \in \overline{otr}(\mathcal{S})$*, and clearly* $ababax \in D$*, so that*



(a) An LTS specification $\mathcal{S}$. (b) An LTS IUT $\mathcal{I}$.

Figure 3: LTS models in the new conformance relation

ababax $\in D \cap \overline{otr}(\mathcal{S})$. *From Figure 3(b), by an easy inspection* $ababax \in otr(\mathcal{I})$. *So,* $otr(\mathcal{I}) \cap D \cap \overline{otr}(\mathcal{S}) \neq \emptyset$ *and again* $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ *does not hold. In this case, either* F *or* D *is enough to guarantee non-conformance.*

Now take $F = abb^\star x$ *and* $D = aaa^\star b(bb)^\star ax$. *By inspection, we get* $\sigma x \notin otr(\mathcal{I})$ *for all* $\sigma \in abb^\star$, *so that* $otr(\mathcal{I}) \cap F \cap otr(\mathcal{S}) = \emptyset$. *Moreover, for any* $\sigma \in D$ *we get* $s_0 \stackrel{\sigma}{\Rightarrow} s_2$ *so that* $D \subseteq otr(\mathcal{S})$. *Hence,* $D \cap \overline{otr}(\mathcal{S}) = \emptyset$. *Therefore,* $otr(\mathcal{I}) \cap \big[(D \cap \overline{otr}(\mathcal{S})) \cup (F \cap otr(\mathcal{S}))\big] = \emptyset$, *and we see that* $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ *holds.*

In Subsection 3.2 we will use this example to relate the new relation to **ioco** *relation [5].* □

By varying D and F, the $\mathbf{conf}_{D,F}$ relation can accommodate several different notions of conformance:

1. Any observable behavior of $\mathcal{I}$ must rest specified in $\mathcal{S}$. Let $D = L^\star$ and $F = \emptyset$. We get $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ if and only if $otr(\mathcal{I}) \subseteq otr(\mathcal{S})$.

2. Let C and E be disjoint subsets of locations of $\mathcal{I}$. The set of allowed behaviors, H_C, must lead to a location in C, and the set of forbidden behaviors, H_E, lead to a location in E. Then, $\mathcal{I}\,\mathbf{conf}_{H_C,H_E}\,\mathcal{S}$ if and only if any allowed behavior of $\mathcal{I}$ is also specified in $\mathcal{S}$ and no forbidden behavior of $\mathcal{I}$ is specified in $\mathcal{S}$.

3. Desirable behaviors must end in $H \subseteq L$, undesirable behaviors occurring in the IUT are irrelevant and need not be checked for conformance. Let $F = \emptyset$, $D = L^\star H$.

Clearly, D and F are regular in all cases listed above.

In general, if $\mathcal{I}$ conforms to $\mathcal{S}$ according to (D, F), then we can assume D and F to be finite languages.

Corollary 1 *Let* $\mathcal{S}$ *and* $\mathcal{I}$ *be LTSs over* L, *and let* $D, F \subseteq L^\star$. *Then, there are finite sets* $D' \subseteq D$ *and* $F' \subseteq F$ *such that* $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ *if and only if* $\mathcal{I}\,\mathbf{conf}_{D',F'}\,\mathcal{S}$.

Proof If $D \cap [otr(\mathcal{I}) \cap \overline{otr}(\mathcal{S})] \neq \emptyset$ then let $D' = \{x\}$ where $x \in D \cap [otr(\mathcal{I}) \cap \overline{otr}(\mathcal{S})]$, else let $D' = \emptyset$. Also, if $F \cap [otr(\mathcal{I}) \cap otr(\mathcal{S})] \neq \emptyset$ then let $F' = \{y\}$ where $y \in F \cap [otr(\mathcal{I}) \cap otr(\mathcal{S})]$, else let $F' = \emptyset$. Then use Proposition 6. □

3.2 The ioco Conformance Relation

The **ioco**-conformance relation [5] essentially requires that if σ is any observable trace of both an implementation $\mathcal{I}$ and of a given specification $\mathcal{S}$, then if σ leads $\mathcal{I}$ to a location from which it can move on a symbol ℓ, then σ must also lead $\mathcal{S}$ to a location from which it can move on ℓ. This motivates the following definitions.

Definition 13 ([5]) *Let* $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle$, $\mathcal{I} = \langle Q, q_0, L_I, L_U, R\rangle$, *and* $L = L_I \cup L_U$.

1. *Define* $\mathbf{out}(V) = \bigcup_{s \in V} \{\ell \in L_U \mid s \stackrel{\ell}{\Rightarrow}\}$, *all* $V \subseteq S$.

2. *Define* s **after** $\sigma = \{q \mid s \stackrel{\sigma}{\Rightarrow} q\}$, *all* $s \in S$, $\sigma \in L^\star$.

3. *Define* $\mathcal{I}$ **ioco** $\mathcal{S}$ *if and only if* $\mathbf{out}(q_0$ **after** $\sigma) \subseteq \mathbf{out}(s_0$ **after** $\sigma)$, *all* $\sigma \in otr(\mathcal{S})$.

We may write $\mathbf{out}(s)$ instead of $\mathbf{out}(\{s\})$.

Example 6 *Let* $\mathcal{I} = \langle S, s_0, L_I, L_U, T\rangle$ *as in Figure 2 where* $L_I = \{but?\}$ *and* $L_U = \{coffee!, tea!\}$. *We then see that* $\mathbf{out}(\{s_1\}) = \{but?, tea!\}$, *and* $\mathbf{out}(tea) = \{but?\}$. *Also,* s_0 **after** $but?tea! = \{tea, s_0\}$ *and* cof **after** $but? = \{s_1, s_2\}$. □

Definition 12 then subsumes **ioco**-conformance [5].

Lemma 1 *Let $\mathcal{S}, \mathcal{I} \in \mathcal{IO}(L_I, L_U)$. Then $D = otr(\mathcal{S})L_U$ is regular and $\mathcal{I}$* **ioco** *$\mathcal{S}$ if and only if $\mathcal{I}$* **conf**$_{D,\emptyset}$ *$\mathcal{S}$.*

Proof From Definition 9 we see that $otr(\mathcal{S})$ is regular, and so D is also regular.

Assume $\mathcal{I}$ **conf**$_{D,\emptyset}$ $\mathcal{S}$. From Definition 12 it is clear that $\mathcal{I}$ **conf**$_{D,\emptyset}$ $\mathcal{S}$ is equivalent to $otr(\mathcal{I}) \cap D \subseteq otr(\mathcal{S})$. In order to prove that $\mathcal{I}$ **ioco** $\mathcal{S}$, let $\sigma \in otr(\mathcal{S})$ and let $\ell \in$ **out**$(q_0$ **after** $\sigma)$. We must show that $\ell \in$ **out**$(s_0$ **after** $\sigma)$. Because $\ell \in$ **out**$(q_0$ **after** $\sigma)$ we get $\sigma, \sigma\ell \in otr(\mathcal{I})$. Since $\ell \in L_U$, we get $\sigma\ell \in otr(\mathcal{S})L_U$ and so $\sigma\ell \in D$. Hence, $\sigma\ell \in otr(\mathcal{I}) \cap D$, and then $\sigma\ell \in otr(\mathcal{S})$. So, $\ell \in$ **out**$(s_0$ **after** $\sigma)$, as desired.

Next, assume that $\mathcal{I}$ **ioco** $\mathcal{S}$ and we want to show that $\mathcal{I}$ **conf**$_{D,\emptyset}$ $\mathcal{S}$. Since $otr(\mathcal{I}) \cap \emptyset \cap otr(\mathcal{S}) = \emptyset$, the first condition of Definition 12 holds. In order to show that $otr(\mathcal{I}) \cap D \subseteq otr(\mathcal{S})$, let $\sigma \in otr(\mathcal{I}) \cap D$, so that $\sigma = \alpha\ell$ with $\ell \in L_U$ and $\alpha \in otr(\mathcal{S})$. Then $\sigma \in otr(\mathcal{I})$ gives $\alpha\ell \in otr(\mathcal{I})$, and so $\alpha \in otr(\mathcal{I})$. Hence, $\ell \in$ **out**$(q_0$ **after** $\alpha)$. Since $\mathcal{I}$ **ioco** $\mathcal{S}$ and $\alpha \in otr(\mathcal{S})$, we get $\ell \in$ **out**$(s_0$ **after** $\alpha)$, and so $\alpha\ell \in otr(\mathcal{S})$. Hence, $\sigma \in otr(\mathcal{S})$. Thus, we have $otr(\mathcal{I}) \cap D \subseteq otr(\mathcal{S})$. □

The next example illustrates Lemma 1.

Example 7 *Let $\mathcal{S}$ be as in Figure 3(a), and $\mathcal{I}$ as in Figure 3(b) with the extra transition $q_3 \xrightarrow{x} q_0$. Recall that $L_I = \{a, b\}$, $L_U = \{x\}$.*

From Figure 3(a) and the new $\mathcal{I}$ it is apparent that $s_0 \stackrel{aa}{\Rightarrow} s_3$ and also $q_0 \stackrel{aa}{\Rightarrow} q_3$. Also, $x \in$ **out**(q_3)*, but $x \notin$* **out**(s_3)*. So, by Definition 13, $\mathcal{I}$* **ioco** *$\mathcal{S}$ does not hold. Now take $\sigma = aax$. Since $aa \in otr(\mathcal{S})$, we get $\sigma \in otr(\mathcal{S})L_U = D$. Also, $\sigma \in otr(\mathcal{I})$ and $\sigma \notin otr(\mathcal{S})$, so that $\sigma \in otr(\mathcal{I}) \cap D \cap \overline{otr}(\mathcal{S})$. From Proposition 6, we see that $\mathcal{I}$* **conf**$_{D,\emptyset}$ *$\mathcal{S}$ does not hold, as expected.* □

The next example shows that the new conformance, **conf**$_{D,F}$, is able to capture non-conformance situations where the **ioco** relation would yield positive verdicts.

Example 8 *Let $\mathcal{S}$ as in Figure 3(a) and $\mathcal{I}$ as in Figure 3(b). Let $L_I = \{a, b\}$, $L_U = \{x\}$.*

From Figures 3(a) and 3(b) we see that there is no $\sigma \in (L_I \cup L_U)^\star$, $s \in S$ and $q \in Q$ such that $s_0 \stackrel{\sigma}{\Rightarrow} s$ and $q_0 \stackrel{\sigma}{\Rightarrow} q$ with $x \in$ **out**(q)*, but $x \notin$* **out**(s)*. By Definition 13, $\mathcal{I}$* **ioco** *$\mathcal{S}$ holds.*

Now let $F = \emptyset$, $D = (a + b)^\star ax$, and take $\sigma = ababax \in D$. We see that $\sigma \in \overline{otr}(\mathcal{S})$ and $\sigma \in otr(\mathcal{I})$. So that $\sigma \in otr(\mathcal{I}) \cap D \cap \overline{otr}(\mathcal{S})$. From Proposition 6, $\mathcal{I}$ **conf**$_{D,\emptyset}$ *$\mathcal{S}$ does not hold, whereas $\mathcal{I}$* **ioco** *$\mathcal{S}$ would always hold.* □

We know that **ioco**-conformance is equivalent do $\mathcal{I}$ **conf**$_{D,F}$ $\mathcal{S}$ when $F = \emptyset$. By taking $F \neq \emptyset$, however, gives the test designer even more freedom to check whether some behaviors that occur in the specification are, or are not, also represented in the implementation. In particular, when $D = \emptyset$ and a verdict of conformance is obtained, the behaviors specified in F are not present, whereas a verdict of non-conformance would say that some behavior of F is present in the implementation.

Lemma 1 and Corollary 1 lead to the next result.

Corollary 2 *Let $\mathcal{S}, \mathcal{I} \in \mathcal{IO}(L_I, L_U)$. Then $\mathcal{I}$* **ioco** *$\mathcal{S}$ if and only if $\mathcal{I}$* **conf**$_{D,\emptyset}$ *$\mathcal{S}$ for some finite $D \subseteq (L_I \cup L_U)^\star$.*

From Lemma 1 and Proposition 6 we get a new **ioco** characterization.

Corollary 3 *$\mathcal{I}$* **ioco** *$\mathcal{S}$ if and only if $otr(\mathcal{I}) \cap T = \emptyset$, where $T = \overline{otr}(\mathcal{S}) \cap \big[otr(\mathcal{S})L_U\big]$, all $\mathcal{S}, \mathcal{I} \in \mathcal{IO}(L_I, L_U)$.*

4 Test Generation for IOLTS models

We show how to generate complete test suites for checking conformance of IOLTS models according to the new relation defined in Section 3, and for arbitrary sets of behaviors D and F. For deterministic IOLTS models, the algorithm has time complexity linear on the number of states of the implementation, regardless of the specification. Recall Remark 2.

4.1 Complete Test Suite Generation

A test suite is a set of words over the IOLTS alphabet.

Definition 14 *Let L be a set of symbols. A test suite is any set $T \subseteq L^\star$. Each $\sigma \in T$ is called a* test case.

Loosely speaking, an IUT $\mathcal{I}$ conforms to a specification $\mathcal{S}$ when behaviors of $\mathcal{I}$ do not deviate from those specified by $\mathcal{S}$. Test suites are designed to flag any such deviations, or guarantee that none is possible.

Definition 15 *$\mathcal{I} \in \mathcal{L}(L)$* adheres *to $T \subseteq L^\star$ if $T \cap otr(\mathcal{I}) = \emptyset$, and $\mathcal{I} \in \mathcal{IO}(L_I, L_U)$ adheres to T if $\mathcal{S}_\mathcal{I}$ does.*

Given a pair of languages (D, F) and a specification $\mathcal{S}$, we want to generate test suites T that are sound, that is, if $\mathcal{I}$ adheres to T then $\mathcal{I}$ also (D, F)-conforms to $\mathcal{S}$. The converse is also desirable, that is, when $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ then adherence of $\mathcal{I}$ to T should also be guaranteed.

Definition 16 *Let $\mathcal{S} \in \mathcal{IO}(L_I, L_U)$ and $T \subseteq L^\star$. Also, let $D, F \subseteq L^\star$. We say that:*

1. *T is* sound *for $\mathcal{S}$ and (D, F) if, for all $\mathcal{I} \in \mathcal{IO}(L_I, L_U)$, $\mathcal{I}$ adheres to T implies $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$.*
2. *T is* exhaustive *for $\mathcal{S}$ and (D, F) if, for all $\mathcal{I} \in \mathcal{IO}(L_I, L_U)$, $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ implies $\mathcal{I}$ adheres to T.*
3. *T is* complete *for $\mathcal{S}$ and (D, F) if it is both sound and exhaustive for $\mathcal{S}$ and (D, F).*

We view adherence as a syntactic notion, as it is verified just by the emptiness of a set intersection. On the other hand, conformance is closer to a semantic notion, since it is based on output events generated by the intrinsic behavior of the IUT. So, here, we preferred to use soundness in the tradition in mathematical logic, in the sense that once a sentence is (syntactically) proven, it is guaranteed to be (semantically) true. On the other direction, also as in mathematical logic, when every (semantically) true sentence has a (syntactic) proof, the theory is said to be complete. Here instead, we used the term exhaustive to indicate that if an IUT (semantically) conforms to a specification, then it must (syntactically) adhere to the test suite, reserving the term complete to indicate situations when both soundness and exhaustiveness are guaranteed to be present. Our notions of soundness and exhaustiveness might have their meaning reversed when compared to other definitions in the literature. But, of course, when both soundness and exhaustiveness are guaranteed, as is the case with our results, this reversal of notation is innocuous.

We can always have complete test suites. Furthermore, in a sense they are also unique.

Lemma 2 *Let $\mathcal{S} \in \mathcal{IO}(L_I, L_U)$, and D, $F \subseteq L^\star$. Then, $T = \left[(D \cap \overline{otr}(\mathcal{S})) \cup (F \cap otr(\mathcal{S}))\right]$ is the only complete test suite for $\mathcal{S}$ and (D, F).*

Proof That T is complete follows from Proposition 6. Now let Z be complete for $\mathcal{S}$ and (D, F), with $Z \neq T$. Let $\mathcal{I} \in \mathcal{L}(L)$ be arbitrary. Since T and Z are complete we have $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ if and only if $otr(\mathcal{I}) \cap T = \emptyset$ if and only if $otr(\mathcal{I}) \cap Z = \emptyset$. By symmetry, since $Z \neq T$, we get some $\sigma \in T \cap \overline{Z}$. It is simple to construct $\mathcal{I} \in \mathcal{L}(L)$ with $\sigma \in otr(\mathcal{I})$, leading to a contradiction. □

Because we have no control over the size of the witness $\sigma \in otr(\mathcal{I})$, it was crucial that we had no restrictions on the size of $\mathcal{I}$. This indicates that the size of the implementations will affect the complexity of the test suites can verify (D, F)-conformance. We investigate the complexity of complete test suites in the next subsection.

Lemma 2 says that $T = \left[(D \cap \overline{otr}(\mathcal{S})) \cup (F \cap otr(\mathcal{S}))\right]$ is complete for $\mathcal{S}$ and (D, F). So checking if $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ is equivalent to checking if $otr(\mathcal{I}) \cap T = \emptyset$. If D and F are regular, that is $D = L(\mathcal{A}_D)$ and $F = L(\mathcal{A}_F)$ for some FSAs $\mathcal{A}_D$ and $\mathcal{A}_F$, then using Propositions 4 and 3 we can construct a FSA $\mathcal{A}_T$ such that $T = L(\mathcal{A}_T)$. Further, by the same propositions we know that $otr(\mathcal{I}) \cap T$ is also regular. Moreover, if $\mathcal{I}$ is a "white-box", that is, we have access to its the syntactic description, we can construct the FSA $\mathcal{A}_\mathcal{I}$ such that $otr(\mathcal{I}) = L(\mathcal{A}_\mathcal{I})$ and so, using Proposition 3 we can obtain a FSA $\mathcal{A}$ such that $L(\mathcal{A}) = otr(\mathcal{I}) \cap T$. Now, a simple breadth-first algorithm can check if $L(\mathcal{A}) = \emptyset$, so that we can effectively decide if $\mathcal{I}$ (D, F)-conforms to $\mathcal{S}$. Proposition 7, in the sequel, details the time complexity of such an algorithm.

4.2 On the Complexity of Test Suites

An important issue is the size of test suites. If $\mathcal{S}$ has n states and t transitions, then $n - 1 \leq t \leq n^2$, and it is reasonable to take t as the size of $\mathcal{S}$.

Let $D, F \subseteq L^\star$, and $\mathcal{S} = \langle S, s_0, L, T\rangle$ be deterministic with n_S states. Lemma 2 says that $T = \left[(D \cap \overline{otr}(\mathcal{S})) \cup (F \cap otr(\mathcal{S}))\right]$ is complete for $\mathcal{S}$ and (D, F). Assume that D and F are regular with $L(\mathcal{A}_D) = D$ and $L(\mathcal{A}_F) = F$ for deterministic FSAs $\mathcal{A}_D$ and $\mathcal{A}_F$ with n_D and n_F states, respectively. By Proposition 4 we can construct a deterministic FSA $\mathcal{A}_1$ with $n_S + 1$ states and $L(\mathcal{A}_1) = otr(\mathcal{S})$. The proof of Proposition 3 shows that we can get a deterministic FSA $\mathcal{A}_2$ with at most $(n_S + 1)n_F$ states and such that $L(\mathcal{A}_2) = F \cap otr(\mathcal{S})$. Reversing the set of final states of $\mathcal{A}_1$ we get a deterministic FSA $\mathcal{B}_1$ with $L(\mathcal{B}_1) = \overline{otr}(\mathcal{S})$. From Proposition 3 we get a deterministic FSA $\mathcal{B}_2$ with $(n_S + 1)n_D$ states and $L(\mathcal{B}_2) = L(\mathcal{A}_D) \cap L(\mathcal{B}_1) = D \cap \overline{otr}(\mathcal{S})$. Again, Proposition 3 yields a FSA $\mathcal{C}$ with $(n_S + 1)^2 n_D n_F$ states and such that $L(\mathcal{C}) = L(\mathcal{A}_2) \cup L(\mathcal{B}_2) = T$.

