Generating Test Suites for Input/Output Labeled Transition Systems

Adilson Luiz Bonifacio *Arnaldo Vieira Moura*

Technical Report - IC-17-16 - Relatório Técnico
November - 2017 - Novembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Generating Test Suites for Input/Output Labeled Transition Systems

Adilson Luiz Bonifacio[*]   Arnaldo Vieira Moura[†]

Resumo

Model based testing is a well-established approach to test reactive systems. One of the challenges stemming from model based testing is the generation of test suites, specially when completeness is a required property these test suites. In order to check whether an implementation under test is in compliance with its respective specification model one resorts to some form of a conformance relation that guarantees some expected behavior of the implementations, given the behavior of the specification. The **ioco** conformance relation is an example of such a relation, specially suited for reactive and asynchronous models. In this work we study a more general conformance relation for such models. We also describe a method to generate finite and complete test suites which are complete in general, and we also discuss the complexity of the generation mechanism, as well as the complexity of testing implementations under this more general conformance relation. We show that **ioco** conformance is a special case of this new conformance relation, for two classes of fault models, and we also investigate the complexity of generating complete test suites for these fault models, and also the complexity of running verification experiments using the test suites so generated.

1 Introduction

Software testing has always been an important part in many systems development processes, with the goal of improving the quality of the final product. 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) [2, 4, 7, 11, 13, 15], can capture some aspects of reactive systems. However, in this formalism, input actions from the environment and output actions produced by the model are strongly related and must occur synchronously. In many situations, this limits the designer ability to model more complex reactive behaviors. This work studies aspects of a class of more powerful formalisms that can be used to model reactive behaviors, where the exchange of input and output stimuli can occur asynchronously, namely, the class of Labeled Transition Systems (LTSs) [23] and its close variant class of Input/Output Labeled Transition Systems (IOLTSs) [26].

In this context, model based testing [8, 21, 25] has been widely used as a formal framework to verify whether an implementation under test (IUT) is in conformance to desired behaviors of a

[*]Computing Department, University of Londrina, Londrina, Brazil, `email: bonifacio@uel.br`.

[†]Computing Institute, University of Campinas, Campinas, Brazil, `email: arnaldo@ic.unicamp.br`.

specification, according to a given fault model and a given conformance relation [1]. Conformance testing [5, 17, 23, 26] is a well-established approach for model based testing that can be used for the validation of reactive systems. A well-known conformance relation, called **ioco** conformance, has already been proposed for testing IOLTS models [24, 27].

Some works have studied the problem of constructing test suites to efficiently search for faults on IOLTS implementations. Ideally, one strives to obtain small and complete test suites for wide classes of IOLTS models, and some progress has been recently achieved in this direction [22]. But, in order to obtain such complete test suites, a number of restrictions is imposed upon the specification and the implementation models, which limits somewhat the applicability of such approaches.

Here, we propose a different and more general notion of conformance relation for testing IOLTS models, one that is based finite state automata. Under this more general notion of conformance, it is possible to accommodate much wider classes of IOLTS models, thus removing some of the structural restrictions imposed on the models by other approaches. The main contributions can be summarized as follows:

— The more general notion of conformance allows for the specification of arbitrary desired and undesired behaviors from a IUT. When the language specifying these behaviors are both regular, we show that there is a (unique) regular language that can be used as a complete test suite to detect the presence of all desired behaviors and the absence of any undesired behaviors.

— In a scenario of a "white box" testing architecture, we prove the correctness of a polynomial time algorithm that can be used for checking conformance in this more general setting. If the specification model is fixed, then the algorithm runs in linear time in the number of states in the implementation.

— We show that the classical **ioco** conformance relation [27], and when treating implementations as "black boxes", is a special case of this new conformance relation. We show how to generate complete test suites that can be used to verify **ioco**-conformance under the same set of fault models [27], and for implementations of any number of states, independently of the number of states in the specification. Further, under the "black box" assumption, we prove an asymptotic exponential lower bound $\Omega(()\Phi^m)$ on the size of any **ioco**-complete test suites, where m is the number of states of the largest implementation to be tested, and $\Phi = (1 + \sqrt{5})/2$. We also show that our approach, in general, attains such a lower bound.

— We also consider another testing architecture, discussed in a more recent work [22]. Again, even under the restrictions imposed on the specifications and implementations by that architecture, we prove that a variation of the previous algorithm suffices to generate complete test suites, in general. Moreover, we show that complete test suites for this test architecture must be, in general, of asymptotic exponential size, even when considering a family of simple specification models.

The paper is organized as follows. Section 2 establishes notations, definitions and lists some preliminary results. Section 3 defines the new notion of conformance relation and relates it with the classical **ioco** conformance relation. A method for generating complete test suites for testing adherence to the new conformance relation by IOLTS models, and its complexity, is described in Section 4. Section 5 visits the classical **ioco**-conformance relation and establishes the exponential lower bounds on the size of complete test suites. Section 6 looks at another class of IOLTS models and show hw to obtain complete test suites for this class, and also discusses performance issues when working with models in this class. We also briefly comment on other related results in Section 7. Section 8 offers some final remarks.

2 Notation and preliminary results

This section defines Labeled Transition Systems (LTSs), and the variation of Input/Output Labeled Transition Systems (IOLTSs). For the sake of completeness, we also include standard definitions and properties of regular languages and finite state automata (FSAs) that will be useful later. Some preliminary results stating the relationship between LTSs and the associated FSAs are then discussed. It starts with the formal models and some notation.

2.1 Basic Notation

Let X and Y be sets. We indicate by $\mathcal{P}(X) = \{Z \,|\, Z \subseteq X\}$ the power set of X, and $X - Y = \{z \,|\, z \in X \text{ and } z \notin Y\}$ indicates set difference. We will let $X_Y = X \cup Y$. When $Y = \{y\}$ is a singleton we may also write X_y for $X_{\{y\}}$. If X is a finite set, the size of X will be indicated by $|X|$.

An alphabet is any non-empty set of symbols. Let A be an alphabet. A word over A is any finite sequence $\sigma = x_1 \dots x_n$ of symbols in A, that is, $n \geq 0$ and $x_i \in A$, for all $i = 1, 2, \dots, n$. Note that when $n = 0$ then σ is the empty sequence, also indicated by ε. When we write $x_1 x_2 \dots x_n \in A^\star$, it is implicitly assumed that $n \geq 0$ and that $x_i \in A$, $1 \leq i \leq n$, unless explicitly noted otherwise. The length of a word α over A 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 of σ and ρ, indicated by $\sigma\rho$, is the word $\sigma_1 \dots \sigma_n \rho_1 \dots \rho_m$. Clearly, $|\sigma\rho| = |\sigma| + |\rho|$. The set of all finite sequences, or words, over A is denoted by $A^\star$. A language G over A is any set $G \subseteq A^\star$ of words over A. Let G_1, $G_2 \subseteq A^\star$ be languages over A. Their product, indicated by $G_1 G_2$, is the language $\{\sigma\rho \,|\, \sigma \in G_1, \rho \in G_2\}$. If $G \subseteq A^\star$ is a language over A, then its complement is the language $\overline{G} = A^\star - G$.

We will also need the notion of a morphism between alphabets.

Definition 2.1. *Let A, B be alphabets. A* homomorphism*, or just a* morphism*, from A to B is any function $h : A \to B^\star$.*

A morphism $h : A \to B^\star$ can be extended in a natural way to a function $\widehat{h} : A^\star \to B^\star$, thus

$$\widehat{h}(\sigma) = \begin{cases} \varepsilon & \text{if } \sigma = \varepsilon \\ h(a)\widehat{h}(\rho) & \text{if } \sigma = a\rho \text{ with } a \in A. \end{cases}$$

We can further lift $\widehat{h}$ to a function $\widetilde{h} : \mathcal{P}(A^\star) \to \mathcal{P}(B^\star)$ in a natural way, by letting $\widetilde{h}(G) = \cup_{\sigma \in G} \widehat{h}(\sigma)$, for all $G \subseteq A^\star$. In order to avoid cluttering the notation, we often write h in place of $\widehat{h}$, or of $\widetilde{h}$, when no confusion can arise.

When a is any symbol in A, we define the simple morphism $h_a : A \to A - \{a\}$ by letting $h_a(a) = \varepsilon$, and $h_a(x) = x$ when $x \neq a$. So, we see that $h_a(\sigma)$ just erases all occurrences of the symbol a in the word σ.

The following result will prove very useful later on.

Lemma 2.2. *Let A be an alphabet and let A_1, $A_2 \subseteq A^\star$ be languages over A. Let $B \subseteq A^\star$ be a third language over A. Then, there is a finite language $C \subseteq B$ such that the following holds*

$$A_2 \cap B \subseteq A_1 \quad \textit{if and only if} \quad A_2 \cap C \subseteq A_1.$$

Demonstração. If $A_2 \cap B \subseteq A_1$, then let $C = \emptyset$ and we immediately get $C \subseteq B$ and $A_2 \cap C = \emptyset \subseteq A_1$. If $A_2 \cap B \not\subseteq A_1$, let $\sigma \in A_2 \cap B$ and $\sigma \notin A_1$. Define $C = \{\sigma\}$. Clearly, $C \subseteq B$ and also $A_2 \cap C = \{\sigma\} \not\subseteq A_1$. □

Having established some basic notation, we can now turn to transition systems.

2.2 Labeled Transition Systems

The syntactic description of a labeled transition system is introduced next.

Definition 2.3. *A* Labeled Transition System *(LTS) is a tuple* $\mathcal{S} = \langle S, s_0, L, T\rangle$*, where:*

— S *is a countable set of* states *or* locations*;*

— s_0 *is the* initial state*, or* initial location*;*

— L *is a countable set of* labels*, or* actions*. We also have a symbol* $\tau \notin L$*, the* internal action symbol*;*

— $T \subseteq S \times L_\tau \times S$ *is a countable set of* transitions*.*

The set of all LTSs having L as the set of labels is indicated by $\mathcal{L}(L)$.

Example 2.4. Figure 1 represents a LTS $\mathcal{S} = \langle S, s_0, L, T\rangle$ where the set of states is $S = \{s_0, s_1, s_2, s_3, s_4\}$ and the set of labels is $\{b, c, t\}$. The set of transitions is given by the arrows,

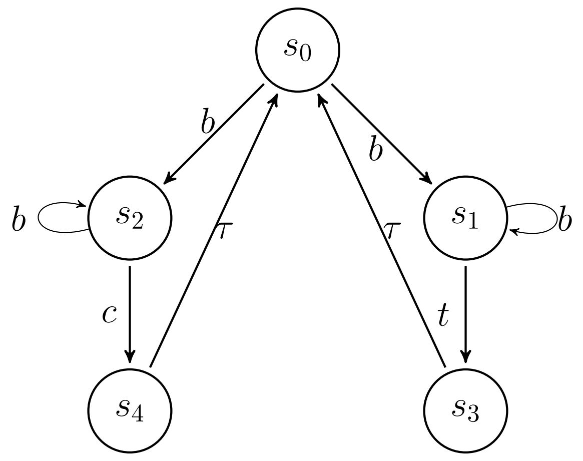

Figura 1: A LTS with 5 states and 8 transitions.

so that (s_0, b, s_1) and (s_3, τ, s_0) are transitions. Then, we have

$$T = \{(s_0,b,s_1),(s_0,b,s_2),(s_1,b,s_1),(s_1,t,s_3),(s_2,b,s_2),(s_2,c,s_4),(s_3,\tau,s_0),(s_4,\tau,s_0)\},$$

as the set of all transitions in $\mathcal{S}$. □

We gave only the syntax, *i.e.*, a finite description, of a LTS. The semantics of a LTS is given by its traces, or behaviors. But first we need the notion of paths in a LTS.

Definition 2.5. *Let* $\mathcal{S} = \langle S, s_0, L, T\rangle$ *be a LTS and let* $p, q \in S$*.*

1. *Let $\sigma = l_1, \ldots, l_n$ be a word in $L_\tau^\star$. We say that σ is a* path *from p to q in $\mathcal{S}$ if there are states $r_i \in S$, $0 \leq i \leq n$, and labels $l_i \in L_\tau$, $1 \leq i \leq n$, such that $(r_{i-1}, l_i, r_i) \in T$, $1 \leq i \leq n$, with $r_0 = p$ and $r_n = q$.*

2. *Let σ be a word in $L^\star$. We say that σ is an* observable path *from p to q in $\mathcal{S}$ if there is a path μ from p to q in $\mathcal{S}$ such that $\sigma = h_\tau(\mu)$.*

In both cases we also say that the path starts at p and ends at q.

Example 2.6. Consider the LTS depicted at Figure 1. The following sequences are paths starting at s_2: ε, b, $bc\tau$, $c\tau bbbt\tau b$. We then get, among others, the following observable paths starting at s_2: ε, b, bc, $cbbbtb$. There are observable paths σ from s_2 to s_1 of length $|\sigma| > n$, for all $n \geq 0$. □

Clearly, internal labeled moves can occur in a path. An observable path is just a path from which moves labeled by internal actions were removed. If σ is a path from p to q, this can also be indicated by writing $p \xrightarrow{\sigma} q$. We may also write $p \xrightarrow{\sigma}$ to indicate that there is some $q \in S$ such that $p \xrightarrow{\sigma} q$; likewise, $p \rightarrow q$ means that there is some $\sigma \in L_\tau^*$ such that $p \xrightarrow{\sigma} q$. Also $p \rightarrow$ means $p \xrightarrow{\sigma} q$ for some $q \in S$ and some $\sigma \in L_\tau^*$. When σ is an observable path from p to q we may write $p \overset{\sigma}{\Rightarrow} q$, with similar shorthand notation also carrying over to the $\Rightarrow$ relation. When we want to emphasize that the underlying LTS is $\mathcal{S}$, we write $p \underset{\mathcal{S}}{\xrightarrow{\sigma}} q$, or $p \underset{\mathcal{S}}{\overset{\sigma}{\Rightarrow}} q$.

Remark 2.7. *Note that in any LTS $\mathcal{S} = \langle S, s_0, L, T\rangle$ we have $s \Rightarrow p$ if and only if we also have $s \rightarrow p$ in $\mathcal{S}$, for all s, $p \in S$.*

Paths starting at a given state p are also called the traces of p, or the traces starting at p. The semantics of a LTS is related to traces starting at the initial state.

Definition 2.8. *Let $\mathcal{S} = \langle S, s_0, L, T\rangle$ be a LTS and let $p \in S$.*

1. *The set of* traces *of p is $tr(p) = \{\sigma \,|\, p \xrightarrow{\sigma}\}$. The set of* observable traces *of p is $otr(p) = \{\sigma \,|\, p \overset{\sigma}{\Rightarrow}\}$.*

2. *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 will also indicate the semantics, and respectively the observable semantics, of $\mathcal{S}$ by $tr(\mathcal{S})$ and $otr(\mathcal{S})$. The following assertion is immediate from the definitions.

Remark 2.9. *If $\mathcal{S}$ has no τ-labeled transitions, then $otr(\mathcal{S}) = tr(\mathcal{S})$.*

Example 2.10. Consider the LTS depicted at Figure 1. The semantics of $\mathcal{S}$ include such sequences: ε, bbc, $bt\tau bc\tau bbb$, among others. The observable semantics of $\mathcal{S}$ include: ε, $bcbt$, $bbbtbcbtb$, among others. □

A LTS $\mathcal{S}$ where the set of states S is infinite is better represented (or modeled) using a set of internal variables that can take values in infinite domains. When the set of labels L is infinite, it is better to use parameters, that is variables whose values can be exchanged with the environment, and letting such parameters also take values in infinite domains. We can also restrict the syntactic description LTS model somewhat, without loosing any descriptive capability. First, since τ indicates an internal move in the LTS, we will also assume that we do not have useless internal moves that do not change states. Further, we can do away with states that are not reachable from the initial state, since these states will play no role when considering any system behaviors that start at the initial state.

We formalize these observations in te following remark.

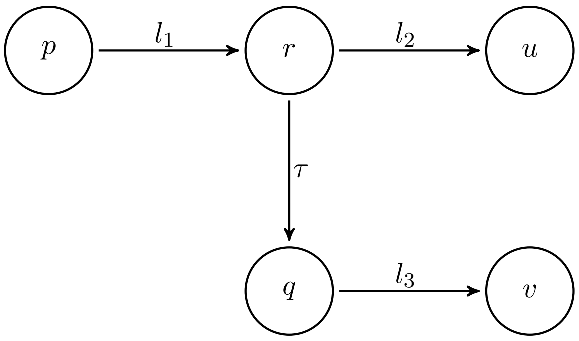

Figura 2: Part of a LTS $\mathcal{S}_1$ with internal moves.

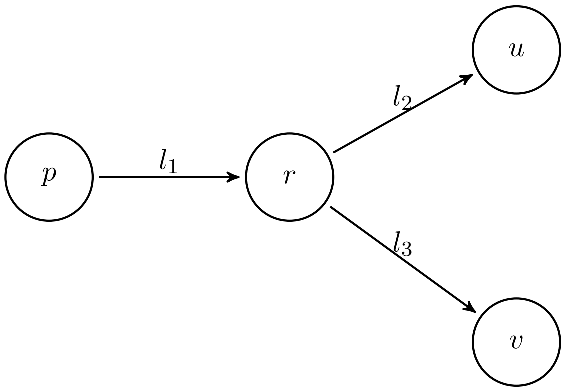

Figura 3: An LTS $\mathcal{S}_1$ with no internal moves.

Remark 2.11. *In Definition 2.3 we will always assume that S and L are finite sets. Further, for any $s \in S$ we postulate that $(s, \tau, s) \notin T$, and that $s_0 \to s$.*

The intended interpretation for the "internal action symbol" τ is that the LTS can autonomously move along any transition labeled by τ, without the need to consume any observable labels. In other words, from an "external" perspective, label τ induces a kind of "implicit nondeterminism". Consider Figure 2. After the move $p \xrightarrow{l_1} r$ in $\mathcal{S}_1$ the machine can move to u on label l_2, or it can also move to v on label l_3 because it autonomously can also take the internal move from r to q. From an "external" perspective this situation is equivalent to the transitions depicted in Figure 3.

We understand that internal actions can facilitate the specification of formal models that correspond to some intended behavior of the physical devices, *e.g.*, hardware or software, being modeled. For example, consider a real specification saying that "after delivering money, the ATM returns to the initial state". If this behavior does not require exchanging messages with a user, then it can be easily specified by an internal action transition back to the initial state . In some situations, however, we may want to avoid such ambiguous behaviors. In general, we might want that no observable behavior leading to two or more distinct states. This motivates a special variant of LTSs.

Definition 2.12. *Let $\mathcal{S} = \langle S, s_0, L, T\rangle$ be a LTS. We say that $\mathcal{S}$ 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 $\sigma \in L^\star$.*

As a consequence, we have the following result.

Proposition 2.13. *Let $\mathcal{S} = \langle S, s_0, L, T\rangle$ be a deterministic LTS. Then $\mathcal{S}$ has no τ-labeled transitions.*

Demonstração. For the sake of contradiction, assume that (p, τ, q) is a transition of $\mathcal{S}$. So, $p \xrightarrow{\tau} q$. From Remark 2.11 we get $p \neq q$ and $\sigma \in L^\star$ such that $s_0 \overset{\sigma}{\Rightarrow} p$. From Definition 2.8, $s_0 \xrightarrow{\mu} p$, where $h_\tau(\mu) = \sigma$, and so $s_0 \xrightarrow{\mu} p \xrightarrow{\tau} q$. Then, $s_0 \overset{\sigma}{\Rightarrow} q$ because $h_\tau(\mu\tau) = h_\tau(\mu) = \sigma$. This contradicts the fact that $\mathcal{S}$ is deterministic, according to Definition 2.12. □

2.3 Finite State Automata

As will be apparent, any LTS naturally induces a finite state automaton, which is a well studied formal model, with a number of convenient simple properties, and for which there are simple decision algorithms. In particular, transitions labeled by internal actions in the LTS will correspond to nondeterminism induced by ε-moves in the finite automaton.

First we give the formal syntax of a finite state automaton.

Definition 2.14. *A* Finite State Automaton *(FSA, for short) is a tuple $\mathcal{A} = \langle S, s_0, A, \rho, F\rangle$, where:*

— *S is a finite set of* states*;*

— *s_0 is the* initial state*;*

— *A is a finite nonempty* alphabet*;*

— *$\rho \subseteq S \times (A \cup \{\varepsilon\}) \times S$ is the* transition relation*;*

— *$F \subseteq S$ is the set of* final states*.*

*A transition $(p, \varepsilon, q) \in \rho$ is called an ε-*move *of $\mathcal{A}$. We also say that $\mathcal{A}$ is a FSA over the alphabet A.*

The semantics of a FSA is given by the language it accepts. But first, we establish a notation to describe moves in a FSA.

Definition 2.15. *Let $\mathcal{A} = \langle S, s_0, A, \rho, F\rangle$ be a FSA. Then we say that $\mathcal{A}$ induces the move relation $\vdash_{\mathcal{A}}$ defined in the set $S \times A^\star$ of* configurations *of $\mathcal{A}$, where $(p, \sigma) \vdash_{\mathcal{A}} (q, \mu)$ if either*

- *$\sigma = \mu$ and $(p, \varepsilon, q) \in \rho$, or*
- *$\sigma = x\mu$, $x \in A$ and $(p, x, q) \in \rho$.*

Since $\vdash_{\mathcal{A}}$ is defined in the set $S \times A^\star$ we write $\vdash^n_{\mathcal{A}}$ for its n-th power ($n \geq 0$), and we indicate by $\vdash^\star_{\mathcal{A}}$ its reflexive and transitive closure, that is, $\vdash^\star_{\mathcal{A}}$ if and only if $\vdash^n_{\mathcal{A}}$ for some $n \geq 0$. In order to simplify the notation, we may also write $p \overset{\sigma}{\mapsto} q$ when $(p, \sigma) \vdash^\star_{\mathcal{A}} (q, \varepsilon)$. We may use the same short versions of this notation as those already mentioned for LTSs and write, *e.g.*, $p \mapsto q$, or $p \overset{\sigma}{\mapsto}$, or even $p \mapsto$, when the missing terms exist but are unimportant in the current discourse.

Definition 2.16. *Let $\mathcal{A} = \langle S, s_0, A, \rho, F\rangle$ be a FSA. The* language *accepted by $\mathcal{A}$ is the set*

$$L(\mathcal{A}) = \{\sigma \in A^\star \mid (s_0, \sigma) \vdash^\star_{\mathcal{A}} (p, \varepsilon) \text{ and } p \in F\}.$$

A language is said to be regular when it is accepted by some FSA.

Definition 2.17. *Let A be an alphabet and let $G \subseteq A^\star$ be a language over A. Then G is* regular *if there is a FSA $\mathcal{A} = \langle S, s_0, A, \rho, F\rangle$ such that $L(\mathcal{A}) = G$.*

We have the following easy observation, which will be useful later on.

Remark 2.18. *Let A be an alphabet. Then, any finite language over A is a regular language. Also, $A^\star$ is a regular language.*

Next, we state some closure properties of the family of regular languages.

Proposition 2.19. *Let A and B be alphabets, and let $h : A \to B^\star$ be a morphism. Also, let G, $H \subseteq A^\star$, and $L \subseteq B^\star$ be regular languages. We have:*

1. *$G \cdot H$, $G \cap H$, $G \cup H$, and $\overline{G}$ are regular languages.*
2. *$h(G) \subseteq B^\star$ and $h^{-1}(L) \subseteq A^\star$ are regular languages.*

Moreover, in any of these cases, given FSAs for the original languages we can effectively construct a FSA that accepts the corresponding language indicated by the operation.

Demonstração. See, *e.g.*, [14, 12]. □

A FSA with no ε-moves and in which paths do not lead to distinct states is called a deterministic FSA.

Definition 2.20. *Let $\mathcal{A} = \langle S, s_0, A, \rho, F\rangle$ be a FSA. We say that $\mathcal{A}$ is a* deterministic finite state automaton *(DFSA) if ρ is a partial function from $S \times A$ into S. We say that $\mathcal{A}$ is a* complete FSA *if it has no ε-moves and ρ is a total function from $S \times A$ into S.*

Hence, $\mathcal{A} = \langle S, s_0, A, \rho, F\rangle$ is deterministic if it has no ε-moves and $(s, x, p_1) \in \rho$ and $(s, x, p_2) \in \rho$ gives $p_1 = p_2$. And $\mathcal{A}$ is complete when it is deterministic and for all $s \in S$ and all $x \in A$ there is $p \in S$ with $(s, x, p) \in \rho$.

The following proposition says that both ε-moves and nondeterministic moves over symbols in the alphabet can be removed from a FSA, resulting in a deterministic FSA. The resulting FSA will be obtained using the standard subset construction [19, 14], but here applied to simultaneously eliminate both ε-moves and nondeterministic moves over symbols. We will observe closely the final states in the resulting FSA.

Proposition 2.21. *Let $\mathcal{A} = \langle S_\mathcal{A}, a_0, A, \rho_\mathcal{A}, F_\mathcal{A}\rangle$ be a FSA. Then, there is a deterministic FSA $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$. Moreover, if $S_\mathcal{A} = F_\mathcal{A}$, then all states in $\mathcal{B}$ are also final states. Further, $\mathcal{B}$ can be algorithmically constructed from $\mathcal{A}$.*

Demonstração. Consider the sets $E(s) = \{p \,|\, s \overset{\varepsilon}{\mapsto} p\} \subseteq S$, defined for all $s \in S$, that is $p \in E(s)$ if and only if $(s, \varepsilon) \,|\!\frac{\star}{\mathcal{A}}\, (p, \varepsilon)$. Using a standard backward breadth-first graph traversal algorithm we can easily compute these sets. It is clear that if $p \in E(s)$ then we have $E(p) \subseteq E(s)$.

The new FSA will be $\mathcal{B} = \langle S_\mathcal{B}, b_0, A, \rho_\mathcal{B}, F_\mathcal{B}\rangle$. Let $b_0 = E(a_0)$. The components $S_\mathcal{B}$ and $\rho_\mathcal{B}$ are described first, with each item in $S_\mathcal{B}$ being a subset of $S_\mathcal{A}$, that is $S_\mathcal{B} \subseteq \mathcal{P}(S_\mathcal{A})$. Start with $S_\mathcal{B} = Z_0 = \{E(s_0)\}$ and proceed inductively. Given Z_i, begin another inductive cycle with $Z_{i+1} = \emptyset$. Next, for all $x \in A$ and all $Q_1 \in Z_i$, compute

$$Q_2 = \bigcup_{p \in Q_1} \big\{E(q) \,|\, (p, x, q) \in \rho_\mathcal{A}\big\}. \tag{1}$$

If $Q_2 \neq \emptyset$ and Q_2 is not already in $S_\mathcal{B}$: (i) include Q_2 in Z_{i+1}, and in $S_\mathcal{B}$; and (ii) add (Q_1, x, Q_2) to $\rho_\mathcal{B}$. If this inductive cycle terminates with $Z_{i+1} = \emptyset$ then the construction is complete; otherwise repeat the inductive cycle. After the last inductive cycle terminates we have $S_\mathcal{B}$ and $\rho_\mathcal{B}$. We complete the construction of $\mathcal{B}$ defining the set of final states as

$$F_\mathcal{B} = \{Q \in S_\mathcal{B} \mid Q \cap F_\mathcal{A} \neq \emptyset\}. \tag{2}$$

Clearly, by construction, there are no ε-moves in $\mathcal{B}$. Since each state Q is added to $S_\mathcal{B}$ only once, Eq. (1) guarantees that (Q, x, Q_1), $(Q, x, Q_2) \in \rho_\mathcal{B}$ imply $Q_1 = Q_2$, for all $x \in A$. We, thus, conclude that $\mathcal{B}$ is a deterministic FSA. Also, it is easy to see, inductively, that if $q \in Q$ then $E(q) \subseteq Q$, for all $q \in S$ and all $Q \in S_\mathcal{B}$.