Proposition 7 *Let $\mathcal{S} \in \mathcal{IO}(L_I, L_U)$ be deterministic with n_S states. Let $L(\mathcal{A}_D) = D$, $L(\mathcal{A}_F) = F$ where $\mathcal{A}_D$, $\mathcal{A}_F$ are deterministic FSAs with n_D, n_F states, respectively. We can construct a deterministic FSA $\mathcal{T}$ with $(n_S + 1)^2 n_D n_F$ states, and $L(\mathcal{T})$ a complete test suite for $\mathcal{S}$ and (D, F). Also, if $\mathcal{I} \in \mathcal{IO}(L_I, L_U)$ is deterministic with n_I states, there is a $\mathcal{O}(n_S^2 n_I n_D n_F n_L)$ algorithm that verifies $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$, where $n_L = |L_I \cup L_U|$.*

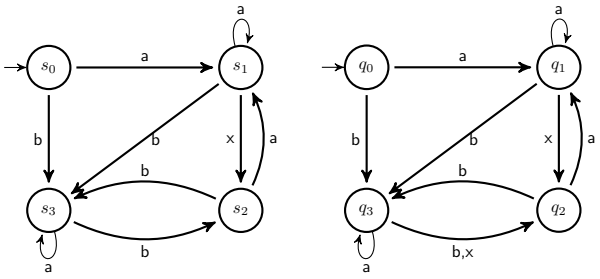
Proof The preceding discussion and Lemma 2 yield $\mathcal{T}$. A breadth-first search over $\mathcal{T}$ gives the algorithm. □

We can state a similar result for checking **ioco**-conformance when a description of the IUT is available.

Theorem 9 *Let $\mathcal{S}, \mathcal{I} \in \mathcal{IO}(L_I, L_U)$ deterministic with n_S and n_I states, respectively, and $n_L = |L_I \cup L_U|$. There is a $\mathcal{O}(n_S n_I n_L)$ algorithm to check $\mathcal{I}$ **ioco** $\mathcal{S}$.*

Proof Let $\mathcal{A}$ be the FSA with n_S and $L(\mathcal{A}) = otr(\mathcal{S})$ given by Proposition 4. Starting with $\mathcal{A}$ we construct a deterministic FSA $\mathcal{B}$ with $n_S + 2$ as follows. Let s_I and s_U be new states. For any state s of $\mathcal{A}$ and any symbol $\ell \in L_I \cup L_U$: if there is no transition (s, ℓ, p) in $\mathcal{A}$, for any state p of $\mathcal{A}$, we add to $\mathcal{B}$ the transition (s, ℓ, s_I) when $\ell \in L_I$, and the transition (s, ℓ, s_U) when $\ell \in L_U$. Finally, we turn $\mathcal{B}$ into a deterministic FSA by adding the transitions (s_U, ℓ, s_I) and (s_I, ℓ, s_I) for all $\ell \in L_I \cup L_U$. Let the initial state of $\mathcal{B}$ to be as in $\mathcal{A}$, and s_U be the only final state of $\mathcal{B}$. It is easy to see that $L(\mathcal{B}) = \overline{otr}(\mathcal{S}) \cap \big[otr(\mathcal{S})L_U\big]$. Use Proposition 4 again to get a deterministic FSA $\mathcal{C}$ with $n_I + 1$ states and $L(\mathcal{C}) = otr(\mathcal{I})$. Proposition 3 gives a deterministic FSA $\mathcal{T}$ with $(n_S + 2)(n_I + 1)$ states and $L(\mathcal{T}) = otr(\mathcal{I}) \cap \overline{otr}(\mathcal{S}) \cap \big[otr(\mathcal{S})L_U\big]$. Now use Corollary 3. A simple breadth-first search on $\mathcal{T}$ gives the algorithm. □

Example 10 *Let $\mathcal{S}$ be as in Figure 4(a), with $L_I = \{a, b\}$, and $L_U = \{x\}$. Following Theorem 9, the FSA*



(a) An LTS specification $\mathcal{S}$. (b) An LTS IUT $\mathcal{I}$.

Figure 4: Relating the new conformance and **ioco** relations.

*in Figure 5 accepts $T = \overline{otr}(\mathcal{S}) \cap D$, with $D = otr(\mathcal{S})L_U$. So T is a complete test suite for $\mathcal{S}$ and $(D, \emptyset)$. Let $\mathcal{I}$ be as in Figure 4(b). We get $bax \in L(\mathcal{I})$, $bax \in T$. Hence, $\mathcal{I}$ does not $(D, \emptyset)$-conform to $\mathcal{S}$, i.e., $\mathcal{I}$ **ioco** $\mathcal{S}$ fails.* □

Given a specification $\mathcal{S}$, of which we know its structure, Theorem 9 can check if $\mathcal{I}$ **ioco**-conforms to $\mathcal{S}$ in $\mathcal{O}(t)$ time complexity for *any* IUT $\mathcal{I}$ of size $\mathcal{O}(t)$.

Corollary 4 *Let $\mathcal{S} \in \mathcal{IO}(L_I, L_U)$ be deterministic. Let $\mathcal{M}$ be a class of deterministic IUTs, of which we know their syntactic description. There is an $\mathcal{O}(t)$ algorithm to verify $\mathcal{I}$ **ioco** $\mathcal{S}$, where t is the size of $\mathcal{I}$, for all $\mathcal{I} \in \mathcal{M}$.*

Proof Follows from Theorem 9. □

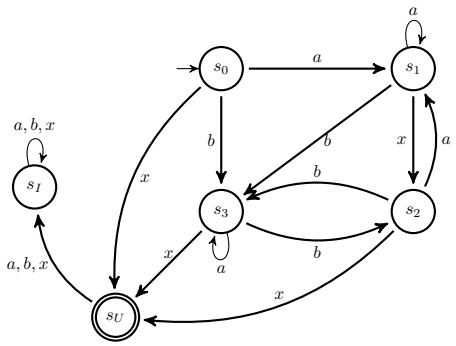


Figure 5: FSA that accepts $\overline{otr}(\mathcal{S}) \cap D$ for $\mathcal{S}$ in Figure 4(a).

5 Testing IOLTS with Test Purposes

In Section 4 the testing architecture presupposed that one has access to a syntactic description of the IUTs. In a contrasting setting, where IUTs are "black-boxes", we do not have access to their syntactic structure. In this case, we can imagine a scenario where there is a tester or "artificial environment", $\mathcal{T}$, and an IUT, $\mathcal{I}$, which are connected by a "zero-capacity" bidirectional and lossless channel. At each step either of two moves may occur: (i) $\mathcal{T}$ issues an output action symbol x to $\mathcal{I}$ and changes state. At once, $\mathcal{I}$ accepts x as an input symbol and also changes state; or (ii) the move is reversed with $\mathcal{I}$ sending an output symbol y and changing its state, while $\mathcal{T}$ accepts y as an input symbol and also changes state. Clearly, a sequence of type (i) moves can occur before a type (ii) move occurs; and vice-versa. We see that the input and output sets of symbols are interchanged in $\mathcal{T}$ and $\mathcal{I}$. Hence, one should write $\mathcal{I} = \langle S, s_0, L_I, L_U, T\rangle$ and $\mathcal{T} = \langle Q, q_0, L_U, L_I, R\rangle$. We will always refer to a symbol x as an input or an output symbol from the perspective of the IUT $\mathcal{I}$, unless there is an explicit mention to the contrary.

It may happen, however, that $\mathcal{I}$ reaches a so called *quiescent state*. Informally, those are states from which there are no transitions labeled by some *output* symbol [7, 5]. From this point on $\mathcal{I}$ could no longer respond to $\mathcal{T}$, and the latter will have no way of "knowing" whether $\mathcal{I}$ is rather slow, has timed out, or will not ever respond. If we want to reason about this situation, within the formalism, it will be necessary to somehow signal to the tester that the IUT is in a quiescent state. A usual mechanism [5] is to add a special symbol δ as an output symbol to $\mathcal{I}$, and hence as an input symbol to $\mathcal{T}$. Then $\mathcal{I}$ sends δ to $\mathcal{T}$ when it reaches a quiescent state s. Since $\mathcal{I}$ is not changing states in this situation, we also add the self-loop $s \xrightarrow{\delta} s$ to the transitions of $\mathcal{I}$. On the tester side, being on a state q and upon receiving a δ symbol it decides whether that is appropriate or not, depending on the fault model it is supposed to describe. If that was adequate, the tester may then move to another state through a move $q \xrightarrow{\delta} q'$[1].

In this section we apply our results of Section 4 to the architecture described in Tretmans [5], and where IUTs are black-boxes. But first we must formally prepare our models to deal with quiescent states. Further, when we consider models with quiescent states, we must ensure that we are treating the same class of models as in [5], and that **ioco**-conformance, as in Definition 13, is the same as the **ioco** relation studied in [5].

We proceed as follows:

1. We define a variation of IOLTS models, where the special symbol δ is used to indicate quiescence.
2. We formalize the notion of an external tester in order to reason precisely about test runs. For that, we define test purposes in Subsection 5.1.
3. When IUTs are black-boxes, it is customary to impose a series of restrictions over the formal models that describe specifications, IUTs and test purposes [5], so that some guarantees about the exchange of messages can be stated. Although our methods impose almost no restrictions on the formal models, except for regularity of the D and F sets, in Subsection 5.2 we look at the extra model restrictions imposed by Tretmans [5].
4. In Subsection 5.3 we look at the complexity of complete test purposes under these restrictions, and we establish a *new asymptotic worst case exponential* lower bound of the size of these test suites. Other works hinted at possible exponential upper bounds on the size of complete test suites for these models, but we are not aware of any precise *lower bounds*, under the same restrictions as mentioned here.

We start with the following variation of Definition 10 incorporating quiescence in IOLTS models.

Definition 17 *A δ-Input/Output Labeled Transition System (δ-IOLTS) is a tuple $\mathcal{I} = \langle S, s_0, L_I, L_U, T\rangle$:*

1. *$\langle S, s_0, L_I, L_U, T\rangle$ is an IOLTS;*
2. *$\delta \in L_U$ is a* distinguished quiescent symbol.
3. *For all $s, p \in S$, $x \in L$ we have $(s, \delta, p) \in T$ if and only if (a) $s = p$, and (b) $s \xrightarrow{x}$ implies $x \in L_I \cup \{\delta\}$.*

A state $s \in S$ is said to be quiescent *if $s \xrightarrow{\delta}$.*

We indicate the class of all δ-IOLTSs with input alphabet L_I and output alphabet L_U by $\mathcal{IO}_\delta(L_I, L_U)$. The following example illustrates the situation.

Example 11 *Consider the δ-IOLTS depicted in Figure 6, where $L_I = \{a\}$ and $L_U = \{b, \delta\}$. States s_1 and s_3 are quiescent. Since $s_0 \xrightarrow{\tau} s_1$ and $\tau \notin L_I \cup \{\delta\}$, there is no self-loop δ at s_0, and so it is not quiescent. Although s_0 is not quiescent, it can not, nevertheless, emit any output symbol. Only after the internal τ-move can it issue the symbol δ signaling quiescence.* □

[1] In [5], a different symbol, θ, was used to signal quiescence on the tester side, but in our formalism it makes no difference.

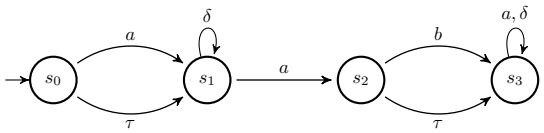


Figure 6: A simple δ-IOLTS.

In the test architecture studied in [5], given an IOLTS $\mathcal{S} = (S, s_0, L_I, L_U, T)$, a state $s \in S$ is said to be quiescent if, for all $x \in L_U \cup \{\tau\}$ we have $s \stackrel{x}{\not\rightarrow}$ in $\mathcal{S}$, a fact there indicated by $\delta(s)$. Then, assuming $\delta \notin L_U$, the extended model $\mathcal{S}_\delta = \langle S, s_0, L_I, L_U \cup \{\delta\}, T \cup T_\delta \rangle$ is defined, with $T_\delta = \{(s, \delta, s) \mid \delta(s)\}$, so that $\mathcal{S}_\delta$ includes self-loops on the new output symbol δ at any quiescent state. Since in this section we are applying our results of Sections 3 and 4 to the test architecture described in [5], it is important to guarantee that the class of δ-IOLTS models from Definition 17 is coextensive with the class of extended models in [5]. Also, the **ioco**-relation used in [5] must coincide with that at Definition 13. All details of these considerations are given in Appendix 8, specially Proposition 17 for the guarantee that both classes of models are the same, and Proposition 21 that shows that both **ioco**-relations coincide.

In this section all IUTs are from $\mathcal{IO}_\delta(L_I, L_U)$.

5.1 A Class of Fault Models

An IUT is a model from $\mathcal{IO}_\delta(L_I, L_U)$. A tester, however, is a model from $\mathcal{IO}(L_U, L_I)$, that is, from the perspective of the tester δ is now an ordinary input symbol. Upon receiving a δ-symbol, signaling quiescence on the IUT side, the test designer is free to specify any δ-transitions to drive the tester model, as deemed appropriate to the design of the test goals.

We will refer to testers as *test purposes*. We require that test purposes have two special states: **pass** and **fail** [5]. Once the **fail** or **pass** state is reached we have a test verdict, so it is reasonable to require that there are no paths from **pass** to **fail**, or vice-versa. A *fault model* is a finite set of test purposes, allowing for several conditions for acceptance and rejection to be verified.

Since $\mathcal{IO}_\delta(L_I, L_U) \subseteq \mathcal{IO}(L_I, L_U)$, it is more profitable to express some notions and results using the full $\mathcal{IO}(L_I, L_U)$ class, and specialize only when needed.

Definition 18 *Let L_I and L_U be sets of input and output symbols, respectively, with $L = L_I \cup L_U$. A* test purpose *over L is an IOLTS $\mathcal{T} = \langle Q, q_0, L_U, L_I, R \rangle$ where for all $\sigma \in L^\star$ we have neither* **fail** $\stackrel{\sigma}{\Rightarrow}$ **pass** *nor* **pass** $\stackrel{\sigma}{\Rightarrow}$ **fail**, *when* **fail**, **pass** $\in Q$. *A* fault model *over L is a finite collection of test purposes over L.*

The exchange of symbols between two models can be described by their cross-product[2].

Definition 19 *Let $\mathcal{T} = \langle Q, q_0, L_U, L_I, R \rangle$ and $\mathcal{I} = \langle S, s_0, L_I, L_U, T \rangle$ be in $\mathcal{IO}(L_I, L_U)$. Their cross-product is the LTS $\mathcal{T} \times \mathcal{I} = \langle Q \times S, (q_0, s_0), L_I \cup L_U, P \rangle$, where $((q_1, s_1), x, (q_2, s_2)) \in P$ if and only if either*

1. *$x = \tau$, $s_1 = s_2$ and $(q_1, \tau, q_2) \in R$, or*
2. *$x = \tau$, $q_1 = q_2$ and $(s_1, \tau, s_2) \in T$, or*
3. *$x \neq \tau$, $(q_1, x, q_2) \in R$ and $(s_1, x, s_2) \in T$.* □

Behavior in the cross-product is tightly tied to behaviors in the participating IOLTSs, and conversely.

Proposition 8 *Let $\mathcal{T}, \mathcal{I} \in \mathcal{IO}(L_I, L_U)$. Then:*

1. *$(t, q) \stackrel{\tau^k}{\rightarrow} (p, r)$ in $\mathcal{T} \times \mathcal{I}$ if and only if $t \stackrel{\tau^n}{\rightarrow} p$ in $\mathcal{T}$ and $q \stackrel{\tau^m}{\rightarrow} r$ in $\mathcal{I}$ with $n + m = k$, for all $m, n \geq 0$.*
2. *$(t, q) \stackrel{\sigma}{\Rightarrow} (p, r)$ in $\mathcal{T} \times \mathcal{I}$ if and only if $t \stackrel{\sigma}{\Rightarrow} p$ in $\mathcal{T}$ and $q \stackrel{\sigma}{\Rightarrow} r$ in $\mathcal{I}$, for all $\sigma \in L^\star$.*

Proof Inductions on $k \geq 0$ and $|\mu| \geq 0$, with $h_\tau(\mu) = \sigma$. □

Assume we want to verify if an IUT $\mathcal{I}$ **ioco**-conforms to a specification $\mathcal{S}$ using a test purpose $\mathcal{T}$. That is, we want $\mathcal{T}$ to act as an "external environment" and drive a run of $\mathcal{I}$, so that if $\mathcal{T}$ reaches a **fail** state in this run then $\mathcal{I}$ does not **ioco**-conform to $\mathcal{S}$. Because the simultaneous run of $\mathcal{T}$ and $\mathcal{I}$ is described by their cross product, it is sufficient to check that $\mathcal{T} \times \mathcal{I}$ cannot reach a (**fail**, s) state. Conversely, when $\mathcal{I}$ does **ioco**-conform to $\mathcal{S}$ we need that $\mathcal{T} \times \mathcal{I}$ never reaches a (**fail**, s) state. Further, we may require that $\mathcal{T}$ correctly verifies **ioco**-conformance only when $\mathcal{I}$ is taken from a subclass of models.

[2] In [5] LTS $\mathcal{T} \times \mathcal{I}$ is known as the *parallel operator* $|[\cdot]|$.

Definition 20 *Let $\mathcal{I} \in \mathcal{IO}(L_I, L_U)$, $\mathcal{T} \in \mathcal{IO}(L_U, L_I)$. We say that* $\mathcal{I}$ passes $\mathcal{T}$ *if* $(\textbf{fail}, q)$ *is not reachable in* $\mathcal{T} \times \mathcal{I}$*, for any state* q *of* $\mathcal{I}$*. We say that* $\mathcal{I}$ passes *a fault model* TP *if* $\mathcal{I}$ *passes all* $\mathcal{T}$ *in* TP*. Let* $\mathcal{S} \in \mathcal{IO}(L_I, L_U)$*,* $\mathcal{IMP} \subseteq \mathcal{IO}(L_I, L_U)$*. We say that* TP *is* **ioco**-complete *for* $\mathcal{S}$ *relative to* $\mathcal{IMP}$ *if* $\mathcal{I}$ **ioco** $\mathcal{S}$ *if and only if* $\mathcal{I}$ *passes* TP*, for all* $\mathcal{I} \in \mathcal{IMP}$*. When* $\mathcal{IMP} = \mathcal{IO}(L_I, L_U)$ *we say that* TP *is* **ioco**-*complete for* $\mathcal{S}$*.*

The following implies a single test purpose suffices.