An easy induction of $k \geq 0$ shows that if $(Q_1, \sigma_1) \mid\frac{k}{\mathcal{B}} (Q_2, \sigma_2)$ and $q_2 \in Q_2$, then there is $q_1 \in Q_1$ such that $(q_1, \sigma_1) \mid\frac{\star}{\mathcal{A}} (q_2, \sigma_2)$, for all σ_1, $\sigma_2 \in A^\star$. So, if $\sigma \in L(\mathcal{B})$ then $(b_0, \sigma) \mid\frac{\star}{\mathcal{B}} (Q, \varepsilon)$ for some $Q \in F_\mathcal{B}$, which gives $Q \cap F_\mathcal{A} \neq \emptyset$, so that we can choose some $f \in Q$ and $f \in F_\mathcal{A}$. By the inductive argument, we get some $q \in b_0$ such that $(q, \sigma) \mid\frac{\star}{\mathcal{A}} (f, \varepsilon)$. But $b_0 = E(a_0)$ and so we have $(a_0, \varepsilon) \mid\frac{\star}{\mathcal{A}} (q, \varepsilon)$. Composing, we get $(a_0, \sigma) \mid\frac{\star}{\mathcal{A}} (q, \sigma) \mid\frac{\star}{\mathcal{A}} (f, \varepsilon)$ and then $\sigma \in L(\mathcal{A})$ because $f \in F_\mathcal{A}$. This shows $L(\mathcal{B}) \subseteq L(\mathcal{A})$.

For the converse, first note that if $\varepsilon \in L(\mathcal{B})$, then we have $(b_0, \varepsilon) \mid\frac{\star}{\mathcal{B}} (Q, \varepsilon)$ and $Q \in F_\mathcal{B}$. Hence, there is $f \in Q \cap F_\mathcal{A}$. Since $\mathcal{B}$ has no ε-moves, we get $Q = b_0$ and so $f \in Q$ gives $f \in E(a_0)$. Thus, $(a_0, \varepsilon) \mid\frac{\star}{\mathcal{A}} (f, \varepsilon)$, and so $\varepsilon \in L(\mathcal{A})$ because $f \in F_\mathcal{A}$. Next, let $\sigma \in A^\star$, with $|\sigma| \geq 1$ and $(q_1 \sigma) \mid\frac{\star}{\mathcal{A}} (q_2, \varepsilon)$. A simple induction on $|\sigma|$ shows that if $Q_1 \in S_\mathcal{B}$ with $q_1 \in Q_1$, then there is some $Q_2 \in S_\mathcal{B}$ with $q_2 \in Q_2$ and such that $(Q_1, \sigma) \mid\frac{\star}{\mathcal{B}} (Q_2, \varepsilon)$. Now, if $|\sigma| \geq 1$ and $\sigma \in L(\mathcal{A})$ we get $(a_0, \sigma) \mid\frac{\star}{\mathcal{A}} (f, \varepsilon)$, with $f \in F_\mathcal{A}$. Because $a_0 \in b_0$, the inductive argument gives some $Q \in S_\mathcal{B}$ such that $f \in Q$ and $(b_0, \sigma) \mid\frac{\star}{\mathcal{B}} (Q, \varepsilon)$. So, $Q \cap F_\mathcal{A} \neq \emptyset$ and we have $Q \in F_\mathcal{B}$. Hence, $\sigma \in L(\mathcal{B})$, showing that $L(\mathcal{A}) \subseteq L(\mathcal{B})$. We now have $L(\mathcal{A}) = L(\mathcal{B})$, as desired.

We now assume that all states in $\mathcal{A} = \langle S_\mathcal{A}, a_0, A, \rho_\mathcal{A}, F_\mathcal{A} \rangle$ are final states, that is, $F_\mathcal{A} = S_\mathcal{A}$. Since all states in the equivalent FSA $\mathcal{B}$ are non-empty subsets of $S_\mathcal{A}$, it is clear from Eq. (2) that $Q \cap F_\mathcal{A} \neq \emptyset$ for all states Q of $\mathcal{B}$, that is, Q is also a final state in $\mathcal{B}$. $\square$

From a deterministic FSA we can easily get an equivalent complete FSA with at most one extra state. We just add a transition to that extra state in the new FSA whenever a transition is missing in the original FSA.

Proposition 2.22. *Let $\mathcal{A} = \langle S_\mathcal{A}, s_0, A, \rho_\mathcal{A}, F \rangle$ be a deterministic FSA. Then we can effectively construct a complete FSA $\mathcal{B} = \langle S_\mathcal{B}, s_0, A, \rho_\mathcal{B}, F \rangle$ such that $L(\mathcal{B}) = L(\mathcal{A})$, and with $|S_\mathcal{B}| = |S_\mathcal{A}| + 1$.*

Demonstração. Let $S_\mathcal{B} = S_\mathcal{A} \cup \{e\}$, where $e \notin S_\mathcal{A}$, and let

$$\rho_\mathcal{B} = \rho_\mathcal{A} \cup \{(e, \ell, e) : \text{for any } \ell \in A\} \cup \{(s, \ell, e) : \text{if } (s, \ell, p) \notin \rho_\mathcal{A} \text{ for all } p \in \mathcal{S}_A\}.$$

It is clear that $\mathcal{B}$ is a complete FSA. A simple induction on $|\sigma| \geq 0$ shows that $\sigma \in L(\mathcal{A})$ if and only if $\sigma \in L(\mathcal{B})$, for all $\sigma \in A^\star$. $\square$

The following simple results will be useful later on.

Proposition 2.23. *Let L be a set of symbols, and let $\mathcal{A} = \langle S_\mathcal{A}, a_0, L, \rho_\mathcal{A}, F_\mathcal{A} \rangle$, $\mathcal{B} = \langle S_\mathcal{B}, b_0, L, \rho_\mathcal{B}, F_\mathcal{B} \rangle$ be complete FSAs. Then,*

1. *We can effectively construct a complete FSA $\mathcal{T} = \langle S_\mathcal{T}, t_0, L, \rho_\mathcal{T}, F_\mathcal{T} \rangle$ such that $L(\mathcal{T}) = L(\mathcal{A}) \cap L(\mathcal{B})$.*

2. *We can effectively construct a complete FSA $\mathcal{T} = \langle S_\mathcal{T}, t_0, L, \rho_\mathcal{T}, F_\mathcal{T}\rangle$ such that $L(\mathcal{T}) = L(\mathcal{A}) \cup L(\mathcal{B})$.*

Moreover, in both cases, we have $|S_\mathcal{T}| \le |S_\mathcal{A}| \times |S_\mathcal{B}|$.

Demonstração. Define $t_0 = (a_0, b_0)$, and let $S_\mathcal{T} \subseteq S_\mathcal{A} \times S_\mathcal{B}$, where $(s_1, s_2) \in S_\mathcal{T}$ if and only if $a_0 \overset{\sigma}{\underset{\mathcal{A}}{\mapsto}} s_1$ and $b_0 \overset{\sigma}{\underset{\mathcal{A}}{\mapsto}} s_2$ for some $\sigma \in A^\star$. Clearly, $|S_\mathcal{T}| \le |S_\mathcal{A}| \times |S_\mathcal{B}|$. Now, for any (s_1, q_1), $(s_2, q_2) \in S_\mathcal{T}$, let $\big((s_1, q_1), x, (s_2, q_2)\big) \in \rho_\mathcal{T}$ if and only if $(s_1, x, s_2) \in \rho_\mathcal{A}$ and $(q_1, x, q_2) \in \rho_\mathcal{B}$, for all $x \in A$. Clearly, because $\mathcal{A}$ and $\mathcal{B}$ are complete, it is easily seen that $\mathcal{T}$ is also complete. Moreover, a simple induction on $|\sigma| \ge 0$ gives $(s_1, q_1) \overset{\sigma}{\underset{\mathcal{T}}{\mapsto}} (s_2, q_2)$ if and only if $s_1 \overset{\sigma}{\underset{\mathcal{A}}{\mapsto}} s_2$ and $q_1 \overset{\sigma}{\underset{\mathcal{B}}{\mapsto}} q_2$, for all $\sigma \in A^\star$.

For the first item, define $F_\mathcal{T} = \{(s, q) \mid s \in F_\mathcal{A} \text{ and } q \in F_\mathcal{B}\}$. Using the induction, we readily get $L(\mathcal{T}) = L(\mathcal{A}) \cap L(\mathcal{B})$. For the second item let $F_\mathcal{T} = \{(s, q) \mid s \in F_\mathcal{A} \text{ or } q \in F_\mathcal{B}\}$ and using the induction again we get $L(\mathcal{T}) = L(\mathcal{A}) \cup L(\mathcal{B})$. □

Any LTS $\mathcal{S}$ gives rise to an associated FSA in a natural way. We convert any transition of $\mathcal{S}$ labeled by the internal action τ into a ε-move in the FSA, and we make the set of final states of the FSA as the set of all locations of the LTS $\mathcal{S}$. We can also associate a LTS to any FSA in a direct way, provided that all states are final states in the FSA.

Definition 2.24. *We have the following two associations:*

1. *Let $\mathcal{S} = \langle S, s_0, L, T\rangle$ be a LTS. The FSA* induced *by $\mathcal{S}$ is $\mathcal{A}_\mathcal{S} = \langle S, s_0, L, \rho, S\rangle$ where, for all p, $q \in S$ and all $\ell \in L$, we have*

$$(p, \ell, q) \in \rho \quad \textit{if and only if} \quad (p, \ell, q) \in T,$$

$$(p, \varepsilon, q) \in \rho \quad \textit{if and only if} \quad (p, \tau, q) \in T.$$

2. *Let $\mathcal{A} = \langle S, s_0, A, \rho, S\rangle$ be a FSA. The LTS* induced *by $\mathcal{A}$ is $\mathcal{S}_\mathcal{A} = \langle S, s_0, A, T\rangle$ where, for all p, $q \in S$ and all $a \in A$, we have*

$$(p, a, q) \in T \quad \textit{if and only if} \quad (p, a, q) \in \rho,$$

$$(p, \tau, q) \in T \quad \textit{if and only if} \quad (p, \varepsilon, q) \in \rho.$$

The observable semantics of $\mathcal{S}$ is just the language accepted by $A_\mathcal{S}$. We note this as the next proposition.

Proposition 2.25. *Let $\mathcal{S} = \langle S, s_0, L, T\rangle$ be a LTS and let $\mathcal{A}_\mathcal{S}$ be the FSA induced by $\mathcal{S}$. Then we have $otr(\mathcal{S}) = L(\mathcal{A}_\mathcal{S})$. Conversely, if $\mathcal{A} = \langle S, s_0, A, \rho, S\rangle$ is a FSA and $\mathcal{S}_\mathcal{A}$ is the LTS induced by $\mathcal{A}$, then $L(\mathcal{A}) = otr(\mathcal{S}_\mathcal{A})$.*

Demonstração. For the first assertion, an easy induction on the length of $\mu \in L_\tau^\star$ shows that $\mu \in tr(\mathcal{S})$ if and only if $h_\tau(\mu) \in L(\mathcal{A}_\mathcal{S})$, and, by Definition 2.8, we know that $\sigma \in otr(\mathcal{S})$ if and only if there is some μ such that $\sigma = h_\tau(\mu)$. The second assertion follows by a similar reasoning. □

With Definition 2.17, we now know that $otr(\mathcal{S})$ is a regular language. It is also clear now that for any LTS $\mathcal{S}$, we can easily construct a FSA $\mathcal{A}$ whose semantics is just $otr(\mathcal{S})$.

We can eliminate all τ-labeled moves in any LTS and obtain another LTS with the same observable semantics as the original LTS. More formally, we can ascertain the following result.

Proposition 2.26. *Let $\mathcal{S}$ be a LTS. Then there is a deterministic LTS $\mathcal{T}$ such that $otr(\mathcal{S}) = tr(\mathcal{T}) = otr(\mathcal{T})$. Further, $\mathcal{T}$ can be effectively constructed from $\mathcal{S}$.*

Demonstração. Consider the FSA $\mathcal{A}_{\mathcal{S}}$ induced by $\mathcal{S}$. From Proposition 2.25 we know that $L(\mathcal{A}_{\mathcal{S}}) = otr(\mathcal{S})$. Use Proposition 2.21 to get a deterministic FSA $\mathcal{B}$ such that $L(\mathcal{A}_{\mathcal{S}}) = L(\mathcal{B})$. Hence $otr(\mathcal{S}) = L(\mathcal{B})$. From Definition 2.24 we know that all states of $\mathcal{A}_{\mathcal{S}}$ are final states and so, from Proposition 2.21 it follows that the same is true of $\mathcal{B}$. Thus, from Definition 2.24 again, we can get the LTS $\mathcal{S}_{\mathcal{B}}$, induced by $\mathcal{B}$. By Proposition 2.25 again we now have $L(\mathcal{B}) = otr(\mathcal{S}_{\mathcal{B}})$, and so $otr(\mathcal{S}) = otr(\mathcal{S}_{\mathcal{B}})$. Moreover, since $\mathcal{B}$ is a deterministic FSA, it is clear from Definitions 2.24, 2.20, and 2.12 that $\mathcal{S}_{\mathcal{B}}$ is also deterministic. Hence, $\mathcal{S}_{\mathcal{B}}$ has no τ-labeled transitions, and so Remark 2.9 says that $tr(\mathcal{S}_{\mathcal{B}}) = otr(\mathcal{S}_{\mathcal{B}})$. Putting it all together, we have that $\mathcal{S}_{\mathcal{B}}$ is a deterministic LTS with $otr(\mathcal{S}) = tr(\mathcal{S}_{\mathcal{B}}) = otr(\mathcal{S}_{\mathcal{B}})$, as desired.

Since the construction at Proposition 2.21 is effective, we see that $\mathcal{S}_{\mathcal{B}}$ can also be effectively constructed from $\mathcal{S}$. □

If $\mathcal{T}$ is the LTS constructed from a LTS $\mathcal{S}$ in Proposition 2.26, then $p \overset{\sigma}{\Rightarrow} q$ in $\mathcal{S}$ if and only if $p \overset{\sigma}{\rightarrow} q$ in $\mathcal{T}$ if and only if $p \overset{\sigma}{\Rightarrow} q$ in $\mathcal{T}$. Moreover, $\mathcal{T}$ has no τ-labeled transitions, and if $p \overset{\sigma}{\Rightarrow} q$ in $\mathcal{T}$, then q is a unique state, given p and σ.

2.4 Input Output Transition Systems

In many situations, we wish treat some action labels as symbols that the LTS "receives" from the environment, and some other set of action labels as symbols that the LTS "sends back" to the environment, when the LTS models de facto a reactive system. The next LTS variation differentiates between input action symbols and output action symbols.

Definition 2.27. *An Input/Output Labeled Transition System (IOLTS) is a tuple $\mathcal{I} = \langle S, s_0, L_I, L_U, T\rangle$, where*

- *L_I is a countable set of* input actions*, or* input labels*;*
- *L_U is a countable set of* output actions*, or* output labels*;*
- *$L_I \cap L_U = \emptyset$, and $L = L_I \cup L_U$ is the set of* actions *or* labels*; and*
- *$\langle S, s_0, L, T\rangle$ is a LTS, the* underlying LTS *associated to $\mathcal{I}$.*

We indicate the underlying LTS of an IOLTS $\mathcal{I} = \langle S, s_0, L_I, L_U, T\rangle$ by $\mathcal{S}_{\mathcal{I}} = \langle S, s_0, L, T\rangle$, and we denote the class of all IOLTSs with input alphabet L_I and output alphabet L_U by $\mathcal{IO}(L_I, L_U)$. In order to ease the notation, in specific examples a symbol in L_I can be denoted by following it with a "?" mark, and a symbol in L_U can be denoted by following it with a "!" mark. Then $a? \in L_I$ and $a! \in L_U$ are distinct symbols. We also note that other works, *e.g.*, [27], impose additional restrictions to the basic IOLTS model, but we do not need to further restrict the models at this point.

Several notions involving IOLTSs will be defined by a direct reference to their underlying LTSs. The semantics of an IOLTS is just the set of its observable traces, that is, observable traces of its underlying LTS.

Definition 2.28. *Let $\mathcal{I} = \langle S, s_0, L_I, L_U, T\rangle$ be an IOLTS. The* semantics *of $\mathcal{I}$ is the set $otr(\mathcal{I}) = otr(\mathcal{S}_{\mathcal{I}})$, where $\mathcal{S}_{\mathcal{I}}$ is the underlying LTS associated to $\mathcal{I}$.*

Also, when referring an IOLTS $\mathcal{I}$, 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 associated to $\mathcal{I}$.

We see that an IOLTS generalizes the simpler model known as a Mealy machine [10]. In the latter, for each transition we get exactly one input action from the environment, and the machine responds simultaneously with exactly one output action to the environment. Hence communications in this model is synchronous. In an IOLTS model, input labels and output labels are exchanged asynchronously. This facilitates the specification of more complex behaviors.

Example 2.29. Figure 4 represents an IOLTS, adapted from [16]. The model represents the

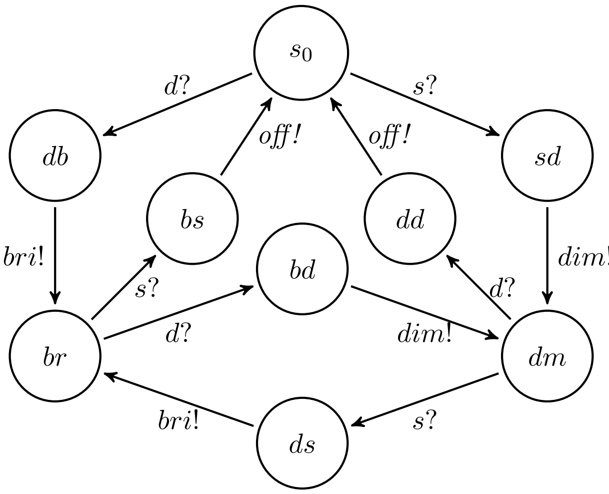

Figura 4: An IOTS.

lightening of a bulb. It has the two input actions s and d representing a single or a double click on the lamp switch, respectively. It has three output labels, *dim*, *bri*, and *off*, informing the environment that the illumination turned to dim, bright or off, respectively. The initial state is s_0. Following the rightmost circuit, starting at s_0, when the user — that is, the environment — hits a single click on the switch the system moves from state s_0 to state sd on the input action $s?$. This is represented by the transition $s_0 \xrightarrow{s?} sd$. Then, following the transition $sd \xrightarrow{dim!} dm$, the system reaches the state dm and outputs the label $dim!$, informing the user that the illumination is now dimmed. At state dm if the user double clicks at the switch, the system responds with $off!$ and moves back to state s_0. This corresponds to the transitions $dm \xrightarrow{d?} dd$ and $dd \xrightarrow{off!} s_0$. But, if at state dm the user clicks only once then the transitions $dm \xrightarrow{s?} ds$ and $ds \xrightarrow{bri!} br$ are traversed, and the system moves to state br issuing the output $bri!$ to signal that the lamp is now in the bright mode. □

Example 2.30. Figure 5 represents another IOLTS, adapted from [27]. It describes a strange coffee machine. When the user hits the start button — represented by the input label *but* — the machine chooses to go either to state s_1 or to state s_2. If it goes to state s_1, no matter how many extra times the user hits the button the loop labeled $but?$ at state s_1 is be traversed, and the machine dispenses a cup of tea, signaled by the output label *tea*. Then the machine performs an internal action and moves to the start state again. This corresponds to the transition $tea \xrightarrow{\tau} s_0$. If it chooses to follow down the left branch from the start state the situation is similar, but now the machine will dispense a cup of coffee, indicated by the output label *coffee*, and another internal action moves it back to the start state again. □

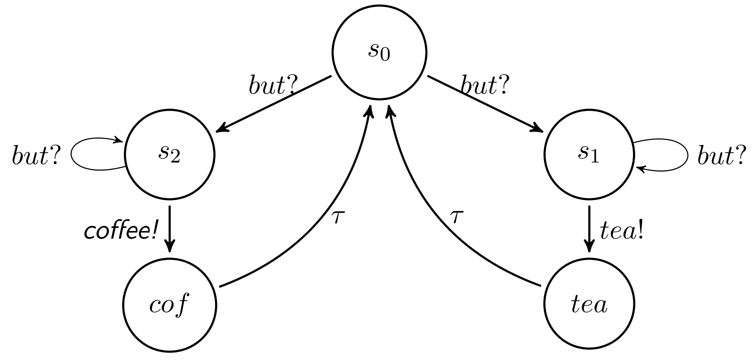

Figura 5: Another IOTS.

2.5 Quiescent States

When treating the **ioco** conformance relation, some works [25, 27] distinguish the so called quiescent states. Informally, those are states from which there are no transitions labeled by some output action symbol in IUTs. This motivates the following definitions. The function **inp** will be useful in the sequel.

Definition 2.31. *Let $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle$ be an IOLTS. Define the functions* **out**, **inp** *and* **init** *from $\mathcal{P}(S)$ into $L_U \cup L_I$, respectively, as* $\textbf{out}(V) = \bigcup_{s\in V} \{\ell \in L_U \mid s \overset{\ell}{\Rightarrow}\}$, $\textbf{inp}(V) = \bigcup_{s\in V} \{\ell \in L_I \mid s \overset{\ell}{\Rightarrow}\}$, *and* $\textbf{init}(V) = \textbf{inp}(V) \cup \textbf{out}(V)$, *for all $V \subseteq S$. A state $s \in S$ is said to be* quiescent *if* $\textbf{out}(\{s\}) = \emptyset$.

We may also write $\textbf{out}(s)$, $\textbf{inp}(s)$ and $\textbf{init}(s)$ instead of $\textbf{out}(\{s\})$, $\textbf{inp}(\{s\})$ and $\textbf{init}(\{s\})$, respectively.

We can imagine a setting where there is a "tester", modeled by an IOLTS $\mathcal{T}$, and an implementation being tested, modeled by an IOLTS $\mathcal{I}$. In this setting, the tester, or an "artificial environment", $\mathcal{T}$ issues one of its output action symbols x to $\mathcal{I}$, which simultaneously accepts x as one of its input symbols. In the reverse direction, the implementation $\mathcal{I}$ may respond with one of its output symbols y, that is simultaneously accepted by $\mathcal{T}$ as on of its input symbols. That, is $\mathcal{T}$ and $\mathcal{I}$ move synchronously, but with input and output sets of symbols interchanged. We will always refer to a symbol x as an input or an output symbol from the perspective of the implementation $\mathcal{I}$, unless there is an explicit mention to the contrary. Hence, one should write $\mathcal{I} = \langle S, s_0, L_I, L_U, T\rangle$ and $\mathcal{T} = \langle Q, q_0, L_U, L_I, R\rangle$.

As a result of accepting an input x from $\mathcal{T}$, the implementation $\mathcal{I}$ may reach a quiescent state s. In a practical scenario, from this point on the implementation could no longer send responses back to the tester, and the latter will have no way of "knowing" whether the implementation 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 the tester that the implementation is in a quiescent state. A usual mechanism [27] is to imagine that the implementation has a special output symbol $\delta \notin L_I \cup L_U$, $\delta \neq \tau$, and that δ is then sent back to $\mathcal{T}$ when $\mathcal{I}$ reaches state s. Since $\mathcal{I}$ is not changing states in this situation, we include the loop $s \xrightarrow{\delta} s$ into the set T of transitions of $\mathcal{I}$. On the tester side, being on a state $q \in Q$ and upon receiving a δ symbol from the implementation, the tester may decide whether receiving such a signal in state q is appropriate or not, depending on the fault model it was designed for. If that response from the implementation was an adequate one, the tester may then move to another state q' to continue the test run. That is, on the tester side we add transitions[1] $q \xrightarrow{\delta} q'$ between specific states $q, q' \in Q$. But note that we are still allowing both the tester $\mathcal{T}$ and the implementation $\mathcal{I}$ to freely move asynchronously along any number of internal τ-labeled transitions, after reaching states with a δ self-loop.

When we need to explicitly refer to an IOLTS $\mathcal{S} = \langle S, s_0, L_I, L_U \cup \{\delta\}, T\rangle$, with $\delta \notin L_I \cup L_U$, that is equipped with quiescent δ labeled transitions, we may refer to $\mathcal{S}$ as a δ-IOLTS.

Remark 2.32. *Note that, for any effect within the formalism, a δ-IOLTS is still an ordinary IOLTS, only with its output alphabet extended by the inclusion of a new distinct symbol δ.*

3 Conformance testing

In this section we define a generalized notion of a conformance relation. We also relate such a generalization to the classical **ioco** conformance relation [27].

3.1 The General Conformance Relation

Here we give a more general conformance relation for language-based testing. Informally, we consider a language D, the set of "desirable", or "allowed", behaviors, and a language F, the set of "forbidden", or "undesirable", behaviors of a system. If we have a specification LTS $\mathcal{S}$ and an implementation LTS $\mathcal{I}$ we want to say that $\mathcal{I}$ *conforms* to $\mathcal{S}$ according to (D, F) if no undesired behavior in F that is observable in $\mathcal{I}$ is specified in $\mathcal{S}$, and all desired behaviors in D that are observable in $\mathcal{I}$ are specified in $\mathcal{S}$. This leads to the following definition of conformance.

Definition 3.1. *Let L be a set of symbols, and let $D, F \subseteq L^\star$ be languages over L. Let $\mathcal{S}$ and $\mathcal{I}$ be LTSs with L as their set of labels. We say that $\mathcal{I}$ (D, F)-*conforms to $\mathcal{S}$*, written $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$, if and only if*

1. *$\sigma \in otr(\mathcal{I}) \cap F$, then $\sigma \notin otr(\mathcal{S})$;*
2. *$\sigma \in otr(\mathcal{I}) \cap D$, then $\sigma \in otr(\mathcal{S})$.*

We note an equivalent way of expressing these conditions that may also be useful. Write $\overline{otr}(\mathcal{S})$ for the complement of $otr(\mathcal{S})$, that is, $\overline{otr}(\mathcal{S}) = L^\star - otr(\mathcal{S})$. We show when an implementation conforms to a given specification according to the desirable behaviors defined by language D and the undesirable behaviors given by language F.

Proposition 3.2. *Let $\mathcal{S}$ and $\mathcal{I}$ be LTSs with L as their set of labels and let $D, F \subseteq L^\star$ be languages over L. Then $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ if and only if*

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

Demonstração. From Definition 3.1 we readily get $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ if and only if $otr(\mathcal{I}) \cap F \cap otr(\mathcal{S}) = \emptyset$ and $otr(\mathcal{I}) \cap D \cap \overline{otr}(\mathcal{S}) = \emptyset$. And the statement holds if and only if

$$\emptyset = \big[otr(\mathcal{I}) \cap F \cap otr(\mathcal{S})\big] \cup \big[otr(\mathcal{I}) \cap D \cap \overline{otr}(\mathcal{S})\big] = otr(\mathcal{I}) \cap \big[(D \cap \overline{otr}(\mathcal{S})) \cup (F \cap otr(\mathcal{S}))\big],$$

as desired. □

Example 3.3. Let $\mathcal{S}$ be a specification depicted in Figure 6 and also let $\mathcal{I}$ be an implementation as depicted in Figure 8 with $L = \{a, b, x\}$. Take the languages $D = (a + b)^\star ax$ and $F = (a + b)^\star bx$. We want to check whether $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ holds.

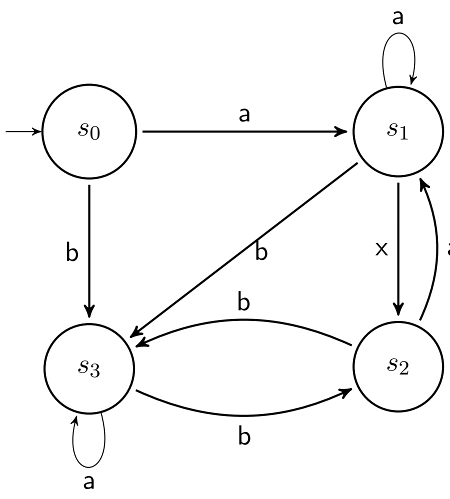

Figura 6: A LTS specification $\mathcal{S}$.

The condition $F \cap otr(\mathcal{S}) = \emptyset$ holds since F represents words that end in bx, and bx is not a suffix of any observable behavior of $\mathcal{S}$.