Lemma 3 *Let* $\mathcal{S} \in \mathcal{IO}(L_I, L_U)$*. We can construct a fault model* $\{\mathcal{T}\}$ *which is* **ioco**-*complete for* $\mathcal{S}$*. Also,* $\mathcal{T}$ *is deterministic and has a single* **fail** *and no* **pass** *states.*

Proof Let $\mathcal{I}$ be an arbitrary IUT. From Definition 20 we need $\mathcal{I}$ **ioco** $\mathcal{S}$ does not hold if and only if $\mathcal{I}$ does not pass $\mathcal{T}$, that is, $(t_0, q_0) \stackrel{\sigma}{\Rightarrow} (\textbf{fail}, q)$ in $\mathcal{T} \times \mathcal{I}$ for some $\sigma \in L^\star$, where t_0 and q_0 are initial states. From Proposition 8(2) this is equivalent to $t_0 \stackrel{\sigma}{\Rightarrow} \textbf{fail}$ in $\mathcal{T}$ and $\sigma \in otr(\mathcal{I})$. We construct $\mathcal{T}$ deterministic and such that $t_0 \stackrel{\sigma}{\Rightarrow} \textbf{fail}$ holds if and only if $\sigma \in \overline{otr}(\mathcal{S}) \cap \left[otr(\mathcal{S})L_U\right]$. Thus, $t_0 \stackrel{\sigma}{\Rightarrow} \textbf{fail}$ in $\mathcal{T}$ and $\sigma \in otr(\mathcal{I})$ is equivalent to $otr(\mathcal{I}) \cap \overline{otr}(\mathcal{S}) \cap \left[otr(\mathcal{S})L_U\right] \neq \emptyset$, and Corollary 3 closes the argument.

From Proposition 5 we may take $\mathcal{S}$ deterministic. We follow the same idea as in the proof of Theorem 9, and construct $\mathcal{T}$ by extending $\mathcal{S}$ with a **fail** state and the transitions: (i) $(s, \ell, \textbf{fail})$ when $\ell \in L_U$ and there is no transition (s, ℓ, p) in $\mathcal{S}$ for any state p, and (ii) $(\textbf{fail}, \ell, \textbf{fail})$ for all $\ell \in L_U \cup L_I$. Again, we get $s_0 \stackrel{\sigma}{\Rightarrow} \textbf{fail}$ in $\mathcal{T}$ if and only if $\sigma \in \overline{otr(\mathcal{S})} \cap \left[otr(\mathcal{S})L_U\right]$. Since $\mathcal{S}$ is deterministic, it is clear that $\mathcal{T}$ is deterministic. □

As an illustration we have the following:

Example 12 *Let* $\mathcal{S}$ *as in Figure 4(a) and recall that* $L_I = \{a, b\}$ *and* $L_U = \{x\}$*. To avoid cluttering, quiescent* δ*-loops were not inserted in Figure 4(a). If needed, just add* δ*-self-loops at states* s_0*,* s_2 *and* s_3*. Also, this should pose no difficulty since a* δ *is treated as just any other symbol in* L_U*.*

The argument in Lemma 3 yields the structure in Figure 5, but: (i) s_I *and all transitions into it are removed, and (ii)* s_U *is relabeled* **fail** *and we do not need final states in* $\mathcal{T}$*.*

We have $(t_0, q_0) \stackrel{axbx}{\Rightarrow} (\textbf{fail}, q_2)$ *in* $\mathcal{T} \times \mathcal{I}$ *with* $\mathcal{I}$ *in Figure 4(b), so that* $\mathcal{I}$ *does not pass* $\mathcal{T}$*. Since Lemma 3 says* $\{\mathcal{T}\}$ *is complete for* $\mathcal{S}$*, then* $\mathcal{I}$ **ioco** $\mathcal{S}$ *should not hold. To check this, we have* $s_0 \stackrel{ba}{\Rightarrow} s_3$ *in* $\mathcal{S}$ *and* $q_0 \stackrel{ba}{\Rightarrow} q_3$ *in* $\mathcal{I}$*. Thus* $x \notin out(s_3)$ *and and* $x \in \textbf{out}(q_3)$*, so that* $out(q_0 \textbf{ after } \beta) \not\subseteq out(s_0 \textbf{ after } \beta)$*. Hence,* $\mathcal{I}$ **ioco** $\mathcal{S}$ *does not hold according to Definition 13.* □

5.2 A Specific Family of Formal Models

Some extra restrictions are imposed by Tretmans [5] so that test runs can be adjusted to more practical situations. First, one requires test purposes to be acyclic, so that any test run is a finite process. Secondly, since one cannot predict in advance which output symbol a black-box IUT will be sending back at any instant, test purposes must be able to synchronize on any of such symbols, that is, the tester must be input-enabled. Further, since the tester drives the IUT, at any state it must be able to send at least one symbol to the IUT. Also, in order to avoid arbitrary choices and non-determinism, the tester is required to be output-deterministic, that is, at any state it can emit only one of its output symbols. Moreover, because **fail** and **pass** states already hold a verdict, it is required that the tester can only have self-loops at these states. Recall Definition 13.

Definition 21 *Let* $\mathcal{T} = \langle S, s_0, L_I, L_U, R \rangle$*. We say that* $\mathcal{T}$ *is* output-deterministic *if* $|\textbf{out}(s)| = 1$*, for all* $s \in S$*.* $\mathcal{T}$ *is* input-enabled *if* $\textbf{inp}(s) = L_I$*, for all* $s \in S$*, where* $\textbf{inp}(s) = \{\ell \in L_I \mid s \stackrel{\ell}{\Rightarrow}\}$*.* $\mathcal{T}$ *is* acyclic *if it has no cycles, except for self-loops at states* **fail** *and* **pass***.* $\mathcal{IOE}(L_I, L_U)$ *is the set of all input-enabled IOLTSs over* L_I *and* L_U*.*

Input-enabledness is no serious restriction.

Corollary 5 *For any specification* $\mathcal{S}$ *there is an* **ioco**-*complete fault model* $\{\mathcal{T}\}$ *where* $\mathcal{T}$ *is deterministic, input-enabled and has a single* **fail** *and no* **pass** *states.*

Proof The test purpose constructed in proof of Lemma 3 is deterministic and input-enabled. □

It is no surprise that the acyclic requirement imposes restrictions on the size of IUTs.

Proposition 9 *Consider the deterministic specification* $\mathcal{S} = (\{s_0\}, s_0, \{a\}, \{x\}, \{(s_o, a, s_0)\})$*. Then there is no fault model* TP*, comprised only of acyclic test purposes, and which is* **ioco**-*complete for* $\mathcal{S}$ *even with respect to the subclass of all deterministic IUTs.*

Proof We see that $a^n x \in \overline{otr}(\mathcal{S}) \cap (otr(\mathcal{S})\ L_U)$ for all $n \geq 0$. Consider an IUT $\mathcal{I}_n$ with transitions (q_{i-1}, a, q_i) for $i = 1, \ldots, n$, (q_n, x, q), where q_0 is the initial state. Clearly, $\mathcal{I}_n$ is deterministic. We have $a^n x \in otr(\mathcal{I}_n)$ so that $otr(\mathcal{I}_n) \cap \left[\overline{otr}(\mathcal{S}) \cap otr(\mathcal{S})\ L_U\right] \neq \emptyset$. Then Corollary 3 says that $\mathcal{I}_n$ **ioco** $\mathcal{S}$ does not hold, for all $n \geq 0$. Because TP is complete, we must have $\mathcal{T} \in TP$ such that $\mathcal{I}_n$ does not pass $\mathcal{T}$, that is, $(\textbf{fail}, q)$ is reachable in $\mathcal{T} \times \mathcal{I}_n$. By Proposition 8 we have $t_0 \stackrel{\sigma}{\Rightarrow} \textbf{fail}$ in $\mathcal{T}$ and $q_0 \stackrel{\sigma}{\Rightarrow} r$ in $\mathcal{I}_n$, where t_0 is initial in $\mathcal{T}$ and q_0 is initial in $\mathcal{I}$. By construction of $\mathcal{I}_n$, either $\sigma = a^k$ for some $k \leq n$, or $\sigma = a^n x$. We can not have $\sigma = a^k$ for any $k \geq 0$ because then $(\textbf{fail}, s_0)$ is reachable in $\mathcal{T} \times \mathcal{S}$, implying that $\mathcal{S}$ does not pass $\mathcal{T}$. Since TP is **ioco**-complete for $\mathcal{S}$ we get that $\mathcal{S}$ **ioco** $\mathcal{S}$ does not hold with $\mathcal{S}$ deterministic, a contradiction. Let $\sigma = a^n x$. Since $\mathcal{T}$ is acyclic, except for self-loops at **fail**, and n can be arbitrarily large, $t_0 \stackrel{a^n x}{\Rightarrow} \textbf{fail}$ in $\mathcal{T}$ imply again $t_0 \stackrel{a^n}{\Rightarrow} \textbf{fail}$ in $\mathcal{T}$ for n large enough, and we reach a contradiction again. □

In the preceding proof we can take $\mathcal{S}$ where a state s has a self-loop on any input symbol. This makes the proposition much more widely applicable.

Next, we investigate the case where there is an upper bound on the number of states in the IUTs. The situation is more amenable in these cases.

Definition 22 *Let $\mathbb{IMP} \subseteq \mathbb{IO}(L_I, L_U)$ be any class of IOLTSs. We denote by $\mathbb{IMP}[m]$, $m \geq 1$, the subfamily of $\mathbb{IMP}$ comprised by all models with at most m states. Let $\mathcal{S} \in \mathbb{IO}(L_I, L_U)$. A fault model TP over $L_I \cup L_U$ is m-***ioco***-complete for $\mathcal{S}$ relatively to $\mathbb{IMP}$ if and only if it is* **ioco***-complete for $\mathcal{S}$ relatively to the class $\mathbb{IMP}[m]$.*

Since it is not possible to construct fault models comprised only of acyclic test purposes, and that are **ioco**-complete in general, we turn to the problem of obtaining such fault models that are m-**ioco**-complete, for a given m.

Proposition 10 *Let $\mathcal{S} \in \mathbb{IO}(L_I, L_U)$ be deterministic, and let $m \geq 1$. Then, there is a fault model TP which is m-***ioco***-complete for $\mathcal{S}$, and such that all test purposes in TP are deterministic and acyclic.*

Proof Let $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle$ and $S = \{s_i : 0 \leq i < n\}$. Construct a directed acyclic multi-graph D with $mn + 1$ levels. At level i, $0 \leq i \leq mn$, list all nodes in S as $s_{0,i}$, $s_{1,i}, \ldots, s_{n-1,i}$. Consider $s_{j,k}$, $k < mn$. For each transition (s_j, ℓ, s_i) of $\mathcal{S}$: (i) if $i > j$, add an arc from $s_{j,k}$ to $s_{i,k}$; (ii) if $i \leq j$ add an arc from $s_{j,k}$ to $s_{i,k+1}$. Let ℓ label the new arc. Next, add a **fail** node to S and for any $\ell \in L_U$ and any $s_{i,k}$, if $(s_i, \ell, p) \notin T$ for every $p \in S$, add an arc labeled ℓ from $s_{i,k}$ to **fail**. Finally, let $s_{0,0}$ be the root and discard any node not reachable from it. Clearly, D is acyclic. Also, $s_0 \stackrel{\sigma}{\Rightarrow} s_i$ with $|\sigma| = k \leq mn$ if and only if there is a path labeled σ from $s_{0,0}$ to some $s_{i,r}$ in D.

For each path $s_{i_0} \stackrel{x_1}{\rightarrow} s_{i_1} \stackrel{x_2}{\rightarrow} \cdots \stackrel{x_{i_r}}{\rightarrow} s_{i_r}$ in D where $s_{i_0} = s_{0,0}$ and $s_{i_r} = \textbf{fail}$, we add the acyclic test purpose $\mathcal{T} = (\{s_{i_0}, \ldots, s_{i_r}\}, s_{i_0}, L_U, L_I, \{(s_{i_{j-1}}, x_j, s_{i_j} \mid 1 \leq j \leq r\})$ to TP. Clearly, $\mathcal{T}$ is deterministic.

Assume that $\mathcal{I}$ **ioco** $\mathcal{S}$ does not hold, where $\mathcal{I} = (Q, q_0, L_I, L_U, R)$ with $|Q| = h \leq m$. We show that $\mathcal{I}$ does not pass TP. By Corollary 3 we get some $\sigma \in otr(\mathcal{I})$, $\sigma \notin otr(\mathcal{S})$ and $\sigma \in otr(\mathcal{S})L_U$. Let $|\sigma|$ be minimum. Clearly, $\sigma = \alpha\ell$, with $\ell \in L_U$ and $\alpha \in otr(\mathcal{S})$ so that $s_0 \stackrel{\alpha}{\Rightarrow} s$. From $\alpha\ell \in otr(\mathcal{I})$ we get $q_0 \stackrel{\alpha}{\Rightarrow} q \stackrel{\ell}{\Rightarrow} q'$. Let $\alpha = x_1 \ldots x_r$ $(r \geq 0)$. By Proposition 8, $(s_0, q_0) \stackrel{x_1}{\Rightarrow} (s_1, q_1) \stackrel{x_2}{\Rightarrow} (s_2, q_2) \stackrel{x_3}{\Rightarrow} \ldots \stackrel{x_r}{\Rightarrow} (s_r, q_r)$, with $s = s_r$ and $q = q_r$. We argue that $r < mn$. If $r \geq mn \geq hn$ we get $(s_i, q_i) = (s_j, q_j)$ for some $i < j$. Then, $(s_0, q_0) \stackrel{\mu}{\Rightarrow} (s_r, q_r)$ with $\mu = x_1 \ldots x_i x_{j+1} \ldots x_r$. Again, $s_0 \stackrel{\mu}{\Rightarrow} s_r$ and $q_0 \stackrel{\mu}{\Rightarrow} q_r$, which gives $\mu \in otr(\mathcal{S})$ and $\mu \in otr(\mathcal{I})$. Moreover, $(s_r, \ell, p) \notin T$ for all $p \in S$, otherwise $\sigma = \alpha\ell \in otr(\mathcal{S})$, a contradiction. Hence, $\mu\ell \notin otr(\mathcal{S})$, but $\mu\ell \in otr(\mathcal{S})L_U$. Further, $q \stackrel{\ell}{\Rightarrow} q'$ and $q = q_r$ give $\mu\ell \in otr(\mathcal{I})$. Thus, $\mu\ell \in otr(\mathcal{I}) \cap \left[\overline{otr}(\mathcal{S}) \cap otr(\mathcal{S})L_U\right]$. But $|\mu\ell| < |\alpha\ell| = |\sigma|$ violates the minimality of $|\sigma|$. Hence, $|\sigma| = r + 1 \leq mn$. Since $s_0 \stackrel{\alpha}{\Rightarrow} s_r$ we get a path α from $s_{0,0}$ to a node $s_{i,k}$. Since $(s_r, \ell, p) \notin T$ for any any $p \in S$, there is an arc ℓ from $s_{i,k}$ to **fail**, and so the path $\alpha\ell$ goes from $s_{0,0}$ to **fail**. Thus, we put a $\mathcal{T} = (S_\mathcal{T}, t_0, L_U, L_I, T_\mathcal{T})$ in TP with $t_0 \stackrel{\alpha\ell}{\Rightarrow} \textbf{fail}$. So, $(t_0, q_0) \stackrel{\sigma}{\Rightarrow} (\textbf{fail}, q')$ in $\mathcal{T} \times \mathcal{I}$, and so $\mathcal{I}$ does not pass TP.

For the converse assume that $\mathcal{I} = (Q, q_0, L_I, L_U, R)$ does not pass TP. We have $(t_0, q_0) \stackrel{\sigma}{\Rightarrow} (\textbf{fail}, q)$ in $\mathcal{T} \times \mathcal{I}$, for some $\mathcal{T} = (S_\mathcal{T}, s_{0,0}, L_U, L_I, T_\mathcal{T})$ in TP. So, $s_{0,0} \stackrel{\sigma}{\Rightarrow} \textbf{fail}$ in $\mathcal{T}$ and $q_0 \stackrel{\sigma}{\Rightarrow} q$ in $\mathcal{I}$. Thus, $\sigma \in otr(\mathcal{I})$. By the construction, $s_{0,0} \stackrel{\alpha}{\Rightarrow} s_{i,k} \stackrel{\ell}{\Rightarrow} \textbf{fail}$ in $\mathcal{T}$, and (s_i, ℓ, p) is not a transition in $\mathcal{S}$, for all $p \in S$. Then $s_0 \stackrel{\alpha}{\Rightarrow} s_i$ in $\mathcal{S}$, so that $\sigma = \alpha\ell \in otr(\mathcal{S})L_U$. If $\alpha\ell \in otr(\mathcal{S})$ we would have $s_0 \stackrel{\alpha}{\Rightarrow} s' \stackrel{\ell}{\Rightarrow} s''$ in $\mathcal{S}$. Since $\mathcal{S}$ is deterministic, $s_i = s'$ and so (s_i, ℓ, s'') is a transition in $\mathcal{S}$, a contradiction. Thus, $\sigma = \alpha\ell \in \overline{otr}(\mathcal{S})$. Whence, $\sigma \in otr(\mathcal{I}) \cap \overline{otr}(\mathcal{S}) \cap (otr(\mathcal{S})L_U)$, and, by Corollary 3 $\mathcal{I}$ **ioco** $\mathcal{S}$ does not hold.

We have shown that $\mathcal{I}$ **ioco** $\mathcal{S}$ if and only if $\mathcal{I}$ passes TP, for any IUT $\mathcal{I}$ with at most m states. □

Example 13 *Let $\mathcal{S}$ as in Figure 4(a) with $L_I = \{a, b\}$ and $L_U = \{x\}$. Again, quiescent δ symbols were not included to keep the example simple, and pose no extra difficulty in the construction. The construction in*

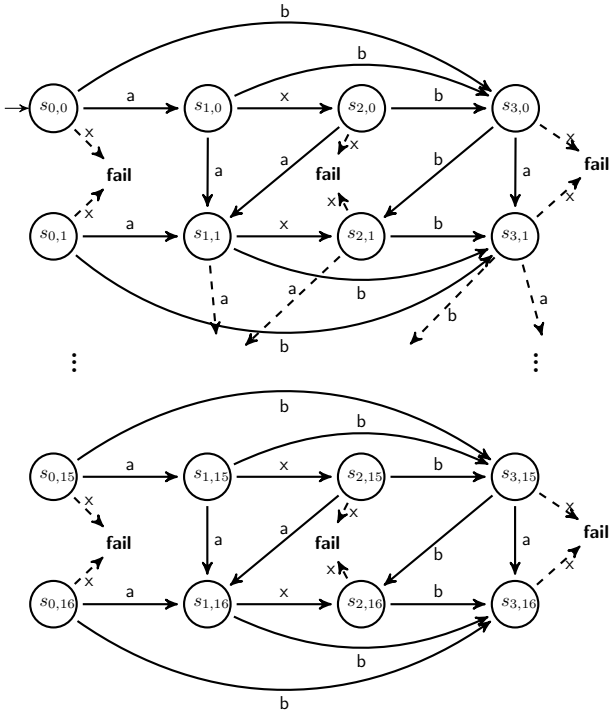


Figure 7: A direct acyclic multi-graph D for Figure 4(a).

the proof of Proposition 10, gives the multi-graph D, of which Figure 7 depicts the first two and the last two levels. As $\mathcal{S}$ has $n = 4$ states we have four states at each level. In this example we consider IUTs with at most $m = n = 4$ states, so there are $mn + 1 = 17$ levels in D. To avoid cluttering figure, we replicated the **fail** *label.*

A simple algorithm collects test purposes by traversing D from $s_{0,0}$ to **fail**. *One possible traversal is $s_{0,0} \to s_{1,0} \to s_{1,1} \to s_{2,1} \to s_{1,2} \to s_{3,2} \to s_{2,3} \to s_{3,3} \to s_{3,4} \to$* **fail**, *and yields $\alpha = aaxabbbax$.*

For $\mathcal{I}$ as in Figure 4(b), α leads from q_0 to q_2. So that IUT does not pass the test purpose induced by α and so $\mathcal{I}$ **ioco** *$\mathcal{S}$ does not hold, as expected again.* □

A careful observation of the construction at Proposition 10, with minor adjustments, reveals that one can get m-**ioco**-complete fault models whose test purposes are deterministic, output-deterministic, input-enabled and acyclic, except for self-loops at **pass** and **fail** states.