On the other hand, $\overline{otr}(\mathcal{S})$ is the language accepted by the FSA $\overline{\mathcal{S}}$, depicted in Figure 7. A simple inspection shows bax is accepted by $\overline{\mathcal{S}}$, and also, clearly, $bax \in D$. Hence $bax \in D \cap \overline{otr}(\mathcal{S})$. Also, from Figure 8, we get $q_0 \xrightarrow{b} q_3 \xrightarrow{ax} q_2$, and so bax is an observable behavior of $\mathcal{I}$. We conclude that $otr(\mathcal{I}) \cap D \cap \overline{otr}(\mathcal{S}) \neq \emptyset$ which means that $\mathcal{I}\,\mathbf{conf}_{D,F}\ \mathcal{S}$ does not hold.

On the other hand, if we assume an implementation $\mathcal{I}$ that is isomorphic to $\mathcal{S}$, $\mathcal{I}$ would not have the transition $q_3 \xrightarrow{x} q_2$ and then bax would not be an observable behavior of $\mathcal{I}$. Actually, in this case, $otr(\mathcal{I}) \cap D \cap \overline{otr}(\mathcal{S}) = \emptyset$. So that now

$$otr(\mathcal{I}) \cap \big[(D \cap \overline{otr}(\mathcal{S})) \cup (F \cap otr(\mathcal{S}))\big] = \emptyset,$$

and therefore $\mathcal{I}\,\mathbf{conf}_{D,F}\ \mathcal{S}$, as expected. $\square$

Depending on languages D and F, we can have several specific notions of conformance:

1. All observable behaviors of $\mathcal{I}$ are of interest, and we are not concerned with any undesirable behaviors of $\mathcal{I}$. Then, 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 $\mathcal{I} = \langle Q, q_0, L, R\rangle$ and let $C \subseteq Q$ be a subset of the locations of $\mathcal{I}$. Behaviors of interest are all observable traces σ of $\mathcal{I}$ that take the initial location s_0 of $\mathcal{I}$ to a location in C, that is $s_0 \overset{\sigma}{\Rightarrow} q$ and $q \in C$. Also, let E be another set of locations of $\mathcal{I}$ with $E \cap C = \emptyset$, and the undesired behaviors of $\mathcal{I}$ are observable traces σ that lead to a location in E. Then, $\mathcal{I}\,\mathbf{conf}_{C,E}\ \mathcal{S}$ if and only if undesired observable traces of $\mathcal{I}$ are not observable in $\mathcal{S}$ and all desirable observable traces of $\mathcal{I}$ are also observable in $\mathcal{S}$.

3. Let $\mathcal{S} = \langle S, s_0, L, T\rangle$ be a specification and let $\mathcal{I} = \langle Q, q_0, L, R\rangle$ be an implementation. Let $H \subseteq L$ be a subset of L. The desirable behaviors of $\mathcal{I}$ are those observable traces that end in a label in H, and we are not interested in undesirable traces of $\mathcal{I}$. In this case, choose $F = \emptyset$ and $D = L^\star \cdot H$.

[1]In [27], a different symbol, θ, was used to signal the acceptance of quiescence on the tester side, but for our formalism, that makes little difference, if any.

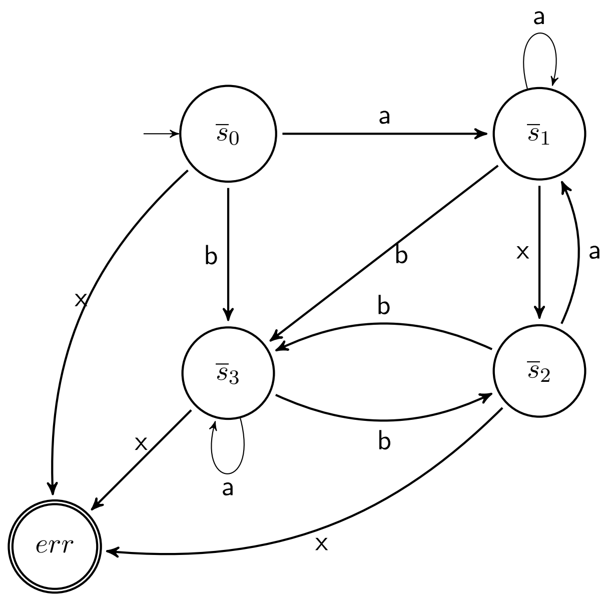

Figura 7: A FSA accepting $\overline{otr}(\mathcal{S})$ for the LTS $\mathcal{S}$ shown in Figure 6.

It is readily seen that both D and F are regular languages, in all cases listed above.

If an implementation $\mathcal{I}$ conforms to a specification $\mathcal{S}$ according to a pair of regular languages (D, F), then we can assume that D and F are, in fact, finite languages.

Corollary 3.4. *Let $\mathcal{S}$ and $\mathcal{I}$ be a specification and an implementation LTS over an alphabet L, respectively, and let D and F be languages over L. Then, there are finite languages $D' \subseteq D$ and $F' \subseteq F$ such that $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ if and only if $\mathcal{I}\,\mathbf{conf}_{D',F'}\,\mathcal{S}$.*

Demonstração. By Proposition 3.2, $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ if and only if $otr(\mathcal{I}) \cap F \subseteq \overline{otr}(\mathcal{S})$ and $otr(\mathcal{I}) \cap D \subseteq otr(\mathcal{S})$. Lemma 2.2 gives a finite language $D' \subseteq D$ such that $otr(\mathcal{I}) \cap D' \subseteq otr(\mathcal{S})$ if and only if $otr(\mathcal{I}) \cap D \subseteq otr(\mathcal{S})$. Likewise, we can get a finite language $F' \subseteq F$ such that $otr(\mathcal{I}) \cap F' \subseteq \overline{otr}(\mathcal{S})$ if and only if $otr(\mathcal{I}) \cap F \subseteq \overline{otr}(\mathcal{S})$. By Proposition 3.2 again, this holds if and only if $\mathcal{I}\,\mathbf{conf}_{D',F'}\,\mathcal{S}$, as desired. □

Example 3.5. Let $\mathcal{S}$ and $\mathcal{I}$ be a specification and an implementation, again as depicted in Figures 6 and 8, respectively. Also assume the same languages $D = (a + b)^\star ax$ and $F = (a + b)^\star bx$. From Example 3.3 we know that $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ does not hold.

Take the finite languages $D' = \{bax\}$ and $F' = \emptyset$. Clearly, $D' \subseteq D$ and $F' \subseteq F$. It is clear that $bax \in otr(\mathcal{I})$, and $bax \notin otr(\mathcal{S})$, so that $bax \in otr(\mathcal{I}) \cap D' \cap \overline{otr}(\mathcal{S})$. So, by Proposition 3.2, $\mathcal{I}\,\mathbf{conf}_{D',F'}\,\mathcal{S}$ does not hold, as desired. □

3.2 The ioco Conformance Relation

Here we state the classical notion of **ioco** conformance relation [27]. Let $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle$ be a specification IOLTS and let $\mathcal{I} = \langle Q, q_0, L_I, L_U, R\rangle$ be an implementation IOLTS. The **ioco** relation essentially requires that any observable trace σ of $\mathcal{I}$ is also an observable trace of $\mathcal{S}$ and, further, if σ leads $\mathcal{I}$ to a location from which $\mathcal{I}$ can emit the output label ℓ, then $\mathcal{S}$ must also end up in a location from which the label ℓ can also be output. Recalling Definition 2.31, we formalize these ideas next.

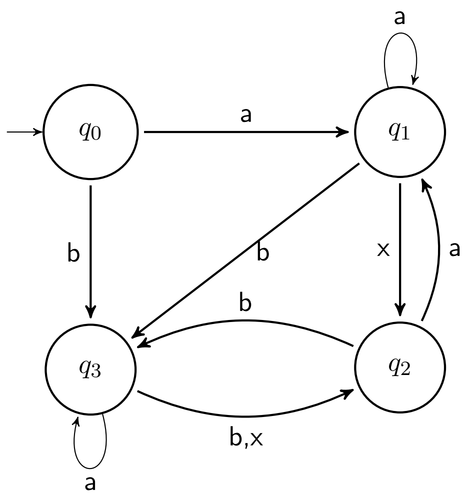

Figura 8: A LTS implementation $\mathcal{I}$.

Definition 3.6 ([27])**.** *Let $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle$ and $\mathcal{I} = \langle Q, q_0, L_I, L_U, R\rangle$ be IOLTSs, with $L = L_I \cup L_U$.*

1. *Define a function* $\textbf{after}\colon S \times L^\star \to \mathcal{P}(S)$ *by letting* $s\ \textbf{after}\ \sigma = \{q \mid s \overset{\sigma}{\Rightarrow} q\}$*, for all* $s \in S$*,* $\sigma \in otr(\mathcal{S})$*;*

2. *We say that* $\mathcal{I}\ \textbf{ioco}\ \mathcal{S}$ *if and only if for all* $\sigma \in otr(\mathcal{S})$ *we have* $\textbf{out}(q_0\ \textbf{after}\ \sigma) \subseteq \textbf{out}(s_0\ \textbf{after}\ \sigma)$*.*

Now we want to show that the more general notion of conformance relation given in Definition 3.1 restrains the classical **ioco** conformance relation [27].

Lemma 3.7. *Let $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle$ be a specification IOLTS and let $\mathcal{I} = \langle Q, q_0, L_I, L_U, R\rangle$ be an implementation IOLTS. Then $D = otr(\mathcal{S})L_U$ is a regular language over L, and we have that* $\mathcal{I}\ \textbf{ioco}\ \mathcal{S}$ *if and only if* $\mathcal{I}\,\textbf{conf}_{D,\emptyset}\ \mathcal{S}$*.*

Demonstração. From Definition 2.28 we know that the semantics of an IOLTS is given by the semantics of its underlying LTS. So, for the remainder of this proof, when we write $\mathcal{S}$ and $\mathcal{I}$ we will be referring to the underlying LTSs of the given IOLTSs $\mathcal{S}$ and $\mathcal{I}$, respectively.

By Proposition 2.25 and Remark 2.18 we see that $otr(\mathcal{S})$ and L_U are regular languages. Hence, by Proposition 2.19 we conclude that D is also a regular language.

Now, we show that $\mathcal{I}\ \textbf{ioco}\ \mathcal{S}$ if and only if $\mathcal{I}\,\textbf{conf}_{D,\emptyset}\ \mathcal{S}$. First assume that we have $\mathcal{I}\,\textbf{conf}_{D,\emptyset}\ \mathcal{S}$. Because $\mathcal{I} \cap \emptyset \cap \mathcal{S} = \emptyset$, it is clear from Definition 3.1 that $\mathcal{I}\,\textbf{conf}_{D,\emptyset}\ \mathcal{S}$ is equivalent to $otr(\mathcal{I}) \cap D \subseteq otr(\mathcal{S})$. In order to prove that $\mathcal{I}\ \textbf{ioco}\ \mathcal{S}$, let $\sigma \in otr(\mathcal{S})$ and let $\ell \in \textbf{out}(q_0\ \textbf{after}\ \sigma)$. We must show that $\ell \in \textbf{out}(s_0\ \textbf{after}\ \sigma)$. Because $\ell \in \textbf{out}(q_0\ \textbf{after}\ \sigma)$ 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 $\sigma\ell \in D$. We conclude that $\sigma\ell \in otr(\mathcal{I}) \cap D$. Since we already know that $otr(\mathcal{I}) \cap D \subseteq otr(\mathcal{S})$, we now have $\sigma\ell \in otr(\mathcal{S})$. So, $\ell \in \textbf{out}(s_0\ \textbf{after}\ \sigma)$, as desired.

Next, assume that $\mathcal{I}\ \textbf{ioco}\ \mathcal{S}$ and we want to show that $\mathcal{I}\,\textbf{conf}_{D,\emptyset}\ \mathcal{S}$ holds. Since $otr(\mathcal{I}) \cap \emptyset \cap otr(\mathcal{S}) = \emptyset$, the first condition of Definition 3.1 is immediately verified. We now turn to the second condition of Definition 3.1. In order to show that $otr(\mathcal{I}) \cap D \subseteq otr(\mathcal{S})$, let $\sigma \in otr(\mathcal{I}) \cap D$. Then, $\sigma \in D$ and so $\sigma = \alpha\ell$ with $\ell \in L_U$ and $\alpha \in otr(\mathcal{S})$, because $D = otr(\mathcal{S})L_U$. Also, $\sigma \in otr(\mathcal{I})$ gives $\alpha\ell \in otr(\mathcal{I})$, and so $\alpha \in otr(\mathcal{I})$. Then, because $\ell \in L_U$, we get $\ell \in \textbf{out}(q_0\ \textbf{after}\ \alpha)$. Because we assumed $\mathcal{I}$ **ioco** $\mathcal{S}$ and we have $\alpha \in otr(\mathcal{S})$, we also get $\ell \in \mathbf{out}(s_0 \textbf{ after } \alpha)$, and so $\alpha\ell \in otr(\mathcal{S})$. Because $\sigma = \alpha\ell$, we have $\sigma \in otr(\mathcal{S})$. We have, thus, showed that $otr(\mathcal{I}) \cap D \subseteq otr(\mathcal{S})$, as desired. □

Example 3.8. To illustrate the relationship between $\mathcal{I}$ **ioco** $\mathcal{S}$ and $\mathcal{I}\, \mathbf{conf}_{D,\emptyset}\, \mathcal{S}$, consider the machines $\mathcal{S}$ and $\mathcal{I}$, as depicted in Figures 6 and 8, respectively, and assume $L_I = \{a, b\}$, $L_U = \{x\}$. Figure 9 depicts a FSA $\mathcal{D}$ such that $L(\mathcal{D}) = D = otr(\mathcal{S})L_U$.

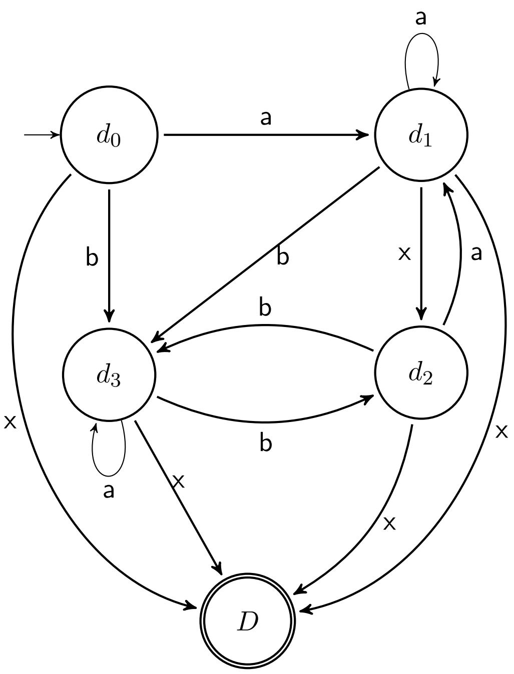

Figura 9: FSA $\mathcal{D}$ for the language $D = otr(\mathcal{S}) \cdot L_U$.

From Figures 6 and 8 it is apparent that $s_0 \overset{b}{\Rightarrow} s_3$ and $q_0 \overset{b}{\Rightarrow} q_3$. Also, $x \in \mathbf{out}(q_3)$, but $x \notin \mathbf{out}(s_3)$. So, by Definition 3.6, $\mathcal{I}$ **ioco** $\mathcal{S}$ does not hold.

On the other hand, take $\sigma = bx$. Since $b \in otr(\mathcal{S})$, we get $bx \in otr(\mathcal{S})L_U = D$. Also, $bx \in otr(\mathcal{I})$ and $bx \notin otr(\mathcal{S})$, so that $bx \in otr(\mathcal{I}) \cap D \cap \overline{otr}(\mathcal{S})$. Using Proposition 3.2, we conclude that $\mathcal{I}\, \mathbf{conf}_{D,\emptyset}\, \mathcal{S}$ does not hold also, as expected. □

In fact, we can take D to be a finite language.

Corollary 3.9. *Let $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle$ be a specification IOLTS and let $\mathcal{I} = \langle Q, q_0, L_I, L_U, R\rangle$ be an implementation IOLTS. Then there is a finite language $D \subseteq (L_I \cup L_U)^\star$ such that $\mathcal{I}$ **ioco** $\mathcal{S}$ if and only if $\mathcal{I}\, \mathbf{conf}_{D,\emptyset}\, \mathcal{S}$.*

Demonstração. Follows immediately from Lemma 3.7 and Corollary 3.4. □

We can also characterize the **ioco** relation as follows.

Corollary 3.10. *Let $\mathcal{S}$ be a specification IOLTS and let $\mathcal{I}$ be an implementation IOLTS. Then $\mathcal{I}$ **ioco** $\mathcal{S}$ if and only if $otr(\mathcal{I}) \cap T = \emptyset$, where $T = \overline{otr}(\mathcal{S}) \cap \big[otr(\mathcal{S})L_U\big]$.*

Demonstração. From Lemma 3.7 we have that $\mathcal{I}$ **ioco** $\mathcal{S}$ if and only if $\mathcal{I}\, \mathbf{conf}_{D,\emptyset}\, \mathcal{S}$, where $D = otr(\mathcal{S}) \cdot L_U$. From Proposition 3.2 we know that the latter holds if and only if $otr(\mathcal{I}) \cap (D \cap \overline{otr}(\mathcal{S})) = \emptyset$. □

4 Test Suite Generation for IOLTS models

In this section we say how to generate test suites as sets of words. We show that the generated test suites based on languages can decide implementation conformance with respect to a given specification.

4.1 Complete Test Suites Generation

We first state the general definition of a test suite.

Definition 4.1. *Let L be set of symbols. A test suite T over L is just a language over L, that is, $T \subseteq L^\star$. Each $\sigma \in T$ is called a* test case*.*

A test suite T should be geared to detect bad observable behaviors in an IUT, given a specification $\mathcal{S}$ and a pair of conforming languages (D, F). Clearly, when T is a regular language, then it could be specified by a FSA $\mathcal{A}$. The final states in $\mathcal{A}$ — the "fail" states — would then give the set of bad, or undesirable, behaviors to be considered. Equivalently, we could specify a finite set of such FSAs. Then, the union of all the undesirable behaviors specified by these FSAs would comprise the fault model.

We can now say that an implementation $\mathcal{I}$ satisfies, or adheres, to a test suite T when no observable behavior of $\mathcal{I}$ is a harmful behavior present in T.

Definition 4.2. *Let T be a test suite over L. A LTS $\mathcal{I}$* adheres *to T if and only if for all $\sigma \in otr(\mathcal{I})$ we have $\sigma \notin T$. Further, an IOLTS $\mathcal{I} = \langle S, s_0, L_I, L_U, T\rangle$ with $L = L_I \cup L_U$ adheres to T if and only if its underlying LTS adheres to T.*

In general, we want test suites to be sound, in the sense that adherence always implies conformance. Moreover, the converse is also desirable, that is, when we have conformance, then we also have adherence to the test suite.

Definition 4.3. *Let L be a set of symbols and let T be a test suite over L. Let $\mathcal{S}$ be a LTS over L, and let $D, F \subseteq L^\star$ be languages over L. We say that:*

1. *T is* sound *for $\mathcal{S}$ and (D, F) if $\mathcal{I}$ adheres to T implies $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$, for all LTS $\mathcal{I}$ over L.*

2. *T is* exhaustive *for $\mathcal{S}$ and (D, F) if $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ implies that $\mathcal{I}$ adheres to T, for all LTS $\mathcal{I}$ over L.*

3. *T is* complete *for $\mathcal{S}$ and (D, F) if it is both sound and exhaustive for $\mathcal{S}$ and (D, F).*

It comes as no surprise that the test suite we can extract from Proposition 3.2 is always complete. But, furthermore, we will also show that it is unique.

Lemma 4.4. *Let L be a set of symbols. Let $\mathcal{S}$ be a specification LTS over L, and let D, $F \subseteq L^\star$ be a pair of languages over L. Then, the set*

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

is the only complete test suite for $\mathcal{S}$ and (D, F).

Demonstração. Let $T \subseteq L^\star$ be any test suite over L, and let $\mathcal{I}$ be an arbitrary implementation LTS over L. From Definition 4.2, we know that $\mathcal{I}$ adheres to T if and only if $otr(\mathcal{I}) \cap T = \emptyset$. From Proposition 3.2 we get that $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ if and only if $otr(\mathcal{I}) \cap C = \emptyset$, where

$$C = \big[(D \cap \overline{otr}(\mathcal{S})) \cup (F \cap otr(\mathcal{S}))\big].$$

Hence, if $T = C$ we conclude that $\mathcal{I}$ adheres to T if and only if $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$. Since $\mathcal{I}$ was arbitrary, from Definition 4.3, we conclude that T is a complete test suite for $\mathcal{S}$ and (D, F).

Now, take another test suite $Z \subseteq L^\star$, with $Z \neq C$. For the sake of contradiction, assume that Z is also complete for $\mathcal{S}$ and (D, F). Since Z is complete, we can repeat the same reasoning as before and conclude that, for any implementation $\mathcal{I}$, we must have that $\mathcal{I}$ adheres to Z if and only if $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$, that is, $otr(\mathcal{I}) \cap Z = \emptyset$ if and only if $otr(\mathcal{I}) \cap C = \emptyset$, for any implementation $\mathcal{I}$. But $Z \neq C$ gives some $\sigma \in L^\star$ such that $\sigma \in C$ and $\sigma \notin Z$. The case $\sigma \notin C$ and $\sigma \in Z$ is entirely analogous. We now have $\sigma \in C \cap \overline{Z}$. If we can construct an implementation $\mathcal{I}$ with $\sigma \in otr(\mathcal{I})$ then we have reached a contradiction, because we would have $\sigma \in otr(\mathcal{I}) \cap C$ and $\sigma \notin otr(\mathcal{I}) \cap Z$. But that is simple. Let $\sigma = x_1 x_2 \ldots x_k$, with $k \geq 0$ and $x_i \in L$ $(1 \leq i \leq k)$. Define $\mathcal{I} = \langle Q, q_0, L_I, L_U, R\rangle$, where $Q = \{q_i \,|\, 0 \leq i \leq k\}$ and $R = \{(q_{i-1}, x_i, q_i) \,|\, 1 \leq i \leq k\}$. Clearly, $\sigma \in otr(\mathcal{I})$, concluding the proof. □

We note that, in Lemma 4.4, in order to construct $\mathcal{I}$ with $\sigma \in otr(\mathcal{I})$ it was crucial that we had no restrictions on the size of $\mathcal{I}$, once we have no control over the size of the witness string σ.

Hence, Lemma 4.4 says that the test suite $T = \big[(D \cap \overline{otr}(\mathcal{S})) \cup (F \cap otr(\mathcal{S}))\big]$ is complete for a specification $\mathcal{S}$ and the pair of languages (D, F). So, given an implementation $\mathcal{I}$, if one wants to check if it (D, F)-conforms to $\mathcal{S}$ it suffices to check if $\mathcal{I}$ adheres to T, that is, by Definition 4.2, it suffices to check that we have $otr(\mathcal{I}) \cap T = \emptyset$. Using Proposition 2.25 we see that $otr(\mathcal{S})$ is a regular language, and so, by Proposition 2.19 so is $\overline{otr}(\mathcal{S})$. Thus, if D and F are regular languages, then Proposition 2.19 again says that T is also a regular language. Moreover, it also says that, given FSAs $\mathcal{A}_D$ and $\mathcal{A}_F$ specifying D and F, respectively, we can effectively construct a FSA $\mathcal{A}_T$ whose semantics is the test suite T. Hence, using Propositions 2.25 and 2.19 again, we know that $otr(\mathcal{I}) \cap T$ is also a regular language, and that we can effectively construct a FSA $\mathcal{A}$ whose language is just $otr(\mathcal{I}) \cap T$. Then, a simple breadth-first traversal algorithm applied to $\mathcal{A}$ can then check if $otr(\mathcal{I}) \cap T = \emptyset$, so that we can effectively decide if $\mathcal{I}$ (D, F)-conforms to $\mathcal{S}$. Proposition 4.6 details the time complexity of such an algorithm.

A simple strategy in a test run should extract input strings from the FSA $\mathcal{A}_T$, i.e., strings (test cases) of the language T to be applied to an implementation. The length of such strings must be bounded by the number of states in the IUT. Hence we would get that the IUT **ioco** to the specification if all strings did not run to completion in IUT. Otherwise, the IUT **ioco** to the specification would not hold.

Example 4.5. Let the IOLTS $\mathcal{S}$ of Figure 6 be the specification model, with $L_I = \{a, b\}$, and $L_U = \{x\}$. The FSA $\overline{\mathcal{S}}$, depicted in Figure 7, is such that $L(\overline{\mathcal{S}}) = \overline{otr}(\mathcal{S})$. The FSA $\mathcal{D}$, shown in Figure 9, is such that $L(\mathcal{D}) = D$, where $D = otr(\mathcal{S})L_U$. The product $\mathcal{D} \times \overline{\mathcal{S}}$ is illustrated in Figure 10. The language accepted by $\mathcal{D} \times \overline{\mathcal{S}}$ is $TS = L(\mathcal{D} \times \overline{\mathcal{S}}) = D \cap \overline{otr}(\mathcal{S})$. According to the Lemma 4.4 and Corollary 3.10, TS represents the fault model which is **ioco**-complete for the specification $\mathcal{S}$.

Now let the IOLTS $\mathcal{I}$ of Figure 8 be an implementation. We see $bax \in TS$, *i.e.*, it is a test case of a complete fault model for the specification $\mathcal{S}$. Since $bax \in L(\mathcal{I})$ we then have that $\mathcal{I}$ **ioco** $\mathcal{S}$ does not hold. □

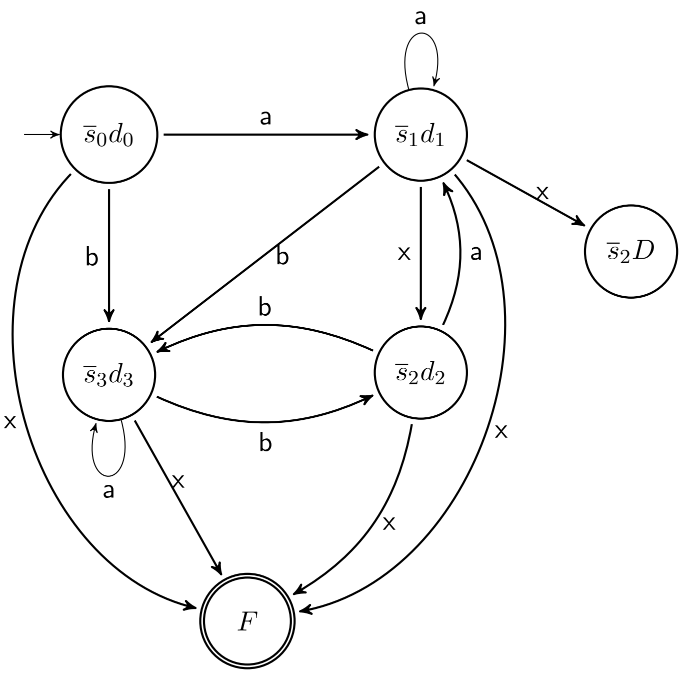

Figura 10: FSA $\mathcal{D} \times \overline{\mathcal{S}}$ for the language TS.

4.2 On the Complexity of Test Suites

Another important issue is the size of test suites, relatively to the size of the given specification. A reasonable measure of the size of a LTS $\mathcal{S} = \langle S, s_0, L, T\rangle$ would be the number of symbols required to write down a complete syntactic description of $\mathcal{S}$. Assume that $\mathcal{S}$ has t transitions, that is, $t = |T|$. Since each transition is labeled by an action symbol, then $|L| \leq t$ since unused symbols in L can be discarded. Also, by Remark 2.11, each state in a LTS is reachable from the initial state and so we always have $|S| \leq t - 1$. Thus, using an alphabet with $k \geq 2$ symbols, each transition (s, ℓ, p) can be written using $\mathcal{O}(\log_k(t))$ symbols[2]. So, a complete description of $\mathcal{S}$ would take $\mathcal{O}(t \log_k(t))$ symbols. Now, clearly, $t \leq n^2k$, where n is the number of states in $\mathcal{S}$. Hence, the size of $\mathcal{S}$ is $\mathcal{O}(n^2k\ln(nk))$. We will take the number of states of specification models as a first measure of their sizes.