Proposition 11 *Let $\mathcal{S} \in \mathcal{IO}(L_I, L_U)$ be deterministic, and $m \geq 1$. Then, there is a fault model TP which is* **ioco***-complete for $\mathcal{S}$ relatively to $\mathcal{IO}(L_I, L_U)[m]$, and such that all test purposes in TP are deterministic, input-enabled, output-deterministic, and acyclic except for self-loops at* **fail** *and* **pass** *states.*

Proof By Proposition 10 we get TP m-**ioco**-complete for $\mathcal{S}$, with all $\mathcal{T} \in TP$ acyclic and deterministic.

Fix $\mathcal{T} = \langle S, t_0, L_U, L_I, T \rangle \in TP$. In order to secure input-enabledness, add a **pass** state to S. For all $\ell \in L_U$, all $t \neq$ **fail**, if $(t, \ell, t') \notin T$ for any $t' \in S$, add $(t, \ell,$ **pass**$)$ to T. Also, add transitions (**pass**$, \ell,$ **pass**$)$ and (**fail**$, \ell,$ **fail**$)$ to T, for all $\ell \in L_U$. In the end, we get $\mathcal{T}'$ which is deterministic, input-enabled and acyclic, except for self-loops at **pass** and **fail**. An simple argument shows that $t_0 \stackrel{\sigma}{\Rightarrow}$ **fail** in $\mathcal{T}$ if and only if $t_0 \stackrel{\sigma}{\Rightarrow}$ **fail** in $\mathcal{T}'$. Hence, we see that $\mathcal{I}$ passes $\mathcal{T}$ if and only if $\mathcal{I}$ passes $\mathcal{T}'$ for all $\mathcal{I} \in \mathcal{IO}(L_I, L_U)[m]$. By so adjusting all $\mathcal{T} \in TP$ we get TP' which is m-**ioco**-complete for $\mathcal{S}$, and such that all $\mathcal{T} \in TP'$ is deterministic, input-enabled and acyclic, except for self-loops at **pass** and **fail**.

Let $\mathcal{T} \in TP$ as in Proposition 10, and $\mathcal{T}' \in TP'$ obtained from $\mathcal{T}$ as in the previous paragraph. Fix some $a \in L_I$. Let t be a state in $\mathcal{T}$, with **fail** $\neq t \neq$ **pass**. We have at exactly one transition (t, ℓ, t') in $\mathcal{T}$, and on passing to $\mathcal{T}'$ we added no transitions on symbols from L_I. Thus, either there is exactly one transition (t, ℓ, t') in $\mathcal{T}'$ with $\ell \in L_I$, or there is none. If there is none, add $(t, a,$ **pass**$)$ to $\mathcal{T}'$, so that now t is output-deterministic. Apply this transformation to all states in $\mathcal{T}'$ and complete the transformation by adding (**fail**$, a,$ **fail**$)$ and (**pass**$, a,$ **pass**$)$ to $\mathcal{T}'$, yielding a new test purpose $\mathcal{T}''$. Clearly, $\mathcal{T}''$ is deterministic, input-complete, output-deterministic, and acyclic except for self-loops at **fail** and **pass**. Further, it is easy to see that $\mathcal{I}$ passes $\mathcal{T}'$ if and only if $\mathcal{I}$ passes $\mathcal{T}''$, for all $\mathcal{I} \in \mathcal{IO}(L_I, L_U)$. Apply this transformation to all $\mathcal{T}' \in TP'$ to get TP''. Then TP'' is m-**ioco**-complete for $\mathcal{S}$, because TP' is. □

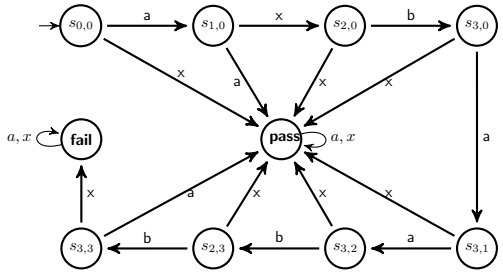


Figure 8: An example TP extracted from Figure 7.

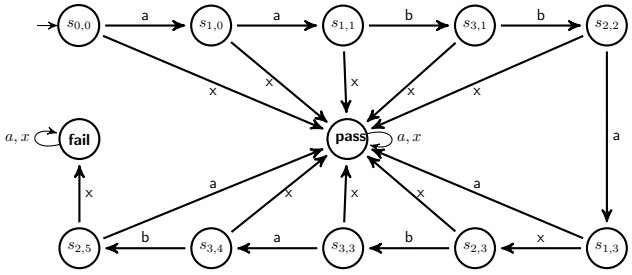


Figure 9: Another TP extracted from Figure 7.

The next example illustrates the construction. Note that the quiescent δ symbols were not included to keep the example uncluttered. They offer no extra difficulty since they are treated as any plain symbol in L_U.

Example 14 *The outer path from $s_{0,0}$ to $s_{3,3}$ to* **fail** *in Figure 8 represents a test purpose extracted according to Proposition 10 and already illustrated in Example 13.*

Recall that $L_U = \{x\}$ and $L_I = \{a, b\}$. Following Proposition 11, in order to secure input-enabledness, we added a new **pass** *state. Proceeding, for all states $s_{i,j}$ for which there were no outgoing transition on x, we added transitions $(s_{i,j}, x, \textbf{pass})$. We completed this step by adding $(\textbf{fail}, x, \textbf{fail})$ and $(\textbf{pass}, x, \textbf{pass})$. Next, we fixed $a \in L_I$, and for all states $s_{i,j}$ for which there were no outgoing transition on a symbol of $\{a, b\}$ we added transitions $(s_{i,j}, a, \textbf{pass})$. Adding the transitions $(\textbf{pass}, a, \textbf{pass})$ and $(\textbf{fail}, a, \textbf{fail})$ completed the construction. The test purpose depicted in Figure 8 is easily seen to be deterministic, input-enabled, output-deterministic and acyclic, except by self-loops at* **fail** *and* **pass**.

Fix the IUT $\mathcal{I}$ in Figure 4(b), and recall that $\mathcal{T}$ at Figure 8 was constructed from the original specification $\mathcal{S}$ in Figure 4(a). In $\mathcal{T}$, the shortest path from $s_{0,0}$ to **fail** *is $\sigma = axba^2b^2x$. We also see that $\sigma \in otr(\mathcal{I})$, so that in the cross-product $\mathcal{T} \times \mathcal{I}$ we get $(s_{0,0}, q_0) \stackrel{\sigma}{\Rightarrow} (\textbf{fail}, q_2)$. This shows that $\mathcal{I}$* **ioco** *$\mathcal{S}$ does not hold, and $\mathcal{T}$ is sufficient to reach a negative verdict. Clearly, from this verdict alone we can not conclude that $\{\mathcal{T}\}$ is 4-complete for $\mathcal{S}$.*

As another illustration, Figure 9 shows $\mathcal{T}'$, the result obtained after extracting another test purpose from the same multi-graph D in Example 13, and transforming it according to Proposition 11. The set of paths from $s_{0,0}$ to **fail** *is $P = \{a^2b^2axbabx^n : n \geq 1\}$. For the same IUT $\mathcal{I}$ in Figure 4(b) we see that $P \cap otr(\mathcal{I}) = \emptyset$, so that we do not have $(s_{0,0}, q_0) \stackrel{\sigma}{\Rightarrow} (\textbf{fail}, q)$ for any state q of $\mathcal{I}$ and any $\sigma \in P$. Since $\mathcal{I}$* **ioco** *$\mathcal{S}$ does not hold, we conclude that $\{\mathcal{T}'\}$ is not 4-***ioco***-complete for $\mathcal{S}$.* □

These partial results show that we effectively construct test purposes that satisfy all requirements listed by Tretmans [5] when testing black-box IUTs.

Theorem 15 *Let $\mathcal{S} \in \mathcal{IO}(L_I, L_U)$ and $m \geq 1$. We can effectively construct a finite fault model TP which is m-***ioco***-complete for $\mathcal{S}$, with all test purposes in TP deterministic, input-enabled, output-deterministic, and acyclic except for self-loops at* **fail** *and* **pass** *states.*

Proof From Propositions 5 and 11. □

It is not hard to see that all our models given by Theorem 15 satisfy all restrictions that must be obeyed by test cases in [5], Definition 10. A detailed, step by step argument can be seen in Appendix 8, specially Proposition 24. Also, if one has a different set of characteristics, stemming from another kind of testing architecture, and with somewhat different requirements to be satisfied by test purposes as compared to those proposed by Tretmans [5], one might try to proceed as discussed here, and transform each basic test purpose so as to make it adhere to that specific set of new requirements.

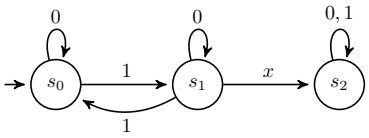


Figure 10: A simple IOLTS that whose fault model is super-polynomial.

5.3 On the Complexity of Test Purposes

We look at the complexity of the specific family of test purposes constructed in Subsections 5.1 and 5.2. Let $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle$ be a deterministic specification, with $|S| = n$. Consider the acyclic multi-graph D, described in the proof of Proposition 10, and that was used to obtain acyclic test purposes that are m-**ioco**-complete for $\mathcal{S}$. Since $\mathcal{S}$ is deterministic, it is clear that D is also deterministic and acyclic. Moreover, since D has $nm+1$ levels with at most n nodes per level, we conclude that D has at most n^2m+n nodes. Although the number of nodes and levels in D are polynomial on n and m, the number of traces in D might be super-polynomial on n and m, in general. Since we extract the test purposes from such traces, the fault model that is so generated might also be of super-polynomial size on n and m. We argue that, in general, this is unavoidable, even when we restrict all models to the smaller class of deterministic input-enabled IOLTS models.

Theorem 16 *Let $m \geq 3$, $L_I = \{0,1\}$, and $L_U = \{x\}$. Consider the simple deterministic, input-enabled specification $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle$ in Figure 10. Let TP be a fault model which is m-**ioco**-complete for $\mathcal{S}$, relatively to the class of all deterministic and input-enabled IUTs. If all test purposes in TP are deterministic and output-deterministic, then TP must be comprised of at least $\Omega(\Phi^m)$ distinct test purposes, where $\Phi = (1+\sqrt{5})/2$.*

Proof Clearly, $\mathcal{S}$ is deterministic and input-enabled. Let $R = (0+11)^\star$. Define $\overline{\sigma} = 1$ when $\sigma = 0$, and let $\overline{\sigma} = 0$ when $\sigma = 1$. Let $\alpha = y_1 \dots y_r \in R$, with $1 \leq r \leq m-3$. It is clear that $\alpha \in otr(\mathcal{S})$, $\alpha x \in otr(\mathcal{S})L_U$, and $\alpha x \notin otr(\mathcal{S})$, and so $\alpha x \in \overline{otr(\mathcal{S})}$.

Now let $\mathcal{I}_\alpha = \langle S_\mathcal{I}, q_o, L_I, L_U, T_\mathcal{I}\rangle$ be an IUT given by (q_{i-1}, y_i, q_i) for $1 \leq i \leq r$, and (q_r, x, q_{r+1}). For input-enabledness, add a new **pass** state, the transitions $(q_{i-1}, \overline{y_i}, \textsf{pass})$ $(1 \leq i \leq r)$, as well as $(q_r, \sigma, \textsf{pass})$ and the self-loops $(q_{r+1}, \sigma, q_{r+1})$, $(\textsf{pass}, \sigma, \textsf{pass})$, for $\sigma \in \{0,1\}$. See Figure 11. Clearly, $\mathcal{I}_\alpha$ has $r+3 \leq m$ states, is deterministic and input-enabled.

Assume that TP is a fault model which is m-**ioco**-complete for $\mathcal{S}$, relatively to the class of all deterministic input-enabled IUTs. Since $\alpha x \in otr(\mathcal{I})$, we get $otr(\mathcal{I}_\alpha) \cap \left[\overline{otr}(\mathcal{S}) \cap (otr(\mathcal{S})L_U)\right] \neq \emptyset$, so Corollary 3 says that $\mathcal{I}_\alpha$ **ioco** $\mathcal{S}$ does not hold. Hence, $t_0 \stackrel{\sigma}{\Rightarrow} \textsf{fail}$ in $\mathcal{T}_\alpha$ and $q_0 \stackrel{\sigma}{\Rightarrow} q$ in $\mathcal{I}_\alpha$ for some $q \in S_\mathcal{I}$, $\sigma \in (L_i \cup L_U)^\star$. If $\sigma \in \{0,1\}^\star$ we get $s_0 \stackrel{\sigma}{\Rightarrow} s$ and then $(t_0, s_0) \stackrel{\sigma}{\Rightarrow} (\textsf{fail}, s)$ in $\mathcal{T}_\alpha \times \mathcal{S}$. and $\mathcal{S}$ does not pass TP. Since TP is m-**ioco**-complete for $\mathcal{S}$, it must be that $\mathcal{S}$ **ioco** $\mathcal{S}$ does not hold, a clear contradiction. Hence $\sigma \notin \{0,1\}^\star$. Thus, since $\sigma \in otr(\mathcal{I}_\alpha)$, we must have $\sigma = \alpha x \alpha'$ with $x \in L_I$ and $\alpha' \in \{0,1\}^\star$.

Next, we look at other test purposes in TP. Let $\beta \in R$, $|\beta| = |\alpha|$ and $\beta \neq \alpha$. A similar reasoning gives $\mathcal{T}_\beta = \langle S_\beta, t_0', L_U, L_I, T_\beta\rangle$ in TP, with $t_0' \stackrel{\beta x \beta'}{\Rightarrow} \textsf{fail}$ in $\mathcal{T}_\beta$, where $\beta' \in \{0,1\}^\star$. We argue that $\mathcal{T}_\alpha \neq \mathcal{T}_\beta$. If $\mathcal{T}_\alpha = \mathcal{T}_\beta$ we get $t_0 \stackrel{\alpha x \alpha'}{\Rightarrow} \textsf{fail}$ and $t_0 \stackrel{\beta x \beta'}{\Rightarrow} \textsf{fail}$ in $\mathcal{T}_\alpha$. Since prefixes of α and of β are in $otr(\mathcal{S}) \cap \{0,1\}^\star$, we can not reach **fail** in $\mathcal{T}_\alpha$ with such prefixes, otherwise we would again conclude that $\mathcal{S}$ **ioco** $\mathcal{S}$ does not hold, a contradiction. We, therefore, must have $t_0 \stackrel{\mu}{\Rightarrow} t_1 \stackrel{x_1}{\Rightarrow} t_2 \stackrel{\alpha_1 x \alpha'}{\Rightarrow} \textsf{fail}$ and $t_0 \stackrel{\mu}{\Rightarrow} t_1' \stackrel{x_2}{\Rightarrow} t_2' \stackrel{\beta_1 x \beta'}{\Rightarrow} \textsf{fail}$ in $\mathcal{T}_\alpha$, with $\alpha = \mu x_1 \alpha_1$ and $\beta = \mu x_2 \beta_1$. Since $\mathcal{T}_\alpha$ is deterministic, we have $t_1 = t_1'$. This gives (t_1, x_1, t_2) and (t_1, x_2, t_2') as transitions in $\mathcal{T}_\alpha$, with $x_1 \neq x_2$ in $\{0,1\}$. But this contradicts $\mathcal{T}_\alpha$ being output-deterministic. We conclude that $\mathcal{T}_\alpha \neq \mathcal{T}_\beta$ when $\alpha \neq \beta$ and $|\alpha| = |\beta|$, with α, $\beta \in R$.

Consider words of length m in R. Since 1 occurs only in blocks of two in R, there are $\binom{m-i}{i}$ distinct words of length m with i such blocks in R. Thus, there are $F_m = \sum_{i=0}^{\lfloor m/2 \rfloor} \binom{m-i}{i}$ words of length m in R, where F_m is the mth Fibonacci number, $F_m = \frac{1}{\sqrt{5}}\left(\Phi^m + \frac{1}{\Phi^m}\right) \geq \frac{\Phi^m}{\sqrt{5}}$, where $\Phi = (1+\sqrt{5})/2$. Thus, there are at least $\Phi^m/\sqrt{5}$ distinct elements in TP.

For later reference, note that both transitions from q_r to **pass**, as well as both self-loops at q_{r+1}, were never necessary in the proof. Their only function here is to make states q_r and q_{r+1} input-enabled. □

It is clear that Theorem 16 applies to any specification $\mathcal{S}$ in which Figure 10 is a sub-model.

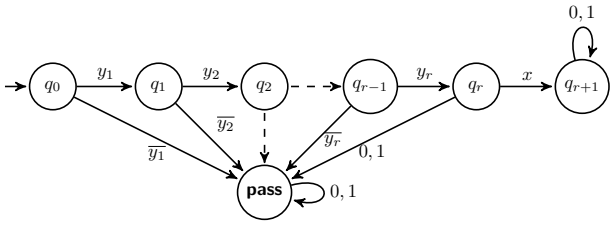


Figure 11: Implementation $\mathcal{I}_\alpha$ for the specification of Figure 10.

6 Another Restricted Class of IOLTS Models

As another illustration of our approach, in this section we look at a subclass of IOLTS models that were studied more recently [10]. In that work, Simão and Petrenko considered a more contrived subclass of IOLTS models, and showed that it is possible to generate **ioco**-complete test suites for specifications in that subclass, although they did not study the complexity of the test suites that were generated. In this section we show how to construct **ioco**-complete test suites for that same subclass of IOLTS models in a more unified and direct way, and we also look at the complexity of the test suites that are generated using our approach. As one of the main results of this section, we also establish a precise *exponential lower bound* on the worst case asymptotic size of *any test suite* that is required to be complete for the class of IOLTS models treated here.

Since there are several restrictions that IOLTS models must satisfy in Simão and Petrenko's approach [10], we introduce them in stages, as needed. Motivation for considering these restrictions can be found in their work [10]. Recall Definitions 13 and 21. First we use the **inp** and **out** functions that collect inbound and outbound transitions, respectively, in order to characterize the notions of input-complete and output-complete states, among others. Also we need the notion of **init**, where $\mathbf{init}(V) = \mathbf{inp}(V) \cup \mathbf{out}(V)$, for all $V \subseteq S$.

Definition 23 ([10]) *Let $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle$ be an IOLTS, with $L = L_I \cup L_U$, and let $s \in S$. We say that s is: (i) a* sink state *if $\mathbf{init}(\{s\}) = \emptyset$; (ii) a* single-input *state when $|\mathbf{inp}(\{s\})| \leq 1$; (iii) a* input-state *when $\mathbf{inp}(\{s\}) \neq \emptyset$; (iv) an* input-complete *state if $\mathbf{inp}(\{s\}) = L_I$ or $\mathbf{inp}(\{s\}) = \emptyset$; and (v) an* output-complete *state when $\mathbf{out}(\{s\}) = L_U$ or $\mathbf{out}(\{s\}) = \emptyset$. We say that the IOLTS $\mathcal{S}$ is single-input, input-complete, or output-complete if all states in S are, respectively, single-input, input-complete, or output-complete. We also say that $\mathcal{S}$ is* initially-connected *if every state in $\mathcal{S}$ is reachable from the initial state, and we say that $\mathcal{S}$ is* progressive *if it has no sink state and for any cycle $q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} q_2 \xrightarrow{x_3} \cdots \xrightarrow{x_k} q_k$, with $q_0 = q_k$ we have $x_j \in L_I$ for some $1 \leq j \leq k$. Let $\mathcal{IOIP}(L_I, L_U) \subseteq \mathcal{IO}(L_I, L_U)$ denote the class of all IOLTSs which are deterministic, input-complete, progressive and initially-connected.*

Note the difference in the notions of input-completeness that appear in Definitions 23 and 21.