Given languages D and F over a set of symbols L, and given a specification LTS $\mathcal{S} = \langle S, s_0, L, T\rangle$ over L, Lemma 4.4 says that the fault model T is complete for $\mathcal{S}$ and (D, F), where

$$T = \big[(D \cap \overline{otr}(\mathcal{S})) \cup (F \cap otr(\mathcal{S}))\big].$$

Assume that D and F are regular languages. Then, by Definitions 2.17 and 2.20, together with Propositions 2.21 and 2.22 we can effectively get complete FSAs $\mathcal{A}_D$ and $\mathcal{A}_F$ such that $D = L(\mathcal{A}_D)$ and $F = L(\mathcal{A}_F)$. Let n_D and n_F be the number of states in $\mathcal{A}_D$ and in $\mathcal{A}_F$, respectively. Assume also that $\mathcal{S}$ is deterministic and let n_S be the number of states in $\mathcal{S}$. Using Proposition 2.13, we know that the FSA $\mathcal{A}_1$ induced by $\mathcal{S}$, described in Definition 2.24, will also be deterministic with n_S states. Then, by Proposition 2.25 we can write $L(\mathcal{A}_1) = otr(\mathcal{S})$. Using Proposition 2.22 again we can effectively get a complete FSA $\mathcal{A}_2$ with $n_S + 1$ states, and such that $L(\mathcal{A}_2) = L(\mathcal{A}_1) = otr(\mathcal{S})$.

[2]We will use the standard big-oh, big-theta, and big-omega notations, written $\mathcal{O}(\cdot)$, $\Theta(\cdot)$ and $\Omega(\cdot)$, respectively, to indicate an upper bound, an equal rate, and a lower bound, respectively, on the asymptotic growth of two functions.

Hence, from Proposition 2.23, we can effectively get a complete FSA $\mathcal{A}_3$ with at most $(n_S+1)n_F$ states and such that $L(\mathcal{A}_3) = L(\mathcal{A}_F) \cap L(\mathcal{A}_2) = F \cap otr(\mathcal{S})$.

Consider the complete FSA $\mathcal{B}_2$ obtained from $\mathcal{A}_2$ by reversing its set of final states, that is, a state s is a final state in $\mathcal{B}_2$ if and only if s is not a final state in $\mathcal{A}_2$. Clearly, $L(\mathcal{B}_2) = \overline{L(\mathcal{A}_2)} = \overline{otr}(\mathcal{S})$. With Proposition 2.22 again we can now effectively get a complete FSA $\mathcal{B}_3$ with $(n_S+1)n_D$ states and such that $L(\mathcal{B}_3) = L(\mathcal{A}_D) \cap L(\mathcal{B}_2) = D \cap \overline{otr}(\mathcal{S})$.

Since $\mathcal{A}_3$ and $\mathcal{B}_3$ are complete FSAs, one more use of Proposition 2.22 effectively delivers a FSA $\mathcal{C}$ with $(n_S+1)^2 n_D n_F$ states and such that $L(\mathcal{C}) = L(\mathcal{A}_3) \cup L(\mathcal{B}_3)$. But then, $L(\mathcal{C}) = T$.

We conclude that when D and F are regular languages and $\mathcal{S}$ is a deterministic specification, then we can effectively construct a complete FSA $\mathcal{T}$ with at most $(n_S+1)^2 n_D n_F$ states such that $L(\mathcal{T}) = TS$, where n_D and n_F are the number of states in deterministic FSAs whose languages are D and F, respectively, and where n_S is the number of states in $\mathcal{S}$.

Proposition 4.6. *Let $\mathcal{S}$ and $\mathcal{I}$ be deterministic specification and implementation IOLTSs over L with n_S and n_I states, respectively. Let also $|L| = n_L$. Let $\mathcal{A}_D$ and $\mathcal{A}_F$ be deterministic FSAs over L with n_D and n_F states, respectively, and such that $L(\mathcal{A}_D) = D$ and $L(\mathcal{A}_F) = F$. Then, we can effectively construct a complete FSA $\mathcal{T}$ with $(n_S+1)^2 n_D n_F$ states, and such that $L(\mathcal{T})$ is a complete test suite for $\mathcal{S}$ and (D, F). Moreover, there is an algorithm, with polynomial time complexity $\Theta(n_S^2 n_I n_D n_F n_L)$ that effectively checks whether $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ holds.*

Demonstração. The construction of the complete FSA $\mathcal{T} = \langle S_{\mathcal{T}}, t_0, L, \rho_{\mathcal{T}}, F_{\mathcal{T}}\rangle$ with at most $(n_S+1)^2(n_D n_F)$ states follows from the preceding discussion, leading to

$$L(\mathcal{T}) = T = \big[(D \cap \overline{otr}(\mathcal{S})) \cup (F \cap otr(\mathcal{S}))\big].$$

By Lemma 4.4, T is a complete test suite for $\mathcal{S}$ and (D, F). By Definitions 4.2 and 4.3 we have $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ if and only if $otr(\mathcal{I}) \cap T = \emptyset$.

Since $\mathcal{I}$ is deterministic, we can repeat the argument at the preceding discussion and effectively construct a complete FSA $\mathcal{A} = \langle S_A, a_0, L, \rho_A, F_A\rangle$ with $n_I + 1$, and such that $otr(\mathcal{I}) = L(\mathcal{A})$. Using a simple breadth-first algorithm we can now start at the pair (a_0, t_0) and, in polynomial time $\Theta(n_S^2 n_I n_D n_F n_L)$, visit all states (a, t) such that $(a_0, t_0) \overset{\sigma}{\mapsto} (a, t)$, for some $\sigma \in L^\star$. Here, of course, for all $\ell \in L$ we have $(a_1, t_1) \overset{\ell}{\mapsto} (a_2, t_2)$ if and only if both $a_1 \overset{\ell}{\mapsto} a_2$ in $\mathcal{A}$ and $t_1 \overset{\ell}{\mapsto} t_2$ in $\mathcal{T}$. Then, we will reach a pair (a, t) where $a \in F_{\mathcal{A}}$ and $t \in F_{\mathcal{T}}$ if and only if $\emptyset \neq L(\mathcal{A}) \cap L(\mathcal{T}) = otr(\mathcal{I}) \cap T$, that is, if and only if $\mathcal{I}$ does not (D, F)-conform to $\mathcal{S}$, and the process is interrupted. Otherwise, if after visiting all such (a, t) pairs this condition is not met, we announce that $\mathcal{I}$ does (D, F)-conform to $\mathcal{S}$. $\square$

We can use Lemma 3.7 to get a similar result for the **ioco** conformance relation.

Theorem 4.7. *Let $\mathcal{S}$ and $\mathcal{I}$ be deterministic specification and implementation IOLTSs over L with n_S and n_I states, respectively. Let $L = L_I \cup L_U$, and $|L| = n_L$. Then, we can effectively construct an algorithm with polynomial time complexity $\Theta(n_S n_I n_L)$ that checks whether $\mathcal{I}$ **ioco** $\mathcal{S}$ holds.*

Demonstração. Let $\mathcal{A} = \langle S_A, s_0, L, \rho_A, S_A\rangle$ be the FSA induced by $\mathcal{S}$, according to Definition 2.24. Moreover, since $\mathcal{S}$ is deterministic, $\mathcal{A}$ is also deterministic. From Proposition 2.25 we get $otr(\mathcal{S}) = L(\mathcal{A})$. Clearly, $\mathcal{A}$ has n_S states.

Let $e, f \notin S_A$, with $e \neq f$, and define $S_T = S_A \cup \{e, f\}$. Also, extend the transition relation ρ_A as follows. Start with $\rho_T = \rho_A$. Next: (i) For any $\ell \in L_U$ and any $s \in S_A$ such that (s, ℓ, p) is not in ρ_A for any $p \in S_A$, add (s, ℓ, f) to ρ_T; (ii) for any $\ell \in L_I$ and any $s \in S_A$ such that (s, ℓ, p) is not in ρ_A for any $p \in S_A$, add (s, ℓ, e) to ρ_T. Now define the FSA $\mathcal{T} = \langle S_T, s_0, L, \rho_T, \{f\}\rangle$. Since $\mathcal{A}$

is deterministic, the construction implies that $\mathcal{T}$ is a deterministic FSA with $n_S + 2$ states. We see that f is the only final state in $\mathcal{T}$, and that f and e are sink states in $\mathcal{T}$.

Let $\sigma \in L^\star$. By the construction, we get $s_0 \overset{\sigma}{\mapsto} f$ in $\mathcal{T}$ if and only if $s_0 \overset{\alpha}{\mapsto} p \overset{\ell}{\mapsto} f$ in $\mathcal{T}$ for some $p \in S_A$, where $\ell \in L_U$ and $\sigma = \alpha\ell$.

We claim that, for any $\ell \in L_U$, we have $s_0 \overset{\alpha}{\mapsto} p \overset{\ell}{\mapsto} f$ in $\mathcal{T}$ with $p \in S_A$ if and only if we have $\alpha \in L(\mathcal{A})$ and $\alpha\ell \notin L(\mathcal{A})$. Assume that $s_0 \overset{\alpha}{\mapsto} p \overset{\ell}{\mapsto} f$ in $\mathcal{T}$ with $\ell \in L_U$ and $p \in S_A$. Since $p \neq f$ and $p \neq e$, we immediately get $s_0 \overset{\alpha}{\mapsto} p$ in $\mathcal{A}$, that is, $\alpha \in L(\mathcal{A})$. For the sake of contradiction, suppose that $\alpha\ell \in L(\mathcal{A})$. Then, we would get $s_0 \overset{\alpha}{\mapsto} q \overset{\ell}{\mapsto} r$ in $\mathcal{A}$, for some q, $r \in S_A$. But $\mathcal{A}$ is deterministic and we already have $s_0 \overset{\alpha}{\mapsto} p$ in $\mathcal{A}$. So, we must have $q = p$. Now we have (p, ℓ, f) in ρ_T and (p, ℓ, r) in ρ_A with $\ell \in L_U$. This contradicts item (i) of the construction of ρ_T, and we conclude that $\alpha\ell \notin L(\mathcal{A})$. For the converse, assume that $\alpha \in L(\mathcal{A})$ and $\alpha\ell \notin L(\mathcal{A})$ with $\ell \in L_U$. We readily get $s_0 \overset{\alpha}{\mapsto} p$ in $\mathcal{A}$, which immediately implies $s_0 \overset{\alpha}{\mapsto} p$ in $\mathcal{T}$. If we had (p, ℓ, r) in ρ_A for some $r \in S_A$, then we would get $\alpha\ell \in L(\mathcal{A})$, which can not happen. Then, item (ii) of the construction of $\mathcal{T}$ implies that (p, ℓ, f) is in ρ_T, and so we now have $s_0 \overset{\alpha}{\mapsto} p \overset{\ell}{\mapsto} f$ in $\mathcal{T}$, thus establishing the claim.

Putting it together we conclude that $s_0 \overset{\sigma}{\mapsto} f$ in $\mathcal{T}$ if and only if $\sigma = \alpha\ell$, $\alpha \in L(\mathcal{A})$ and $\sigma \notin L(\mathcal{A})$, for some $\ell \in L_U$ and some $\alpha \in L^*$. This shows that $L(\mathcal{T}) = \overline{L(\mathcal{A})} \cap (L(\mathcal{A})L_U) = \overline{otr}(\mathcal{S}) \cap (otr(\mathcal{S})L_U)$.

Consider the deterministic implementation $\mathcal{I}$ with n_I states, and let $\mathcal{B}$ be the FSA induced by $\mathcal{I}$. As before, $\mathcal{B}$ is a deterministic FSA with n_I states and such that $otr(\mathcal{I}) = L(\mathcal{B})$. By Proposition 2.22, we get a complete FSA $\mathcal{C} = \langle S_C, c_0, L, \rho_C, S_C \rangle$ with $n_I + 1$ states and such that $otr(\mathcal{I}) = L(\mathcal{B}) = L(\mathcal{C})$, and a complete FSA $\mathcal{U} = \langle S_\mathcal{U}, s_0, L, \rho_\mathcal{U}, \{f\} \rangle$ with $n_\mathcal{S} + 3$ states, such that $L(\mathcal{U}) = L(\mathcal{T})$. Starting with the pair (s_0, c_0), a breadth-first traversal algorithm will determine if we can reach a pair (f, q), for some $q \in S_C$ in time $\Theta(n_S n_I n_L)$ in the worst case. But we can reach such a pair of states if and only if we have $s_0 \overset{\sigma}{\mapsto} f$ in $\mathcal{U}$ and $c_0 \overset{\sigma}{\mapsto} q$ in $\mathcal{C}$, that is, if and only if

$$\emptyset \neq L(\mathcal{U}) \cap L(\mathcal{C}) = otr(\mathcal{I}) \cap \Big[\overline{otr}(\mathcal{S}) \cap (otr(\mathcal{S}) \cdot L_U)\Big].$$

According to Corollary 3.10, we have that $\mathcal{I}$ **ioco** $\mathcal{S}$ if and only if $otr(\mathcal{I}) \cap \Big[\overline{otr}(\mathcal{S}) \cap (otr(\mathcal{S}) \cdot L_U)\Big] = \emptyset$. Thus, we just say that $\mathcal{I}$ **ioco** $\mathcal{S}$ if and only if the breadth-first search algorithm never reaches a state pair (f, q), for any $q \in S_C$. Otherwise we say that $\mathcal{I}$ **ioco** $\mathcal{S}$ does not hold. $\Box$

Assume that the set of symbols $L_I \cup L_U$ is fixed. Usually, one has access to the internal structure of the specification, that is, an IOLTS model $\mathcal{S}$ is given for the specification. This allows us to construct the FSA $\mathcal{U}$ of Theorem 4.7 in polynomial time $\Theta(n)$, where n is the number of states in the specification. If we also know the internal structure of the implementation to be put under test, that is if we are in a "white box" testing scenario, then we also have an IOLTS model $\mathcal{I}$ for the implementation. In this case, Theorem 4.7 says that we can test the **ioco** -conformance relation between $\mathcal{S}$ and $\mathcal{I}$ in polynomial time $\mathcal{O}(nm)$, for *any* implementations with $\mathcal{O}(m)$ states.