In Definition 14, the terms test case and test suite refer to words and languages over $L_I \cup L_U$, respectively. To avoid confusion in this section we will use the terms *schemes* and *scheme suites*, respectively, when referring to words and languages, as in Definition 3 of Simão and Petrenko [10]. Note that, contrary to the notion of a test purpose in Definition 18, test schemes in Simão and Petrenko's approach do not have their sets of input and output symbols reversed with respect to the sets of input and output symbols of specifications and IUTs.

Definition 24 *Let L_I and L_U be sets with $L_I \cap L_U = \emptyset$ and $L = L_I \cup L_U$. A* scheme over L *is an acyclic single-input and output-complete IOLTS $\mathcal{T} \in \mathcal{IO}(L_I, L_U)$ which has a single sink state, designated* **fail**. *A* scheme suite SS *over L is a finite set of schemes over L.*

Proceeding, recall Definition 19, of the cross-product operator $\mathcal{S} \times \mathcal{I}$. Simão and Petrenko [10] denote the exactly same operator by $\mathcal{S} \cap \mathcal{I}$, with the proviso that internal τ-moves are not considered. In this section we will continue to use $\mathcal{S} \times \mathcal{I}$ to denote synchronous execution. We also remark now that Definition 20, saying when an IUT $\mathcal{I}$ passes a scheme suite SS, exactly matches Definition 4 of Simão and Petrenko. So, we also have the same notion of **ioco**-completeness, as stated in our Definition 20 and their Definition 4.

In what follows we want to show that our approach can also be used to construct **ioco**-complete scheme suites, but with the advantage that we do not need to further constrain specification and IUT models.

Theorem 17 *Let $\mathcal{S} \in \mathcal{IO}(L_I, L_U)$ with $L = L_I \cup L_U$, and let $m \geq 1$. We can effectively construct a scheme suite SS over L which is m-**ioco**-complete for $\mathcal{S}$.*

Proof If needed, use Proposition 5 to transform $\mathcal{S}$ into an equivalent deterministic IOLTS. From Proposition 10 we get a scheme suite SS which is **ioco**-complete for $\mathcal{S}$ relatively to $\mathcal{IO}(L_I, L_U)[m]$, and such that all schemes in SS are deterministic and acyclic, and have a single **fail** state, which is also a sink.

Let $\mathcal{T} = \langle S_\mathcal{T}, t_0, L_I, L_U, R_\mathcal{T}\rangle$ in SS and let $s \in S_\mathcal{T}$. From that proof, we know that there is at most one transition (s, ℓ, p) in $R_\mathcal{T}$, for any $\ell \in L_U \cup L_I$ and any $p \in S_\mathcal{T}$. If $\ell \in L_I$, from Definition 23, we get $|\,\mathbf{inp}(s)| \leq 1$ and $\mathbf{out}(s) = \emptyset$, and so s is single-input and output-complete. If $\ell \in L_U$ then s is already single-input. If $|L_U| = 1$, then s is also output-complete. Else, we construct $\mathcal{T}'$ by adding a new **pass** state and, for any $x \in L_U$ with $x \neq \ell$, we add $(s, x, \mathbf{pass})$ to $R_\mathcal{T}$. Clearly, s is now output-complete in $\mathcal{T}'$.

Fix any IUT $\mathcal{I} = \langle S_\mathcal{I}, q_0, L_I, L_U, R_\mathcal{I}\rangle \in \mathcal{IO}(L_I, L_U)$. Since **pass** is a sink in $\mathcal{T}'$, we get $(t_0, q_0) \stackrel{\star}{\Rightarrow} (\mathbf{fail}, q)$ in $\mathcal{T} \times \mathcal{I}$ if and only if $(t_0, q_0) \stackrel{\star}{\Rightarrow} (\mathbf{fail}, q)$ in $\mathcal{T}' \times \mathcal{I}$. Thus, $\mathcal{I}$ passes $\mathcal{T}$ if and only if $\mathcal{I}$ passes $\mathcal{T}'$.

Apply transformation to all $\mathcal{T}$ in SS to get a new scheme suite SS'. Since SS is m-**ioco**-complete for $\mathcal{S}$ then so is SS'. Clearly, SS' satisfies all the requirements in Definition 24. □

A similar result follows if we constrict all specifications and IUTs to the more restricted subclass $\mathcal{IOIP}(L_I, L_U)$.

Corollary 6 *Let $\mathcal{S} \in \mathcal{IOIP}(L_I, L_U)$ with $L = L_I \cup L_U$, and let $m \geq 1$. We can effectively construct a scheme suite SS over L which is m-**ioco**-complete for $\mathcal{S}$ relatively to $\mathcal{IOIP}(L_I, L_U)$.*

Proof Note that $\mathcal{IOIP}(L_I, L_U) \subseteq \mathcal{IO}(L_I, L_U)$ and use Theorem 17. □

Simão and Petrenko [10] consider specifications and IUTs that are further restricted to be input-state-minimal, in the sense that any two distinct input-states are always distinguishable. In their work, two states r and p are said to be distinguishable when there are no sink state in the cross-product $\mathcal{S}_{/r} \times \mathcal{S}_{/p}$, where $\mathcal{S}_{/p}$ is the as $\mathcal{S}$, but now with p being the initial state.

Definition 25 ([10]) *Let $\mathcal{S} = \langle S_\mathcal{S}, s_0, L_I, L_U, R_\mathcal{S}\rangle$ and $\mathcal{Q} = \langle S_\mathcal{Q}, q_0, L_I, L_U, R_\mathcal{Q}\rangle$, and let $s \in S_\mathcal{S}$, $q \in S_\mathcal{Q}$. We say that s and q are* distinguishable *if there is a sink state in $\mathcal{S}_{/s} \times \mathcal{Q}_{/q}$, else we say that they are* compatible*. We say that $\mathcal{S}$ is* input-state-minimal *if any two distinct input-states r, $s \in S_\mathcal{S}$ are distinguishable. Also, $\mathcal{IOMIN}(L_I, L_U) \subseteq \mathcal{IOIP}(L_I, L_U)$ denotes the subclass of all models which are input-state-minimal.*

Recall Definition 23. IUTs are yet further constrained by Simão and Petrenko [10] to have at most as many input-states as the specification. Let $k \geq 1$, and let $\mathcal{IMP}(L_I, L_U) \subseteq \mathcal{IO}(L_I, L_U)$. We denote by $\mathcal{IMP}(L_I, L_U, k)$ the subclass of $\mathcal{IMP}(L_I, L_U)$ comprised by all models with at most k input-states. The main result in [10] is their Theorem 1, which shows that for any $\mathcal{S} \in \mathcal{IOMIN}(L_I, L_U)$ we can construct scheme suites that are **ioco**-complete for IUTs in $\mathcal{IOMIN}(L_I, L_U, k)$, where k is the number of input-states in $\mathcal{S}$. This result also follows easily from Theorem 17.

Corollary 7 *Let $m \geq 1$ and $\mathcal{S} \in \mathcal{IOMIN}(L_I, L_U)$ with $k \geq 0$ input-states. We can effectively construct a finite scheme suite SS over $L_I \cup L_U$ which is m-**ioco**-complete for $\mathcal{S}$ relatively to the sub-class $\mathcal{IOMIN}(L_I, L_U, k)$.*

Proof We have $\mathcal{IOMIN}(L_I, L_U) \subseteq \mathcal{IO}(L_I, L_U)$ and $\mathcal{IOMIN}(L_I, L_U, k)[m] \subseteq \mathcal{IO}(L_I, L_U)[m]$. Now apply Theorem 17. □

The complexity of the generated test suites were not analyzed by Simão and Petrenko [10]. However, as we argued in Subsection 5.3 and in Theorem 16, we cannot, in general, avoid scheme suites to asymptotically grow very large, even when specifications are confined to the subclass $\mathcal{IOMIN}(L_I, L_U)$, and IUTs are restricted to the subclass $\mathcal{IOMIN}(L_I, L_U, k)$, where k is the number of input-states in $\mathcal{S}$. The next result establishes a worst case exponential asymptotic lower bound on the size of the test schemes that can be generated using their Theorem 1 of [10] or, equivalently, using our Corollary 7.

Theorem 18 *Let $3 \leq m \leq k$, $L_I = \{0, 1\}$ and $L_U = \{a, x\}$. There is a deterministic $\mathcal{S} \in \mathcal{IOMIN}(L_I, L_U)$ with k input-states such that any m-**ioco**-complete scheme suite for $\mathcal{S}$, relatively to $\mathcal{IOMIN}(L_I, L_U, k)$, must be of size $\Omega(\Phi^m)$, where $\Phi = (1 + \sqrt{5})/2$.*

Proof We argue almost exactly as that in the proof of Theorem 16, with a few adjustments to appear later on.

Note that $\mathcal{S}$, as used in the proof of Theorem 16 and depicted in Figure 10, is deterministic, input-complete, progressive and initially-connected, that is, $\mathcal{S} \in \mathcal{IOIP}(L_I, L_U)$. Also, the IUT $\mathcal{I}_\alpha$, constructed in that proof and illustrated in Figure 11, is also deterministic and in the class $\mathcal{IOIP}(L_I, L_U)$. Recall that we write $\overline{y} = 0$ when $y = 1$ and $\overline{y} = 1$ when $y = 0$. The argument now proceeds just as in the proof of Theorem 16, and we get a scheme $\mathcal{T}_\alpha = \langle S_\alpha, t_0, L_I, L_U, T_\alpha\rangle$ in SS and that $\mathcal{I}_\alpha$ does not pass $\mathcal{T}_\alpha$.

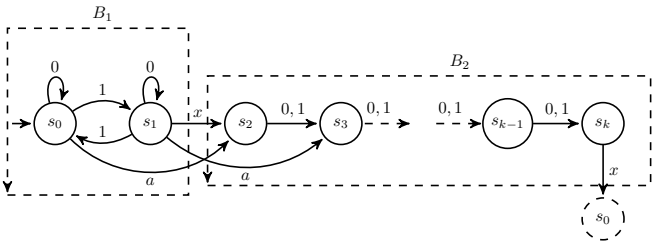


Figure 12: Specification $\mathcal{S}'$ modifying Figure 10.

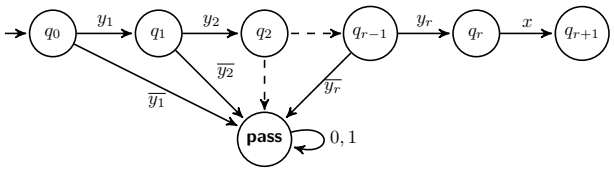


Figure 13: A modified implementation $\mathcal{I}_\alpha$.

We continue as in Theorem 16. Now we have (t_1, x_1, t_2) and (t_1, x_2, t_2') in $\mathcal{T}_\alpha$, with $x_1 \neq x_2$, and $x_1, x_2 \in \{0,1\} = L_I$. The input alphabet for $\mathcal{T}_\alpha$ is $L_I = \{0,1\}$ but, according to Definition 24, any scheme must be single-input. This forces $x_1 = x_2$ and we reach a contradiction again, as in the proof of Theorem 16. As, before, this will imply $\mathcal{T}_\alpha = \mathcal{T}_\beta$ when $\alpha, \beta \in R$, and $\alpha \neq \beta$, with $|\alpha| = |\beta| = r \leq m-3$. From this point on, the argument follows the proof of Theorem 16, establishing that SS must be of size $\Omega(\Phi^m)$.

We would be done if $\mathcal{S} \in \mathfrak{IOMIN}(L_I, L_U)$ and $\mathcal{I}_\alpha \in \mathfrak{IOMIN}(L_I, L_U, k)[m]$, where k is the number of input-states in $\mathcal{S}$. We will now extend Figures 10 and 11 to meet these conditions, while preserving the validity of the previous argument. First note that $\mathcal{S}$ has 3 input-states, whereas $\mathcal{I}_\alpha$ has $r + 3 \leq (m-3) + 3 = m \leq k$ input-states. Next, extend the specification in Figure 10 as in Figure 12, with states $s_3, \ldots, s_k$.

State s_0 is repeated to avoid the clutter. Note that transitions on 0 and 1 out of states s_0 and s_1, as well as the transition on x out of state s_1 were not touched, so that the argument above is still valid when we consider this new specification.

Call this new specification $\mathcal{S}'$. States s_i, $0 \leq i \leq k-1$, are the input-states, so $\mathcal{S}'$ has k input-states. It is also easy to check that $\mathcal{S}'$ is deterministic, input-complete and initially-connected. Moreover, $\mathcal{S}'$ is also progressive, since any cycle in $\mathcal{S}'$ must go through a transition on an input. In order to assert that $\mathcal{S}'$ is in $\mathfrak{IOMIN}(L_I, L_U)$, we need to check that any two input-states in Figure 12 are distinguishable. As indicated therein, states can be partitioned into blocks B_1 and B_2. Consider two distinct states s_i, $s_j \in B_2$ with $2 \leq i < j \leq k$, and let $w = 0^{k-j}$. We see that $(s_i, s_j) \overset{w}{\Rightarrow} (s_\ell, s_k)$ where $\ell = k - (j-i)$ and $2 \leq i \leq \ell \leq k-1$. Since $\mathbf{init}(s_\ell) \cap \mathbf{init}(s_k) = \emptyset$, any two distinct states in B_2 are distinguishable. Then, since $(s_0, s_1) \overset{a}{\Rightarrow} (s_2, s_3)$, it follows that s_0 and s_1 are also distinguishable.

We now argue that s_0 and s_1 are distinguishable from any state $s_i \in B_2$, $2 \leq i \leq k$. Let $w = 0^{k-i}$, so that $(s_0, s_i) \overset{w}{\Rightarrow} (s_0, s_k)$. Since we already know that s_0 is distinguishable from s_k, we conclude that s_0 is distinguishable from any state in B_2. Likewise, with $w = 0^{k-i}x$ we see that $(s_1, s_i) \overset{w}{\Rightarrow} (s_2, s_k)$, so that s_1 is also distinguishable from any state in B_2. Hence, any pair of states in $B_1 \times B_2$ are distinguishable, and we conclude that any two distinct states in $B_1 \cup B_2$ are distinguishable, that is, $\mathcal{S}'$ is input-state-minimal. So, $\mathcal{S}' \in \mathfrak{IOMIN}(L_I, L_U)$ with k input-states, as desired.

We now turn to the IUT. In the proof of Theorem 16, we noted that both transitions from state q_r to state **pass**, together with the self-loops at state q_{r+1} could have been removed, with no prejudice to the argument given therein. See Figure 13, which we also designate by $\mathcal{I}_\alpha$. From the proof of Theorem 16 we have $1 \leq r \leq m-3$. By inspection, we see that $\mathcal{I}_\alpha$ is deterministic, input-complete, progressive, initially-connected, and has $r + 3 \leq (m-3) + 3 = m \leq k$ states. Also, all states, except for q_r and q_{r+1}, are input-states, so that $\mathcal{I}$ has at most k input-states, as needed.

Finally, we show that every pair of distinct input-states of $\mathcal{I}_\alpha$ are distinguishable. Fix some q_j, $0 \leq j \leq r-1$, and define $w = y_{j+1} \cdots y_r$. Clearly, $q_j \overset{w}{\Rightarrow} q_r$ and, since **pass** $\overset{w}{\Rightarrow}$ **pass**, we get $(q_j, \mathbf{pass}) \overset{w}{\Rightarrow} (q_r, \mathbf{pass})$. Since $\mathbf{init}(q_r) \cap \mathbf{init}(\mathbf{pass}) = \emptyset$ we conclude that **pass** is distinguishable from any state q_j, $0 \leq j \leq r-1$. Lastly, take a state q_i distinct from q_j with $0 \leq i < j \leq r-1$. Now we get $q_i \overset{w}{\Rightarrow} q_\ell$ where $\ell = r - j + i = r - (j-i) \leq r-1$. Hence, $(q_i, q_j) \overset{w}{\Rightarrow} (q_\ell, q_r)$ and, because $\ell \leq r-1$, we see that $\mathbf{init}(q_\ell) \cap \mathbf{init}(q_r) = \emptyset$, thus proving that q_i and q_j are also distinguishable. Putting it together, we see that any pair of distinct

input-states of $\mathcal{I}_\alpha$ are distinguishable, that is, $\mathcal{I}_\alpha$ is input-state-minimal. □

Theorem 18 clearly also applies to any specification $\mathcal{S}$ in which the model depicted in Figure 12 appears as a sub-model with state s_0 being reachable from the initial state of $\mathcal{S}$. This is in contrast to Theorem 9 which says that, for a deterministic $\mathcal{S}$ over L, we have an algorithm of time complexity $\mathcal{O}(km)$ for checking m-**ioco**-completeness, where $k = n_S n_L$, $n_L = |L|$ and n_S is the number of states in $\mathcal{S}$, and IUTs have at most m states.

We also remark that in Theorem 1 of Simão and Petrenko [10], implementations are further restricted to be "input-eager", although they do not precisely define this notion in that text. On the other hand, in none of their proofs the input-eager hypothesis explicitly used, leading us to infer that constraining implementations to also be input-eager is a practical consideration to render the testing process more controllable. Since input-eagerness is not strictly necessary to establish their Theorem 1, we conclude that Theorem 18 expresses a valid worst case exponential asymptotic lower bound on the size of the test suites claimed by Theorem 1 in [10].

7 Related Works

IOLTS models are largely used to describe the syntax and the semantics of systems where input and output actions can occur asynchronously, thus capturing a wide class of systems and communication protocols. Several works have studied different aspects of (complete) test suite generation for families of IOLTS models, under various conformance relations. We comment below on some works that are more closely related to our study.

de Vries and Tretmans [14] presented an **ioco**-based testing theory to obtain e-complete test suites. This variant of test suite completeness is based on specific test purposes that share particular properties related to certain testing goals, and they consider only observable behaviors based on some objective criteria when testing black-box IUTs. It turns out that such specific test purposes somewhat limit the fault coverage spectrum, *e.g.*, producing inconclusive verdicts. Large, even infinite, test suites can be produced by their test generation method. Some test selection criteria need to be used to avoid this problem, at least when applied in practical situations. On the other hand, our approach allows for a wider class of IOLTS models, and a low degree polynomial time algorithm are devised for efficiently testing **ioco**-conformance in practical applications.

Petrenko et al. [15] studied IOLTS-testing strategies considering IUTs that cannot block inputs, and also testers that can never prevent an IUT from producing outputs. This scenario calls for input and output communication buffers that can be used for the exchange of symbols between the tester and the IUTs. This leads to an entirely different class of testing strategies, where arbitrarily large buffer memories (queues) are allowed.

Tretmans [5] studied the classic **ioco**-conformance relation for IOLTS models, and developed the foundations of an **ioco**-based testing theory for these models [16], where IUTs are treated as black-boxes, and with a testing architecture where the tester, having no access to the internal structure of IUTs, is seen as an artificial environment that drives the exchange of input and output symbols with the IUT. In this case, some restrictions must be observed by the specification, the IUT and the tester models, such as input-completeness and output-determinism. The algorithms developed therein, however, may in general lead to infinite test suites, making it more difficult to devise solutions for practical applications. In our work we described a method that, considering the exact same restrictions to the IOLTS models, does in fact generate finite sets of test purposes that can be used in practical situations. In rare situations, the algorithm may lead to exponential sized testers. If the same restrictions are to be obeyed by the specification, the IUT and the tester IOLTS models, we established an exponential worst case asymptotic lower bound on the size of the testers. This shows with those restrictions in order, generating exponential sized testers is, in unavoidable general, being rather an intrinsic to the problem when one requires **ioco**-completeness.

Simão and Petrenko [10] also described an approach to generate finite **ioco**-complete test suites for a class of IOLTS models. They, however, also imposed a number of restrictions on the specification and IUT models in order to obtain **ioco**-complete finite test suites. They assumed the test purposes to be single-input and also output-complete. Moreover, specifications and IUTs must be input-complete, progressive, and initially-connected, so further restricting the class of IOLTS models that can be tested according to their fault model. They also did not study the complexity of their method for generating **ioco**-complete test suites under those restrictions. In contrast, we applied our approach to a testing architecture that satisfies the same restrictions, and showed how to generate **ioco**-complete test suites in a more straightforward manner. Further, we examined the complexity of the problem of generating **ioco**-complete test suites under the same restrictions, and established an exponential worst case asymptotic lower bound on the size of any **ioco**-complete test suite that can be generated in this situation.

Noroozi et al. [17] presented a polynomial time reduction from a variation of the SAT problem to the problem of checking **ioco**-completeness, thus establishing that, under very general assumptions about the IOLTS models — including non-determinism, — that checking **ioco**-completeness is a PSPACE-complete problem. In a more restricted scenario, treating only deterministic and input-enabled IOLTS models, they proposed a polynomial time algorithm, based on a simulation-like preorder relation. This is the same complexity bound that our method attains but, in contrast, our approach treats a wider class of conformance relations not being restricted to **ioco**-conformance only. In another work, Noroozi et al. [18] also studied the problem of synchronous and asynchronous conformance testing, when allowing communication channels as auxiliary memories. They treated a more restricted class of IOLTS, the so-called Internal Choice IOLTSs, where quiescent states must be also input-enabled. The notions of **ioco**-conformance and of traces are also more restricted. When the structure of IUTs are accessible, algorithms to generate test cases are shown to be sound and exhaustive. However, in a setting where IUTs are black-boxes these algorithms are not applicable, thus limiting their practical use.

Roehm et al. [19], in a more recent work, introduced a variation of conformance testing, related to safety properties. Despite being a weaker relation than trace-inclusion conformance, it allows for tunning a trade-off between accuracy and computational complexity when checking conformance of hybrid systems. Instead of verifying the whole system, their approach searches for counter-examples. They also proposed a test selection algorithm that uses a coverage measure to reduce the number of test cases. However, since the models are hybrid, the continuous flow of time forces discretizations to reduce the test generation problem to discrete models. This imposes a trade-off between accuracy and computational load, which must be tuned by appropriate choices related to some over-approximations.

Other works have considered **ioco**-based testing for compositional systems, where components of a more complex system can be formally tested using composition operators to capture the resulting behavior of multiple components. Benes et al. [20] have proposed merge and quotient operators in order to check consistency of more complex parts of systems under test. Following a similar line, Daca et al. [21] proposed compositional operators, friendly composition and hiding, applied to an **ioco**-testing theory in order to minimize the integration of testing efforts. The result of the friendly composition is an overall specification that integrates the component specifications while pruning away any inputs that lead to incompatible interactions between the components. The friendly hiding operation can prune inputs that lead to states which are ambiguous with respect to underspecified parts of the system. In a similar vein, Frantzen and Tretmans [22] presented a method when complete behaviors of components of a system are not available, and applied a parallel operator when integrating different components. They proposed a specific conformance relation for the components and devised an algorithm that constructs complete test suites.

8 Conclusions

Conformance of an IUT to a specification often needs to be checked, in order to establish a mathematical guarantee of correctness of the IUTs. The **ioco** framework has been the conformance relation of choice for verifying IOLTS models in several testing architectures.

We addressed the problem of conformance testing and test case generation for asynchronous systems that can be described using IOLTS models as the base formalism. A new notion of conformance relation was studied, one that is more general and encompasses the classic **ioco**-conformance. It opened the possibility for a much wider class of conformance relations, all uniformly treated under the same formalism. In particular, it allows for properties or fault models to be specified by formal languages, *e.g.*, regular languages. Very few restrictions over the IOLTS models must be satisfied when generating finite and complete test suites under any notion of conformance that fits the more general setting studied herein. We also proved correct a polynomial algorithm to test general conformance in a "white-box" architecture. With a fixed specification, the algorithm runs in linear time on the size of the IUTs.

Equipped with this new notion of conformance, we specialized the test generation process in order to cover other special cases of conformance relations, such as classical **ioco**-conformance. Complexity issues related to complete test suite generation for verifying **ioco**-conformance in settings where the IOLTS models were under several specific restrictions were also discussed. For some sets of restrictions we showed that the state explosion problem cannot be avoided, in general, forcing **ioco**-complete test suites to grow exponentially with the size of the IUTs. In these cases, we proved correct general algorithms with time complexities that attained such lower bounds, while still generating complete test suites. This indicates that other families of specialized IOLTS models could be considered by our approach, leading to similar results.

Other research areas that might be inspired by these ideas are symbolic test case generation, where data variables and parameters are also present [23, 24], as well as conformance relations and generation methods for real-time systems [25, 26].

References

[1] A. L. Bonifacio, A. V. Moura, and A. Simao, "Model Partitions and Compact Test Case Suites," *International Journal of Foundations of Computer Science*, vol. 23, no. 01, pp. 147–172, 2012.

[2] R. Cardell-Oliver, "Conformance tests for real-time systems with timed automata specifications," *Formal Aspects of Computing*, vol. 12, no. 5, pp. 350–371, 2000. [Online]. Available: citeseer.ist.psu.edu/385816.html

[3] R. Dorofeeva, K. El-Fakih, and N. Yevtushenko, "An improved conformance testing method," in *FORTE*, F. Wang, Ed., 2005, pp. 204–218.

[4] G. J. Tretmans, "A formal approach to conformance testing," Ph.D. dissertation, University of Twente, Enschede, December 1992.

[5] J. Tretmans, "Model based testing with labelled transition systems," in *Formal Methods and Testing*, 2008, pp. 1–38.

[6] A. Gargantini, "Conformance testing," in *Model-Based Testing of Reactive Systems: Advanced Lectures*, ser. Lecture Notes in Computer Science, M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, Eds., vol. 3472. Springer-Verlag, 2005, pp. 87–111.

[7] J. Tretmans, "Test generation with inputs, outputs, and quiescence." in *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings*, ser. Lecture Notes in Computer Science, T. Margaria and B. Steffen, Eds., vol. 1055. Springer, 1996, pp. 127–146.

[8] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. McMinn, and others, "An orchestrated survey of methodologies for automated software test case generation," *Journal of Systems and Software*, vol. 86, no. 8, pp. 1978–2001, 2013.

[9] S. J. Cunning and J. W. Rozenblit, "Automating test generation for discrete event oriented embedded systems," *J. Intell. Robotics Syst.*, vol. 41, no. 2-3, pp. 87–112, 2005.

[10] A. Simão and A. Petrenko, "Generating complete and finite test suite for ioco is it possible?" in *Ninth Workshop on Model-Based Testing (MBT 2014)*, 2014, pp. 56–70.

[11] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Comutation.* Addison Wesley, 19.

[12] M. Rabin and D. Scott, "Finite automata and their decision problems," *IBM Journal of Research and Development*, vol. 3, no. 2, pp. 114–125, 1959.

[13] A. Gill, *Introduction to the theory of finite-state machines.* New York: McGraw-Hill, 1962.

[14] R. de Vries, "Towards formal test purposes," in *Formal Approaches to Testing of Software 2001 (FATES'01) - Volume NS-01-4*, ser. BRICS Notes Series, G. Tretmans and H. Brinksma, Eds., Aarhus, Denkmark, August 2001, pp. 61–76. [Online]. Available: http://epdoc.utsp.utwente.nl/66272/

[15] A. Petrenko, N. Yevtushenko, and J. L. Huo, "Testing transition systems with input and output testers," in *TESTERS, PROC TESTCOM 2003, SOPHIA ANTIPOLIS.* Springer-Verlag, 2003, pp. 129–145.

[16] G. J. Tretmans, "Test generation with inputs, outputs and repetitive quiescence," Centre for Telematics and Information Technology University of Twente, Enschede, Technical Report TR-CTIT-96-26, 1996.

[17] N. Noroozi, M. R. Mousavi, and T. A. C. Willemse, "On the complexity of input output conformance testing," in *Formal Aspects of Component Software*, J. L. Fiadeiro, Z. Liu, and J. Xue, Eds. Cham: Springer International Publishing, 2014, pp. 291–309.

[18] N. Noroozi, R. Khosravi, M. R. Mousavi, and T. A. C. Willemse, "Synchrony and asynchrony in conformance testing," *Software and System Modeling*, vol. 14, no. 1, pp. 149–172, 2015. [Online]. Available: https://doi.org/10.1007/s10270-012-0302-8

[19] H. Roehm, J. Oehlerking, M. Woehrle, and M. Althoff, "Reachset conformance testing of hybrid automata," in *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '16. New York, NY, USA: ACM, 2016, pp. 277–286. [Online]. Available: http://doi.acm.org/10.1145/2883817.2883828

[20] N. Benes, P. Daca, T. A. Henzinger, J. Kretinsky, and D. Nickovic, "Complete composition operators for ioco-testing theory," in *2015 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2015, pp. 101–110. [Online]. Available: https://doi.ieeecomputersociety.org/10.1145/2737166.2737175

[21] P. Daca, T. A. Henzinger, W. Krenn, and D. Nickovic, "Compositional specifications for ioco testing," in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*, March 2014, pp. 373–382.

[22] L. Frantzen and J. Tretmans, "Model-based testing of environmental conformance of components," in *Formal Methods for Components and Objects*, F. S. de Boer, M. M. Bonsangue, S. Graf, and W.-P. de Roever, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 1–25.

[23] C. Gaston, P. Le Gall, N. Rapin, and A. Touil, *Symbolic Execution Techniques for Test Purpose Definition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–18.

[24] V. Rusu, L. du Bousquet, and T. Jéron, *An Approach to Symbolic Test Generation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 338–357.

[25] L. B. Briones and E. Brinksma, *A Test Generation Framework for quiescent Real-Time Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 64–78.

[26] M. Krichen, "Model-based testing for real-time systems," Ph.D. dissertation, Université Joseph Fourier, 2007.

Appendix

Here we discuss, in detail, the relationship between the **ioco** relation described in this work and a classical **ioco** relation used in the literature [5]. We want to establish that both of these variations describe the same **ioco** relation. Since the precise definitions of several notions in this work and in the original proposal [5] differ slightly, we need to proceed step by step, comparing the same notions as defined in both texts. The differences are most marked when the notion of quiescence is treated in both texts, and so, special care must be taken when comparing notions related to quiescence.

From now on, if X denotes any object defined both in [5] and in this work, we let X_T be the variation of X as defined in [5], and we use X_A for the same object X as defined in this work. For instance, $\rightarrow_T$ is the trace relation in LTSs as defined in [5], and $\rightarrow_A$ is the trace relation as defined in this work. In many cases they will be exactly the same, but in some other cases there might be a slight variation between the two relations.

8.1 The general model

We first note that an LTS model as defined in [5] is denoted by $\langle S, L, T, s_0\rangle$, whereas an LTS model is here denoted as $\langle S, s_0, L, T\rangle$. Further, in [5] LTS models can be infinite objects, whereas we deal only with finite models. We will from now on restrict ourselves to finite models only.

Hypothesis 19 *We assume that all LTS models are finite.*

The notions of a path, $\rightarrow$, and of an observable path, $\Rightarrow$, appear as Definitions 3 and 4 in [5]. Here see also Definition 4. The following is immediate.

Proposition 12 $\rightarrow_T = \rightarrow_A$ *and also* $\Rightarrow_T = \Rightarrow_A$.

Proof Follows from the definitions. □

We now write $\rightarrow$ for both $\rightarrow_T$ and $\rightarrow_A$. Likewise, we write $\Rightarrow$ for both $\Rightarrow_T$ and $\Rightarrow_A$.

The function **after** appears in Definition 5(3) in [5]. See our Definition 13(2).

Proposition 13 *Let* $\mathcal{S} = \langle S, s_0, L, T\rangle$ *be an LTS. For all* $p \in S$, $\sigma \in L_\tau^\star$ *we have*

$$p \textbf{ after}_T\, \sigma = \{q \mid p \stackrel{\sigma}{\Rightarrow} q\} = p \textbf{ after}_A\, \sigma.$$

Proof Immediate from Proposition 12. □

From now on we may write **after** for both $\textbf{after}_T$ and $\textbf{after}_A$.

In order to leave no room for confusion we let $\mathcal{LTS}_A(L)$ be the class of all LTS models over the alphabet L as defined here. We designate by $\mathcal{LTS}_T(L)$ the class of all LTS models according to Definition 5(12) [5]. As a refinement, the class of all models $\mathcal{S} = \langle S, s_0, L, T\rangle$ in $\mathcal{LTS}_T(L)$ where all states are reachable from the initial state, that is, $s_0 \rightarrow s$ for all $s \in S$, will be designated as $\mathcal{LTSR}_T(L)$.

Each model in $\mathcal{LTS}_T(L)$ is assumed to be image finite and strongly converging (Definition 5(12) [5]), where an LTS $\mathcal{S} = \langle S, s_0, L, T\rangle$ is said to be

1. **image finite** when p **after** σ is finite, for all $p \in S$ and all $\sigma \in L_\tau^\star$ (see its Definition 5(10)).
2. **strongly converging** if there is no state that can perform an infinite sequence of internal transitions (see its Definition 5(11)).

We readily have the following result.

Proposition 14 *Assume hypothesis 19. Then*

1. $\mathcal{LTS}_A(L) \subsetneq \mathcal{LTS}_T(L)$ *(properly contained)*
2. $\mathcal{LTSR}_T(L) = \mathcal{LTS}_A(L)$

Proof Let $\mathcal{S} = \langle S, s_0, L, T\rangle$ be an LTS in $\mathcal{LTS}_A(L)$. Under Hypothesis 19, $\mathcal{S}$ is image finite. According to Remark 1, there is no transition (s, τ, s) in T, and so $\mathcal{S}$ is also strongly converging. This proves (1).

For (2), let $\mathcal{S} = \langle S, s_0, L, T\rangle$ be an LTS in $\mathcal{LTSR}_T(L)$. Since $\mathcal{S}$ is strongly converging, there can be no transition (s, τ, s) in T. From the definition of the class $\mathcal{LTSR}_T(L)$ we know that for all $s \in S$ we must have $s_0 \rightarrow s$. We conclude that Remark 1 is satisfied and so $\mathcal{S} \in \mathcal{LTS}_A(L)$. Hence, $\mathcal{LTSR}_T(L) \subseteq \mathcal{LTS}_A(L)$. Using item (1) we conclude the proof. □

8.2 Models with inputs and outputs

Let $L = L_I \cup L_U$ with $L_I \cap L_U = \emptyset$ be alphabets. Then, Definition 10 says that (S, s_0, L_I, L_U, T) is an IOLTS with input alphabet L_I and output alphabet L_U when $(S, s_0, L, T) \in \mathcal{LTS}_A(L)$, that is, an IOLTS is a LTS where the alphabet has been partitioned into disjoint sets of input and output action symbols. In this appendix we will designate the class of IOLTSs over L_I and L_U by $\mathcal{IO}_A(L_I, L_U)$. Likewise, in [5], Definition 6 says that a labeled transition system with inputs and outputs is a system (S, L_I, L_U, T, s_0), where $(S, L, T, s_0) \in \mathcal{LTS}_T(L)$, and we will here denote the class of all such models by $\mathcal{LTS}_T(L_I, L_U)$.

Proposition 15 *Again, assume hypothesis 19. Then*

1. $\mathcal{IO}_A(L_I, L_U) \subsetneq \mathcal{LTS}_T(L_I, L_U)$ *(properly contained)*
2. $\mathcal{LTSR}_T(L_I, L_U) = \mathcal{IO}_A(L_I, L_U)$

Proof Immediately from the definitions and from the Proposition 14. □

8.3 Quiescent states

First we introduce some notation. Let A be any alphabet, and define $A^{+\delta} = A \cup \{\delta\}$ and $A^{-\delta} = A - \{\delta\}$.

Let $\mathcal{S} = (S, s_0, L_I, L_U, T) \in \mathcal{LTS}_T(L_I, L_U)$ be an IOLTS. In [5], Definition 8.1 says that a state $s \in S$ is quiescent (in $\mathcal{S}$), denoted $\delta_T^{\mathcal{S}}(s)$, if for all $x \in L_U \cup \{\tau\}$ we have $s \overset{x}{\not\to}$ in $\mathcal{S}$. When the model is clear from the context we may write only δ_T instead of $\delta_T^{\mathcal{S}}$. Further, given an IOLTS $\mathcal{S} = (S, s_0, L_I, L_U^{-\delta}, T) \in \mathcal{LTS}_T(L_I, L_U^{-\delta})$, Definition 9 [5] creates a new model $\mathcal{S}^\delta = \langle S, s_0, L_I, L_U^{+\delta}, T \cup T_\delta \rangle$, where $T_\delta = \{(s, \delta, s) \mid \delta_T^{\mathcal{S}}(s)\}$. We designate this new class of IOLTSs by $\mathcal{LTS}_T^\delta(L_I, L_U^{-\delta}) = \{\mathcal{S}^\delta \mid \mathcal{S} \in \mathcal{LTS}_T(L_I, L_U^{-\delta})\}$ to stress that these models were constructed from IOLTSs whose output alphabet did not contain δ (so $L_U^{-\delta}$ in the notation), but the output alphabet of the extended model always contains δ (so $\mathcal{LTS}_T^\delta()$ in the notation.) If the original model was from the class $\mathcal{LTSR}_T(L_I, L_U^{-\delta})$, meaning that all states are reachable from the initial state, then the new class of models will be designated by $\mathcal{LTSR}_T^\delta(L_I, L_U^{-\delta})$.

Next proposition states a simple result.

Proposition 16 $\mathcal{LTS}_T^\delta(L_I, L_U^{-\delta}) \subsetneq \mathcal{LTS}_T(L_I, L_U^{+\delta})$ *and* $\mathcal{LTSR}_T^\delta(L_I, L_U^{-\delta}) \subsetneq \mathcal{LTSR}_T(L_I, L_U^{+\delta})$ *(properly contained).*

Proof Immediate from the definitions. □

Now we turn to Definition 17 where quiescence in δ-IOLTS models is introduced in this work. To emphasize that δ is always a symbol in the output alphabet of an δ-IOLTS, in this appendix we designate this class of models by $\mathcal{IO}_A^\delta(L_I, L_U^{+\delta})$ (in the main text it was designated simply by $\mathcal{IO}_\delta(L_I, L_U)$). Let $\mathcal{S} = (S, s_0, L_I, L_U^{+\delta}, T) \in \mathcal{IO}_A^\delta(L_I, L_U^{+\delta})$ be a δ-IOLTS. If $q \in S$ is quiescent according to Definition 17, we write $\delta_A^{\mathcal{S}}(q)$, and may omit the index $^{\mathcal{S}}$ when no confusion can arise.

Proposition 17 is important because it gives the exact relationship between the class of all δ-IOLTS models and the class of all models after they are extended in order to include quiescence, as defined in [5].

Proposition 17 *Assume hypothesis 19, then we have that*

1. $\mathcal{IO}_A^\delta(L_I, L_U^{+\delta}) \subsetneq \mathcal{LTS}_T^\delta(L_I, L_U^{-\delta})$ *(properly contained)*
2. $\mathcal{LTSR}_T^\delta(L_I, L_U^{-\delta}) = \mathcal{IO}_A^\delta(L_I, L_U^{+\delta})$

Proof To prove item (1), assume $\mathcal{S} \in \mathcal{IO}_A^\delta(L_I, L_U^{+\delta})$. From the definition of the class $\mathcal{IO}_A^\delta(L_I, L_U^{+\delta})$ we obtain $\mathcal{S} = \langle S, s_0, L_I, L_U^{+\delta}, T \rangle$, where

$$\mathcal{S} \in \mathcal{IO}_A(L_I, L_U^{+\delta}); \text{ and} \tag{1}$$

$$(s, \delta, q) \in T \quad \text{if and only if} \tag{2}$$

(a) $s = q$;

(b) when $s \xrightarrow{x}$ with $x \in L_I \cup L_U^{+\delta} \cup \{\tau\}$, then $x \in L_I \cup \{\delta\}$.

From the definition of the class $\mathcal{LTS}_T^\delta(L_I, L_U^{-\delta})$, we need to show that $\mathcal{S} = \mathcal{T}^\delta$ for some $\mathcal{T} \in \mathcal{LTS}_T(L_I, L_U^{-\delta})$. Take $\mathcal{T} = \langle S, s_0, L_I, L_U^{-\delta}, R \rangle$, where

$$R = T - \{(s, \delta, q) \mid s, q \in S\}. \tag{3}$$

Since there are no δ-transitions in R, the output alphabet of $\mathcal{T}$ can be taken as $L_U^{-\delta}$. Since $\mathcal{S} \in \mathcal{IO}_A^\delta(L_I, L_U^{+\delta})$, we get $\mathcal{T} \in \mathcal{IO}_A(L_I, L_U^{-\delta})$. From Proposition 15 we have $\mathcal{T} \in \mathcal{LTS}_T(L_I, L_U^{-\delta})$. It remains to show that $\mathcal{S} = \mathcal{T}^\delta$.

From the definitions we have $\mathcal{T}^\delta = \langle S, s_0, L_I, L_U^{+\delta}, R \cup R_\delta \rangle$, where $R_\delta = \{(s, \delta, s) \mid \delta_T^{\mathcal{T}}(s)\}$. In order to complete the proof of item (1), we need show that $R \cup R_\delta = T$.

Let $(s, x, q) \in R$. From the Eq. (3) we have $(s, x, q) \in T - \{(s, \delta, q) \mid s, q \in S\}$, and then $(s, x, q) \in T$.

Let $(s, x, q) \in R_\delta$. From definition of the class $\mathcal{LTS}_T^\delta(L_I, L_U^{-\delta})$ we have that $s = q$ and $x = \delta$. So, $\delta_T^{\mathcal{T}}(s)$ in $\mathcal{T}$. Now assume $s \in S$ and $s \xrightarrow{y}$ with $y \in L_I \cup L_U^{+\delta} \cup \{\tau\}$. Since we have $\delta_T^{\mathcal{T}}(s)$ in $\mathcal{T}$, we get $y \notin L_U^{-\delta} \cup \{\tau\}$. Hence $y \in L_I \cup \{\delta\}$. From Eq. (2), we obtain $(s, x, q) = (s, \delta, s) \in T$ and then we conclude that $R \cup R_\delta \subseteq T$.

In order to prove $T \subseteq R \cup R_\delta$, take $(s, x, q) \in T$ with $x \neq \delta$. From Eq. (3) we get $(s, x, q) \in R$, and then $(s, x, q) \in R \cup R_\delta$. Now assume that $(s, \delta, q) \in T$. From Eq. (2) we have $s = q$ and the following condition is satisfied in s: if $s \xrightarrow{x}$ with $x \in L_I \cup L_U^{+\delta} \cup \{\tau\}$ then $x \in L_I \cup \{\delta\}$. Next, we show that $\delta_T^{\mathcal{T}}(s)$ is in $\mathcal{T}$, which gives $(s, \delta, s) \in R_\delta$, and then $(s, \delta, q) = (s, \delta, s) \in R \cup R_\delta$. This will imply $T \subseteq R \cup R_\delta$, so that $T = R \cup R_\delta$, completing the proof.

So, assume that we do not have $\delta_T^{\mathcal{T}}(s)$ in $\mathcal{T}$. From the definition we would have $s \xrightarrow{x}$ with $x \in L_U^{-\delta} \cup \{\tau\}$. Therefore, $x \in L_I \cup L_U^{+\delta} \cup \{\tau\}$. From Eq. (2) we have $x \in L_I \cup \{\delta\}$, which contradicts with $x \in L_U^{-\delta} \cup \{\tau\}$. Hence, $\delta_T^{\mathcal{T}}(s)$ in $\mathcal{T}$, and the proof is complete.

Now we turn to item (2). Let $\mathcal{S} \in \mathcal{IO}_A^\delta(L_I, L_U^{+\delta})$. From item (1) we get $\mathcal{S} \in \mathcal{LTS}_T^\delta(L_I, L_U^{-\delta})$. Since every state of $\mathcal{S}$ is reachable from its initial state, it follows that $\mathcal{S} \in \mathcal{LTSR}_T^\delta(L_I, L_U^{-\delta})$, and we can conclude that $\mathcal{IO}_A^\delta(L_I, L_U) \subseteq \mathcal{LTSR}_T^\delta(L_I, L_U^{-\delta})$.

Now we prove that $\mathcal{LTSR}_T^\delta(L_I, L_U^{-\delta}) \subseteq \mathcal{IO}_A^\delta(L_I, L_U^{+\delta})$. Let $\mathcal{S} \in \mathcal{LTSR}_T^\delta(L_I, L_U^{-\delta})$ be an IOLTS. We want to show that $\mathcal{S} \in \mathcal{IO}_A^\delta(L_I, L_U^{+\delta})$. From the definition of $\mathcal{LTSR}_T^\delta(L_I, L_U^{-\delta})$ we know that $\mathcal{S} = \mathcal{T}^\delta$ for some $\mathcal{T} = \langle S, s_0, L_I, L_U^{-\delta}, T \rangle \in \mathcal{LTSR}_T(L_I, L_U^{-\delta})$. From the definition we also know that $\mathcal{T}^\delta = \langle S, s_0, L_I, (L_U^{-\delta})^{+\delta}, T \cup T_\delta \rangle$, where $T_\delta = \{(s, \delta, s) \in T \mid \delta_T^{\mathcal{T}}(s)\}$. We want to show that $\mathcal{S} = \mathcal{T}^\delta \in \mathcal{IO}_A^\delta(L_I, L_U^{+\delta})$. From the definition of the class $\mathcal{IO}_A^\delta(L_I, L_U^{+\delta})$, this is equivalent to show that

$$\mathcal{T}^\delta \in \mathcal{IO}_A(L_I, L_U^{+\delta}); \text{ and} \tag{4}$$

$$(s, \delta, q) \in T \cup T_\delta \quad \text{if and only if} \tag{5}$$

(a) $s = q$;

(b) if $s \xrightarrow{x}$ with $x \in L_I \cup L_U^{+\delta} \cup \{\tau\}$, then $x \in L_I \cup \{\delta\}$.

Since $\mathcal{T} = \langle S, s_0, L_I, L_U^{-\delta}, T \rangle$, and $(L_U^{-\delta})^{+\delta} = L_U^{+\delta}$, we have that $\mathcal{T}^\delta \in \mathcal{LTSR}_T(L_I, L_U^{+\delta})$. From Proposition 15(2) we obtain $\mathcal{T}^\delta \in \mathcal{IO}_A(L_I, L_U^{+\delta})$ and the Eq. (4) holds.

It remains to prove Eq. (5). Let $(s, \delta, q) \in T \cup T_\delta$. Since $\mathcal{T} = \langle S, s_0, L_I, L_U^{-\delta}, T \rangle$ and $\delta \notin L_U^{-\delta}$, it follows that $(s, \delta, q) \notin T$ and so $(s, \delta, q) \in T_\delta$. We then have $s = q$ and $\delta_T^{\mathcal{T}}(s)$ in $\mathcal{T}$. Therefore, Eq. (5a) holds. Now we assume $s \xrightarrow{x}$ with $x \in L_I \cup L_U^{+\delta} \cup \{\tau\}$. Since $(s, \delta, s) \in T_\delta$ and $\delta_T^{\mathcal{T}}(s)$ in $\mathcal{T} = \langle S, s_0, L_I, L_U^{-\delta}, T \rangle$, we should have $x \notin L_U^{-\delta} \cup \{\tau\}$. Then, $x \in L_I \cup \{\delta\}$ and Eq. (5b) also holds. □

Now Proposition 18 states quiescence and relate them between the approaches.

Proposition 18 *Let $\mathcal{S} = \langle S, s_0, L_I, L_U^{+\delta}, T \rangle \in \mathcal{IO}_A^\delta(L_I, L_U^{+\delta})$. We then have that $\mathcal{S} = \mathcal{T}^\delta$ where $\mathcal{T} = \langle S, s_0, L_I, L_U^{-\delta}, R \cup R_\delta \rangle \in \mathcal{LTSR}_T(L_I, L_U^{-\delta})$. Further, for all $s \in S$ we get $\delta_A^{\mathcal{S}}(s)$ in $\mathcal{S}$ if, and only if, $\delta_T^{\mathcal{T}}(s)$ in $\mathcal{T}$.*

Proof From Proposition 17(2) we have that $\mathcal{IO}_A^\delta(L_I, L_U^{+\delta}) = \mathcal{LTSR}_T^\delta(L_I, L_U^{-\delta})$. The definition of the class $\mathcal{LTS}_T^\delta(L_I, L_U^{-\delta})$ gives $\mathcal{S} = \mathcal{T}^\delta$, where $\mathcal{T} = \langle S, s_0, L_I, L_U^{-\delta}, R \rangle \in \mathcal{LTSR}_T(L_I, L_U^{-\delta})$, with $T = R \cup R_\delta$ and $R_\delta = \{(p, \delta, p) \mid \delta_T^{\mathcal{T}}(p) \text{ in } \mathcal{T}\}$.

Assume that $\delta_A^{\mathcal{S}}(s)$ holds in $\mathcal{S}$ and $\delta_T^{\mathcal{T}}(s)$ does not hold in $\mathcal{T}$. Definition of $\delta_T^{\mathcal{T}}(\cdot)$ gives that $s \xrightarrow{x}$ in $\mathcal{T}$ with $x \in L_U^{-\delta} \cup \{\tau\}$. So, $(s, x, p) \in R$ for some $p \in S$, and then $(s, x, p) \in T$. Therefore, $s \xrightarrow{x}$ in $\mathcal{S}$. Since we have $\delta_A^{\mathcal{S}}(s)$ in $\mathcal{S}$, the definition of $\delta_A^{\mathcal{S}}(\cdot)$ together with $s \xrightarrow{x}$ in $\mathcal{S}$ gives that $x \in L_I \cup \{\delta\}$, contradicting $x \in L_U^{-\delta} \cup \{\tau\}$.

On the other direction, we assume $\delta_T^{\mathcal{T}}(s)$ em $\mathcal{T}$. From the definition of R_δ we have $(s, \delta, s) \in R_\delta$. Hence $(s, \delta, s) \in T$, i.e., $s \xrightarrow{\delta}$ in $\mathcal{S}$. From the definition of $\delta_A^{\mathcal{S}}(\cdot)$ we also have $\delta_A^{\mathcal{S}}(s)$, concluding the proof. □

8.4 The out relation

In [5], Definition 11 introduces the **out** relation, here denoted $\mathbf{out}_T$, thus: Let $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle \in \mathcal{LTS}_T(L_I, L_U)$. Then, for all $s \in S$ and all $Q \subseteq S$,

$$\mathbf{out}_T(s) = \{x \in L_U | s \xrightarrow{x}\} \cup \{\delta | \delta^{\mathcal{S}}_T(s)\}, \text{ and}$$
$$\mathbf{out}_T(Q) = \bigcup \{out_T(s) | s \in Q\}.$$

In this work, we define the same relation, denoted $\mathbf{out}_A$, as follows: Let $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle \in \mathcal{IO}_A(L_I, L_U)$. Then, for all $Q \subseteq S$

$$\mathbf{out}_A(Q) = \bigcup_{s\in Q} \{x \in L_U \,|\, s \overset{x}{\Rightarrow}\}.$$

See Definition 13.

Next we show that both definitions of **out** coincide.

Proposition 19 *Let $\mathcal{S} = (S, s_0, L_I, L_U^{+\delta}, T) \in \mathcal{IO}^{\delta}_A(L_I, L_U^{+\delta})$ be an IOLTS. Then, we have that $\mathbf{out}_A(Q) = \mathbf{out}_T(Q)$ for all $Q \subseteq S$.*

Proof We show that $\mathbf{out}_A(s) = \mathbf{out}_T(s)$ for all $s \in S$.

From Proposition 12 we have $\Rightarrow_T = \Rightarrow_A$, and so the indexes may be omitted. Likewise, we can write $\rightarrow$ instead of $\rightarrow_T$ or $\rightarrow_A$.

First assume $x \in \mathbf{out}_A(s)$ and $x \notin \mathbf{out}_T(s)$. Since $\mathcal{S} \in \mathcal{IO}^{\delta}_A(L_I, L_U^{+\delta})$ it is clear that $\mathcal{S} \in \mathcal{IO}_A(L_I, L_U^{+\delta})$. Hence, from the definition of $x \in \mathbf{out}_A(s)$ we obtain $s \overset{x}{\Rightarrow}$ and $x \in L_U^{+\delta}$. So, we also have $s \xrightarrow{x}$. From Proposition 17(1) we have $\mathcal{IO}^{\delta}_A(L_I, L_U^{+\delta}) \subsetneq \mathcal{LTS}^{\delta}_T(L_I, L_U^{-\delta})$. Hence, from Proposition 16 we get $\mathcal{S} \in \mathcal{LTS}_T(L_I, L_U^{+\delta})$. Since $s \xrightarrow{x}$ and $x \in L_U^{+\delta}$, from the definition of $\mathbf{out}_T$ we obtain $x \in \mathbf{out}_T(s)$, contradicting $x \notin \mathbf{out}_T(s)$. Then $\mathbf{out}_A(s) \subseteq \mathbf{out}_T(s)$.

Now assume that $x \in \mathbf{out}_T(s)$ and $x \notin \mathbf{out}_A(s)$. Since $\mathcal{S} \in \mathcal{LTS}_T(L_I, L_U^{+\delta})$, from $x \in \mathbf{out}_T(s)$ we get: (i) $\delta^{\mathcal{S}}_T(s)$, or (ii) $s \xrightarrow{x}$ and $x \in L_U^{+\delta}$. First assume (i). From the definition of $\delta^{\mathcal{S}}_T(\cdot)$ we know that for all $x \in L_I \cup L_U^{+\delta} \cup \{\tau\}$, if $(s, x, p) \in T$ then $x \notin L_U^{+\delta} \cup \{\tau\}$, i.e., we must have $x \in L_I$. Since $\mathcal{S} \in \mathcal{IO}^{\delta}_A(L_I, L_U^{+\delta})$, Definition 17 for δ-IOLTS models implies $s = p$ and $(s, \delta, s) \in T$. But such transition contradicts $\delta \notin L_U^{+\delta} \cup \{\tau\}$. Then hypothesis (i) does not hold. Next, assuming (ii) we must have $s \xrightarrow{x}$ and $x \in L_U^{+\delta}$. Thus $s \overset{x}{\Rightarrow}$ and $x \in L_U^{+\delta}$. Since $\mathcal{S} \in \mathcal{IO}_A(L_I, L_U^{+\delta})$, the definition of $\mathbf{out}_A$ results in $x \in \mathbf{out}_A(s)$, contradicting $x \notin \mathbf{out}_A(s)$. Hence hypothesis (ii) does not hold. We conclude that $\mathbf{out}_T \subseteq \mathbf{out}_A$. □

8.5 Input-enabledness property

In [5], Definition 7 says that the class of all input-output transition systems with inputs in L_I and outputs in L_U is a restricted subclass of $\mathcal{LTS}_T(L_I \cup L_U)$. More specifically, $(S, L_I, L_U, T, s_0) \in \mathcal{LTS}_T(L_I \cup L_U)$ is an input-output transition system if any reachable state s is input-enabled, that is, there is a transition out of s for all input symbols. More formally, in Definition 5(6) we have $der_T(p) = \{q \,|\, p \Rightarrow q\}$. Then Definition 7 decrees that $(S, L_I, L_U, T, s_0) \in \mathcal{LTS}_T(L_I \cup L_U)$ is an input-output transition system when for all $p \in der_T(s_0)$ we have $p \overset{a}{\Rightarrow}$ for all $a \in L_I$. The class of all input-output systems is here denoted by $\mathcal{IOTS}_T(L_I, L_U)$. The subclass of all input-output systems $\mathcal{S}$ where all states are reachable from the initial states, that is when $\mathcal{S}$ is in the subclass $\mathcal{LTSR}_T(L_I, L_U)$, will be designated by $\mathcal{IOTSR}_T(L_I, L_U)$.

In our work, Definition 21 says that an IOLTS $(S, s_0, L_I, L_U, T) \in \mathcal{IO}_A(L_I, L_U)$ is input-enabled when $\mathbf{inp}(s) = L_I$ for all $s \in S$. In the same definition we find that $\mathbf{inp}(s) = \{\ell \in L_I \,|\, s \overset{\ell}{\Rightarrow}\}$. In this appendix we designate by $\mathcal{IOE}_A(L_I, L_U)$ the class of all input-enabled IOLTS models over input alphabet L_I and output alphabet L_U.

Proposition 20 *Under Hypothesis 19 we have*

1. $\mathcal{IOE}_A(L_I, L_U) \subsetneq \mathcal{IOTS}_T(L_I, L_U)$ *(properly contained)*
2. $\mathcal{IOTSR}_T(L_I, L_U) = \mathcal{IOE}_A(L_I, L_U)$

Proof First note $\mathbf{inp}(p) = L_I$ if and only if $p \overset{a}{\Rightarrow}$ for all $a \in L_I$. With this observation, we prove item (1) using Proposition 15(1), and prove item (2), using Proposition 15(2). □

Now we are in position to compare the definitions of the **ioco** relation. In [5], the definition of the **ioco** relation depends on the notion of *Straces*, as stated in its Definition 9. It proceeds as follows: Let $\mathcal{S} = (S, s_0, L_I, L_U, T) \in \mathcal{LTS}_T(L_I, L_U)$ and let $\mathcal{S}^\delta = \mathcal{T} \in \mathcal{LTS}^\delta_T(L_I, L_U^{+\delta})$ be its extension to include quiescence. Definition 9 says that, for all $s \in S$ and all $\sigma \in (L_I \cup L_U \cup \{\delta\})^\star$, we have $\sigma \in \mathit{Straces}(s)$ if $s \overset{\sigma}{\underset{\mathcal{T}}{\Rightarrow}}$. So, strings in $\mathit{Straces}(\mathcal{S})$ are just the observable traces of $\mathcal{T}$, i.e., $\mathit{Straces}(\mathcal{S}) = \mathit{otr}(\mathcal{T})$, the observable traces of $\mathcal{T}$. With the notion of *Straces*, Definition 12 in [5] specifies the $\mathbf{ioco}_T$ relation thus: Let $\mathcal{I} = (Q, q_0, L_I, L_U, R) \in \mathcal{IOTS}_T(L_I, L_U) \subseteq \mathcal{LTS}_T(L_I, L_U)$ and let $\mathcal{Q} = \mathcal{I}^\delta \in \mathcal{IOTS}^\delta_T(L_I, L_U^{+\delta})$ be its extension. Also let $\mathcal{S} = (S, s_0, L_I, L_U, T) \in \mathcal{LTS}_T(L_I, L_U)$ with $\mathcal{T} = \mathcal{S}^\delta \in \mathcal{LTS}^\delta_T(L_I, L_U^{+\delta})$ being its extension. Then, $\mathcal{I}$ $\mathbf{ioco}_T$ $\mathcal{S}$ if, and only if, for all $\sigma \in \mathit{Straces}(\mathcal{S})$, we have

$$\mathbf{out}^{\mathcal{Q}}_T(q_0 \ \mathbf{after}^{\mathcal{Q}} \ \sigma) \subseteq \mathbf{out}^{\mathcal{T}}_T(s_0 \ \mathbf{after}^{\mathcal{T}} \ \sigma).$$

Here $\mathbf{out}^{\mathcal{Q}}_T$ indicates that the set $\mathbf{out}_T$ is being obtained in $\mathcal{S}$ and, similarly, $\mathbf{after}^{\mathcal{Q}}$ indicates that the set **after** is being calculated in the $\mathcal{S}$, according to previous definitions.

Since $\mathit{Straces}(s_0) = \mathit{otr}(\mathcal{T})$, we then rewrite the definition of $\mathbf{ioco}_T$ as follows. Assume $\mathcal{I} = (Q, q_0, L_I, L_U, R) \in \mathcal{IOTS}_T(L_I, L_U) \subseteq \mathcal{LTS}_T(L_I, L_U)$ and $\mathcal{S} = (S, s_0, L_I, L_U, T) \in \mathcal{LTS}_T(L_I, L_U)$. We say that $\mathcal{I}$ $\mathbf{ioco}_T$ $\mathcal{S}$ if, and only if, for all $\sigma \in \mathit{otr}(\mathcal{S}^\delta)$, we have

$$\mathbf{out}_T(q_0 \ \mathbf{after} \ \sigma) \subseteq \mathbf{out}_T(s_0 \ \mathbf{after} \ \sigma),$$

where $\mathbf{out}_T$ and **after** are obtained over the corresponding extended models $\mathcal{I}^\delta$ and $\mathcal{S}^\delta$.

In this work, when $\mathcal{S} = (S, s_0, L_I, L_U, T)$ and $\mathcal{I} = (Q, q_0, L_I, L_U, R)$ are in $\mathcal{IO}_A(L_I, L_U)$, we say that $\mathcal{I}$ $\mathbf{ioco}_A$ $\mathcal{S}$ if, and only if, for all $\sigma \in \mathit{otr}(\mathcal{S})$, we have

$$\mathbf{out}_A(q_0 \ \mathbf{after} \ \sigma) \subseteq \mathbf{out}_A(s_0 \ \mathbf{after} \ \sigma).$$

See Definition 13.

Next proposition establishes the relationship between relations $\mathbf{ioco}_T$ and $\mathbf{ioco}_A$.

Proposition 21 *Let $\mathcal{I} \in \mathcal{IOTSR}_T(L_I, L_U^{-\delta})$ and let $\mathcal{S} \in \mathcal{LTSR}_T(L_I, L_U^{-\delta})$, with $\mathcal{Q} = \mathcal{I}^\delta$ and $\mathcal{T} = \mathcal{S}^\delta$ being their corresponding extended models. Then, $\mathcal{I}$ $\mathbf{ioco}_T$ $\mathcal{S}$ if and only if $\mathcal{Q}$ $\mathbf{ioco}_A$ $\mathcal{T}$.*

Proof Let $\mathcal{S} = (S, s_0, L_I, L_U^{-\delta}, T)$ and $\mathcal{I} = (Q, q_0, L_I, L_U^{-\delta}, R)$. From the definitions we get immediately $\mathcal{T} = (S, s_0, L_I, L_U^{+\delta}, T \cup T_\delta) \in \mathcal{LTSR}^\delta_T(L_I, L_U^{-\delta})$ and $\mathcal{Q} = (Q, q_0, L_I, L_U^{+\delta}, R \cup R_\delta) \in \mathcal{LTSR}^\delta_T(L_I, L_U^{-\delta})$, where $T_\delta = \{(s, \delta, s) | \delta^{\mathcal{S}}_T(s)\}$ and $R_\delta = \{(s, \delta, s) | \delta^{\mathcal{I}}_T(s)\}$.

We first show that $\mathcal{I}$ $\mathbf{ioco}_T$ $\mathcal{S}$ implies that $\mathcal{Q}$ $\mathbf{ioco}_A$ $\mathcal{T}$. From Proposition 17(2) we have $\mathcal{T}, \mathcal{Q} \in \mathcal{IO}^\delta_A(L_I, L_U^{+\delta})$, and so the relation $\mathbf{ioco}_A$ is also defined for $\mathcal{Q}$ and $\mathcal{T}$. Assume that we have $\mathcal{I}$ $\mathbf{ioco}_T$ $\mathcal{S}$, but $\mathcal{Q}$ $\mathbf{ioco}_A$ $\mathcal{T}$ does not hold. From the definition we have some $\sigma \in \mathit{otr}(\mathcal{T})$ and some $x \in L_U^{+\delta}$ such that

$$x \in \mathbf{out}^{\mathcal{Q}}_A(q_0 \ \mathbf{after}^{\mathcal{Q}} \ \sigma) \quad \text{and} \quad x \notin \mathbf{out}^{\mathcal{T}}_A(s_0 \ \mathbf{after}^{\mathcal{T}} \ \sigma).$$

Since $\mathcal{Q} \in \mathcal{IO}^\delta_A(L_I, L_U^{+\delta})$, using Proposition 19 we have $\mathbf{out}^{\mathcal{Q}}_A(q_0 \ \mathbf{after}^{\mathcal{Q}} \ \sigma) = \mathbf{out}^{\mathcal{Q}}_T(q_0 \ \mathbf{after}^{\mathcal{Q}} \ \sigma)$ and then $x \in \mathbf{out}^{\mathcal{Q}}_T(q_0 \ \mathbf{after}^{\mathcal{Q}} \ \sigma)$. Since $\mathcal{I}$ $\mathbf{ioco}_T$ $\mathcal{S}$ holds we should have $x \in \mathbf{out}^{\mathcal{T}}_T(s_0 \ \mathbf{after}^{\mathcal{T}} \ \sigma)$. Likewise, $\mathcal{T} \in \mathcal{IO}^\delta_A(L_I, L_U^{+\delta})$ and Proposition 19 now gives $\mathbf{out}^{\mathcal{T}}_A(q_0 \ \mathbf{after}^{\mathcal{T}} \ \sigma) = \mathbf{out}^{\mathcal{T}}_T(q_0 \ \mathbf{after}^{\mathcal{T}} \ \sigma)$. Hence, $x \in \mathbf{out}^{\mathcal{T}}_A(s_0 \ \mathbf{after}^{\mathcal{T}} \ \sigma)$ and we have reached a contradiction. Therefore, if $\mathcal{I}$ $\mathbf{ioco}_T$ $\mathcal{S}$ then $\mathcal{Q}$ $\mathbf{ioco}_A$ $\mathcal{T}$.

On the other direction, we need to show that if $\mathcal{Q}$ $\mathbf{ioco}_A$ $\mathcal{T}$ then $\mathcal{I}$ $\mathbf{ioco}_T$ $\mathcal{S}$. Now, assume that we have $\mathcal{Q}$ $\mathbf{ioco}_A$ $\mathcal{T}$, but $\mathcal{I}$ $\mathbf{ioco}_T$ $\mathcal{S}$ does not hold. The reasoning is entirely analogous and again we would reach a contradiction. □

8.6 Determinism

We also show that the notion of determinism defined in [5] coincides with our definition of determinism. In [5], Definition 5(9) characterizes determinism as follows: $\mathcal{S} = (S, L, T, s_0) \in \mathcal{LTS}_T(L)$ is deterministic if, for all $\sigma \in L^\star$, s_0 **after** σ has at most one element. We indicate this by writing $\mathit{determ}_T(\mathcal{S})$.

In our work, Definition 5 says that $\mathcal{S} = (S, s_0, L, T) \in \mathcal{LTS}_A(L)$ is deterministic if $s_0 \overset{\sigma}{\Rightarrow} s_1$ and $s_0 \overset{\sigma}{\Rightarrow} s_2$ imply $s_1 = s_2$, for all s_1, $s_2 \in S$ and all $\sigma \in L^\star$. If that is the case, we write $\mathit{determ}_A(\mathcal{S})$.

The next proposition shows that these notions coincide for every element in $\mathcal{LTS}_A(L)$.

Proposition 22 *Let $\mathcal{S} = (S, s_0, L, T) \in \mathcal{LTS}_A(L)$. Then $\mathit{determ}_T(\mathcal{S})$ if and only if $\mathit{determ}_A(\mathcal{S})$.*

Proof Note that from Proposition 15(1) we also have $\mathcal{S} \in \mathcal{LTS}_T(L)$.

Assume that $\mathit{determ}_T(\mathcal{S})$ holds. We have that s_0 $\mathbf{after}_T \ \sigma$ has at most one element if and only if $s_0 \overset{\sigma}{\Rightarrow}_T s_1$ and $s_0 \overset{\sigma}{\Rightarrow}_T s_2$ imply $s_1 = s_2$, for all s_1, $s_2 \in S$, and all $\sigma \in L^\star$. From Proposition 12 we get that $s_0 \overset{\sigma}{\Rightarrow}_A s_1$ and $s_0 \overset{\sigma}{\Rightarrow}_A s_2$ imply $s_1 = s_2$, so that $\mathit{determ}_A(\mathcal{S})$ also holds.

For the other direction just reverse the argument. □

8.7 Test cases and test purposes

When defining the class of all test cases with inputs L_I and outputs L_U, [5] starts, at Definition 10, with a model $\mathcal{T} = (S, s_0, L_U^{-\delta}, L_I^{+\theta,-\delta}, T)$ in the class $\mathcal{IOTS}_T(L_U^{-\delta}, L_I^{+\theta,-\delta})$, that is, $\mathcal{T}$ is already input-enabled with respect to $L_U^{-\delta}$. Further restrictions apply, namely:

1. $\mathcal{T}$ is finite state and deterministic as defined in [5]. We recall that according to Definition 2 all our models are finite state.
2. S contains two distinct states, **pass** and **fail**, and $\mathbf{out}(\mathbf{pass}) = \mathbf{out}(\mathbf{fail}) = L_U^{+\theta,-\delta}$.
3. $\mathcal{T}$ has no cycles, except at states **pass** and **fail**, that is $s \stackrel{\sigma}{\Rightarrow} s$ implies $s = \mathbf{pass}$ or $s = \mathbf{fail}$ for any $\sigma \neq \varepsilon$ and $\sigma \in (L_U^{-\delta} \cup L_I^{+\theta,-\delta})^\star$.
4. For all state $s \in S$, we must have $\mathbf{init}_T(s) = \{x\} \cup L_U^{-\delta}$, for some $x \in L_I^{+\theta,-\delta}$.

For any model $\mathcal{S} = (S, s_0, L_I, L_U, T)$, Definition 5 in [5] says that for all $s \in S$ we have $\mathbf{init}_T(s) = \{x \in L_I \cup L_U \cup \{\tau\} \mid s \stackrel{x}{\rightarrow}\}$. Hence, for any model $\mathcal{T} = (S, s_0, L_U^{-\delta}, L_I^{+\theta,-\delta}, T)$ condition 4 reduces to saying that for all $s \in S$ we must have

$$\{x\} \cup L_U^{-\delta} = \{y \in L_I^{+\theta,-\delta} \cup L_U^{-\delta} \cup \{\tau\} \mid s \stackrel{y}{\rightarrow}\},$$

for some $x \in L_I^{+\theta,-\delta}$. We conclude that $s \stackrel{x}{\Rightarrow}$ for all $x \in L_U^{-\delta}$ and all $s \in S$. Thus, when condition 4 holds we know that $\mathcal{T}$ is already input-enabled.

In [5], Definition 16, the θ symbol in test cases synchronizes with the δ symbol that signals quiescence in IUTs. Here we have used the same δ symbol in test purposes to synchronize with the δ symbol that flags quiescence in IUTs, as made explicit in Definition 17. So, specifications and IUTs are models from $\mathcal{IO}_A(L_I^{-\delta}, L_U^{+\delta})$. When constructing test purposes from given specifications, the input and output alphabets are interchanged. Accordingly, a test purpose over the input alphabet L_I and output alphabet L_U is defined as a model $\mathcal{T} = \langle S, s_0, L_U^{+\delta}, L_I^{-\delta}, T\rangle$ in $\mathcal{IO}_A(L_U^{+\delta}, L_I^{-\delta})$.

In our Definition 13, given any model $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle \in \mathcal{IO}_A(L_I, L_U)$ we say that $\mathbf{out}_A(s) = \{x \in L_U \mid s \stackrel{x}{\Rightarrow}\}$ for all $s \in S$, and in Definition 21 we say that $\mathcal{S}$ is output-deterministic when $|\mathbf{out}_A(s)| = 1$ for all $s \in S$. Also such a model $\mathcal{S}$ is input-enabled when $\mathbf{inp}_A(s) = L_I$. Hence, a model $\mathcal{T} = \langle S, s_0, L_U^{+\delta}, L_I^{-\delta}, T\rangle$ is output-deterministic when $|\{x \in L_I^{-\delta} \mid s \stackrel{x}{\Rightarrow}\}| = 1$ and it is input-enabled when $\mathbf{inp}_A(s) = L_U^{+\delta}$, for all $s \in S$. Moreover, since in our models we substitute δ for θ, conditions (2), (3) and (4) should read as follows:

(2′) S contains two distinct states, **pass** and **fail**, and $\mathbf{out}(\mathbf{pass}) = \mathbf{out}(\mathbf{fail}) = L_U^{+\delta}$.

(3′) $\mathcal{T}$ has no cycles, except at states **pass** and **fail**, that is $s \stackrel{\sigma}{\Rightarrow} s$ implies $s = \mathbf{pass}$ or $s = \mathbf{fail}$ for any $\sigma \neq \varepsilon$ and $\sigma \in (L_U^{+\delta} \cup L_I^{-\delta})^\star$.

(4′) For all state $s \in S$, we must have $\mathbf{init}_T(s) = \{x\} \cup L_U^{+\delta}$, for some $x \in L_I^{-\delta}$.

Equivalently, condition (4′) can be written as

$$\{x\} \cup L_U^{+\delta} = \{y \in L_I^{-\delta} \cup L_U^{+\delta} \cup \{\tau\} \mid s \stackrel{y}{\rightarrow}\},$$

for some $x \in L_I^{-\delta}$.

The next result says that certain models $\mathcal{T} \in \mathcal{IO}_A(L_U^{+\delta}, L_I^{-\delta})$ satisfy conditions (1) and (4) above.

Proposition 23 *Let $\mathcal{T} = \langle S, s_0, L_U^{+\delta}, L_I^{-\delta}, T\rangle \in \mathcal{IO}_A(L_U^{+\delta}, L_I^{-\delta})$ be deterministic, input-enabled and output-deterministic. Then $\mathcal{T}$ satisfies conditions (1) and (4′) given above.*

Proof Since $\mathcal{T}$ is deterministic, using Proposition 22 we conclude that $\mathcal{T}$ is also deterministic in the sense defined in [5]. Thus, condition (1) holds.

By the preceding discussion, if $\mathcal{T}$ satisfies condition (4) then is is already input-enabled. Also from the text above, it remains to show that for all $s \in S$ we have

$$\{x\} \cup L_U^{+\delta} = \{y \in L_I^{-\delta} \cup L_U^{+\delta} \cup \{\tau\} \mid s \stackrel{y}{\rightarrow}\},$$

for some $x \in L_I^{-\delta}$. So, fix some $s \in S$. Since $|\mathbf{out}_A(s)| = 1$, we may choose $x \in L_I^{-\delta}$ such that $\mathbf{out}_A(s) = \{x\}$, and now we have to show that

$$\mathbf{out}_A(s) \cup L_U^{+\delta} = \{y \in L_I^{-\delta} \cup L_U^{+\delta} \cup \{\tau\} \mid s \stackrel{y}{\rightarrow}\}.$$

So, let $y \in L_I^{-\delta} \cup L_U^{+\delta} \cup \{\tau\}$ with $s \xrightarrow{y}$. Since $\mathcal{T}$ is deterministic, Proposition 1 says that $\mathcal{T}$ has no τ-labeled transitions. Hence, $y \in L_I^{-\delta} \cup L_U^{+\delta}$. If $y \in L_U^{+\delta}$ then $y \in \mathbf{out}_A(s) \cup L_U^{+\delta}$. Now assume that $y \in L_I^{-\delta}$ with $s \xrightarrow{y}$, so that $y \in \mathbf{out}_A(s)$. Since $|\mathbf{out}_A(s)| = 1$ we have $\mathbf{out}_A(s) = \{y\}$, and again $y \in \mathbf{out}_A(s) \cup L_U^{+\delta}$. We have shown that $\{y \in L_I^{-\delta} \cup L_U^{+\delta} \cup \{\tau\} \mid s \xrightarrow{y}\} \subseteq \mathbf{out}_A(s) \cup L_U^{+\delta}$.

Now let $y \in \mathbf{out}_A(s) \cup L_U^{+\delta}$. If $y \in \mathbf{out}_A(s)$ then $y \in L_I^{-\delta}$ and $s \stackrel{y}{\Rightarrow}$. Since $\mathcal{T}$ has no τ-labeled transitions we get $s \xrightarrow{y}$. Hence $y \in \{y \in L_I^{-\delta} \cup L_U^{+\delta} \cup \{\tau\} \mid s \xrightarrow{y}\}$. Now assume $y \in L_U^{+\delta}$. Since $\mathbf{inp}_A(s) = L_U^{+\delta}$ we get $y \in \mathbf{inp}_A(s)$, so that $s \stackrel{y}{\Rightarrow}$. Because $\mathcal{T}$ has no τ-labeled transitions we have $s \xrightarrow{y}$. Hence, $y \in L_U^{+\delta}$ and $s \xrightarrow{y}$ give $y \in \{y \in L_I^{-\delta} \cup L_U^{+\delta} \cup \{\tau\} \mid s \xrightarrow{y}\}$. We now have $\mathbf{out}_A(s) \cup L_U^{+\delta} \subseteq \{y \in L_I^{-\delta} \cup L_U^{+\delta} \cup \{\tau\} \mid s \xrightarrow{y}\}$ and the proof is complete. □

In Theorem 15 we showed that test purposes satisfying a number of restrictions can be constructed for any given specification. We can now verify that those test purpose models are also test cases, in the sense used in [5].

Proposition 24 *Let $\mathcal{S}$ be a specification in $\mathcal{IO}_A(L_I^{-\delta}, L_U^{+\delta})$, and let $m \geq 1$. Then the set TP of test purposes constructed in Theorem 15 is* **ioco***-complete for $\mathcal{S}$ with respect to any implementation with at most m states. Moreover, any test purpose in TP satisfies conditions (1), (2′), (3′) and (4′) listed above.*

Proof By Theorem 15 we know that TP is m-**ioco**-complete for $\mathcal{S}$.

Let $\mathcal{T} = \langle S, s_0, L_U^{+\delta}, L_I^{-\delta}, T\rangle \in \mathcal{IO}_A(L_U^{+\delta}, L_I^{-\delta})$ be a test purpose constructed in TP. By Theorem 15 we know that $\mathcal{T}$ is already deterministic, input-enabled and output-deterministic. So, by Proposition 23 conditions (1) and (4′) are satisfied.

By Theorem 15, $\mathcal{T}$ has two distinct **pass** and **fail** states, and it is acyclic except for self-loops at these states. The proof of Theorem 15 starts with test purposes constructed at Proposition 10. A simple examination of the proof of Proposition 10 shows that we explicitly add self-loops $(\mathbf{fail}, \ell, \mathbf{fail})$ and $(\mathbf{pass}, \ell, \mathbf{pass})$ to $\mathcal{T}$, for all $\ell \in L_U^{+\delta}$. Hence, conditions (2′) and (3′) are also satisfied. □