5 Using Test Purposes When Testing IOLTSs

In this section we want to apply some of the results of previous sections to the testing architecture described by Tretmans[3] [27]. In a setting where the implementation under test is a "black box", that is, we do not have access to its internal structure through a specific IOLTS model, the tester "drives" the test run, that is, it behaves as an artificial environment to the IUT, and both engage in a cycle of exchanges: at some point, the tester issues one of its output symbols — an input

[3]Here we treat only finite models.

symbol for the IUT —; the IUT receives this input symbol and both models change states based on this symbol; at some other point, by its turn, the IUT sends back to the tester one of its output symbols — an input symbol for the tester —; the tester receives this input symbol and both models change state again; now the tester is ready do repeat the cycle. Of course, at any instant both the tester and the implementation can make any number of internal τ-moves. In any case, it would be reasonable to require that testers and implementations do not refuse symbols exchanged between them. This motivates the following definition. Recall Definition 2.31.

Definition 5.1. *An IOLTS $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle$ is* input-enabled *if* $\mathbf{inp}(s) = L_I$*, for all $s \in S$. The class of all input-enabled IOLTSs over the alphabets L_I and L_U will be designated by $\mathcal{IOE}(I, U)$.*

Later on, in this section, we will focus on input-enabled models. When appropriate, however, we will be considering the full class of IOLTSs models, in order to state more general definitions and get more extensive results.

5.1 A Particular Class of Fault Models

One can specify a fault model by a finite set TP of IOLTSs [27], referred to here as *test purposes.* When an implementation IOLTS $\mathcal{I}$ is put under test, each of these test purposes play the role of a "tester" that drives the testing process, as also discussed in Subsection 2.5. In this fault model, a test purpose $\mathcal{T}$ has special states, marked as **pass** and **fail** states. Behaviors that reach a **fail** state in $\mathcal{T}$ will then be considered harmful, and behaviors that reach a **pass** state will be considered acceptable. To keep those behaviors separate, some mechanism that guarantees that there are no paths in $\mathcal{T}$ from a **pass** state to a **fail** state, and vice-versa, must be in place. In particular, one can require that any transition $(\textbf{pass}, x, q) \in R$ implies $q = \textbf{pass}$. Likewise, $(\textbf{fail}, x, q) \in R$ implies $q = \textbf{fail}$. Next, we frame this notions precisely.

Definition 5.2. *Let $L = L_I \cup L_U$ be a set of symbols. A* test purpose *over L is any IOLTS $\mathcal{T} \in \mathcal{IO}(U, I)$. If* **fail** *and* **pass** *are distinct states of $\mathcal{T}$, then for no $\sigma \in L^*$ we have either* $\textbf{fail} \overset{\sigma}{\Rightarrow} \textbf{pass}$ *or* $\textbf{pass} \overset{\sigma}{\Rightarrow} \textbf{fail}$*. A* fault model *over L is any finite collection of test purposes over L.*

Note that L_U is the set of input symbols in a test case, and L_I is its set of output symbols, in accordance with the scenario where the test case emit symbols that are taken by the implementations, and vice-versa. We also note that, in [27], a number of other restrictions are imposed on $\mathcal{T}$, which we will discuss later on, but are immaterial to us at this point. Also note that, as defined, the semantics represented by all the observable behaviors of $\mathcal{T}$ that reach a **fail** state is a regular language. Likewise for all observable behaviors that reach a **pass** state.

Given a test purpose $\mathcal{T} \in \mathcal{IO}(U, I)$ and an implementation $\mathcal{I} \in \mathcal{IO}(I, U)$, in order to run a test based on $\mathcal{T}$ over $\mathcal{I}$, a cross-product LTS $\mathcal{T} \times \mathcal{I}$ is constructed. Here, all symbols are treated equally as elements of the set $L = L_U \cup L_I$.

Definition 5.3. *Let $\mathcal{T} = \langle Q, q_0, L_U, L_I, R\rangle \in \mathcal{IO}(U, I)$ and $\mathcal{I} = \langle S, s_0, L_I, L_U, T\rangle \in \mathcal{IO}(I, U)$ be IOLTSs. We define their cross-product as the LTS $\mathcal{T} \times \mathcal{I} = \langle Q \times S, (q_0, s_0), L, P\rangle$, where $L = L_I \cup L_U$ and $((q_1, s_1), x, (q_2, s_2))$ is a transition in P if and only if either*

- *$x = \tau$, $s_1 = s_2$ and (q_1, τ, q_2) is a transition in R, or*
- *$x = \tau$, $q_1 = q_2$ and (s_1, τ, s_2) is a transition in T, or*
- *$x \neq \tau$, (q_1, x, q_2) is a transition in R, and (s_1, x, s_2) is a transition in T.* □

We can now show that the collective behavior described in the cross-product always implies the same behavior about the two participating IOLTSs, and conversely.

Proposition 5.4. *Let $\mathcal{T} = \langle Q, q_0, L_U, L_I, R\rangle \in \mathcal{IO}(U, I)$ and $\mathcal{I} = \langle S, s_0, L_I, L_U, T\rangle \in \mathcal{IO}(I, U)$ be IOLTS, and let $\mathcal{T} \times \mathcal{I}$ be their cross-product. Then, we have*

1. *$(t, q) \xrightarrow{\tau^k} (p, r)$ in $\mathcal{T} \times \mathcal{I}$, with $k \geq 0$, if and only if there are n, $m \geq 0$, with $n + m = k$, and such that $t \xrightarrow{\tau^n} p$ in $\mathcal{T}$ and $q \xrightarrow{\tau^m} r$ in $\mathcal{I}$.*

2. *$(t, q) \overset{\sigma}{\Rightarrow} (p, r)$ in $\mathcal{T} \times \mathcal{I}$ if and only if $t \overset{\sigma}{\Rightarrow} p$ in $\mathcal{T}$ and $q \overset{\sigma}{\Rightarrow} r$ in $\mathcal{I}$, for all $\sigma \in L^\star$.*

Demonstração. The first item follows by an easy induction on $k \geq 0$.

For the second item, first assume that $(t, q) \overset{\sigma}{\Rightarrow} (p, r)$ in $\mathcal{T} \times \mathcal{I}$. According to Definition 2.5 we get $(t, q) \xrightarrow{\mu} (p, r)$ with $h_\tau(\mu) = \sigma$. We proceed by induction on $|\mu| \geq 0$. When $|\mu| = 0$, we get $\sigma = \varepsilon$, $t = p$, $q = r$ and the result follows easily. Next, assume $(t, q) \xrightarrow{\mu} (t_1, q_1) \xrightarrow{x} (p, r)$ with $|x| = 1$ and $h_\tau(\mu x) = \sigma$. The induction hypothesis gives $t \overset{\rho}{\Rightarrow} t_1$ in $\mathcal{T}$ and $q \overset{\rho}{\Rightarrow} q_1$ in $\mathcal{I}$ with $h_\tau(\mu) = \rho$. If $x = \tau$, Definition 5.3 gives $t_1 \xrightarrow{\tau} p$ and $q_1 = r$, or $q_1 \xrightarrow{\tau} r$ and $t_1 = p$. We assume the first case, the reasoning for the other case being entirely analogous. We now have $\sigma = h_\tau(\mu x) = h_\tau(\mu\tau) = h_\tau(\mu) = \rho$. We have $q_1 = r$, and $q \overset{\rho}{\Rightarrow} q_1$ in $\mathcal{I}$, so that now we can write $q \overset{\sigma}{\Rightarrow} r$ in $\mathcal{I}$. From $t \overset{\rho}{\Rightarrow} t_1$ we get $t \xrightarrow{\eta} t_1$ with $h_\tau(\eta) = \rho$. Hence, $t \xrightarrow{\eta} t_1 \xrightarrow{\tau} p$, and since $h_\tau(\eta\tau) = h_\tau(\eta) = \rho = \sigma$ we also have $t \overset{\sigma}{\Rightarrow} p$ in $\mathcal{S}$, as desired. Next, we have the case when $x \neq \tau$. Then, $\sigma = h_\tau(\mu x) = h_\tau(\mu)h_\tau(x) = \rho x$. Now, Definition 5.3 gives $t_1 \xrightarrow{x} p$ and $q_1 \xrightarrow{x} r$. From $t \overset{\rho}{\Rightarrow} t_1$ and $q \overset{\rho}{\Rightarrow} q_1$ we obtain $t \xrightarrow{\eta_1} t_1$ and $q \xrightarrow{\eta_2} q_1$ with $h_\tau(\eta_1) = \rho = h_\tau(\eta_2)$. Hence, $t \xrightarrow{\eta_1 x} p$ and $q \xrightarrow{\eta_2 x} r$. Since $h_\tau(\eta_1 x) = h_\tau(\eta_1)h_\tau(x) = \rho x = \sigma = h_\tau(\eta_2)h_\tau(x) = h_\tau(\eta_2 x)$, we can write $t \overset{\sigma}{\Rightarrow} p$ in $\mathcal{T}$ and $q \overset{\sigma}{\Rightarrow} r$ in $\mathcal{I}$, as desired.

For the converse of item 2, we note that from $t \overset{\sigma}{\Rightarrow} p$ in $\mathcal{T}$ and $q \overset{\sigma}{\Rightarrow} r$ in $\mathcal{I}$, we get $t \xrightarrow{\mu_1} p$ and $q \xrightarrow{\mu_2} r$, with $h_\tau(\mu_1) = \sigma = h_\tau(\mu_2)$. We induct on $|\mu_1| + |\mu_2| \geq 0$. If $\mu_1 = \tau^n$, for some $n \geq 0$, we obtain $\sigma = h_\tau(\tau^n) = \varepsilon = h_\tau(\mu_2)$, and we must have $\mu_2 = \tau^m$, for some $m \geq 0$. Using item 1, we obtain $(t, q) \xrightarrow{\tau^{m+n}} (p, r)$, and since $h_\tau(\tau^{m+n}) = \varepsilon = \sigma$ we arrive at $(t, q) \overset{\sigma}{\Rightarrow} (p, r)$, as desired. Now, assume $\mu_1 = \alpha_1 x\beta_1$ with $x \neq \tau$ and $\beta_1 = \tau^n$, for some $n \geq 0$. Then, $\sigma = h_\tau(\mu_1) = h_\tau(\alpha_1)x = h_\tau(\mu_2)$, and we conclude that $\mu_2 = \alpha_2 x\beta_2$ with $h_\tau(\alpha_2) = h_\tau(\alpha_1)$ and $\beta_2 = \tau^m$, for some $m \geq 0$. So, we now have $t \xrightarrow{\alpha_1} t_1 \xrightarrow{x} t_2 \xrightarrow{\beta_1} p$ and $q \xrightarrow{\alpha_2} q_1 \xrightarrow{x} q_2 \xrightarrow{\beta_2} r$. Let $\rho = h_\tau(\alpha_1) = h_\tau(\alpha_2)$, so that $\sigma = \rho x$. The induction hypothesis gives $(t, q) \overset{\rho}{\Rightarrow} (t_1, q_1)$, which means that $(t, q) \xrightarrow{\eta} (t_1, q_1)$ and $h_\tau(\eta) = \rho$. From Definition 5.3 again we get $(t_1, q_1) \xrightarrow{x} (t_2, q_2)$. From the proof of item 1 we get $(t_2, q_2) \xrightarrow{\tau^{m+n}} (p, r)$. Collecting, we have $(t, q) \xrightarrow{\eta} (t_1, q_1) \xrightarrow{x} (t_2, q_2) \xrightarrow{\tau^{n+m}} (p, r)$. Since $h_\tau(\eta x\tau^{n+m}) = h_\tau(\eta)x = \rho x = \sigma$, we can now write $(t, q) \overset{\sigma}{\Rightarrow} (p, r)$, completing the proof. □

In this scenario, we want to say when an IUT $\mathcal{I}$ passes a test purpose $\mathcal{T}$, and when a fault model is **ioco**-complete for a given specification relatively to a class of implementations [27]. Recall Definition 3.6.

Definition 5.5. *Let $\mathcal{I} = \langle S_\mathcal{I}, q_0, L_I, L_U, T_\mathcal{I}\rangle \in \mathcal{IO}(I, U)$ be an IUT and let $\mathcal{T} = \langle S_\mathcal{T}, t_0, L_U, L_I, T_\mathcal{T}\rangle \in \mathcal{IO}(U, I)$ be a test purpose. We say that $\mathcal{I}$* passes *$\mathcal{T}$ if, for any $\sigma \in (L_I \cup L_U)^*$ and any state $q \in S_\mathcal{I}$, we do not have $(t_0, q_0) \overset{\sigma}{\Rightarrow} (\textsf{fail}, q)$ in $\mathcal{T} \times \mathcal{I}$. Let TP be a fault model. We say that $\mathcal{I}$* passes *TP if $\mathcal{I}$ passes all test purposes in TP. Let $\mathcal{S}$ be an IOLTS, and let $\mathcal{IMP} \subseteq \mathcal{IO}(I, U)$ be a family of IOLTSs. We say that TP is* **ioco**-complete *for $\mathcal{S}$ relatively to $\mathcal{IMP}$ if, for all $\mathcal{Q} \in \mathcal{IMP}$, we have $\mathcal{Q}$* **ioco** *$\mathcal{S}$ if and only if $\mathcal{Q}$ passes TP.*

When $\mathcal{IMP}$ is the full class $\mathcal{IO}(I,U)$, we can simply say that TP is **ioco**-complete for $\mathcal{S}$, instead of TP is **ioco**-complete for $\mathcal{S}$ relatively to the class $\mathcal{IO}(I,U)$.

Remark 5.6. *If $\mathcal{IMP}_1 \subseteq \mathcal{IMP}_2 \subseteq \mathcal{IO}(I,U)$ are two families of IOLTSs, $\mathcal{S} \in \mathcal{IO}(I,U)$ and TP is a fault model over $L_I \cup L_U$, then it is clear that TP being* **ioco***-complete for $\mathcal{S}$ relatively to $\mathcal{IMP}_2$ implies that TP is* **ioco***-complete for $\mathcal{S}$ relatively to $\mathcal{IMP}_1$.*

Given a specification $\mathcal{S}$, we can always construct a fault model which is **ioco**-complete for $\mathcal{S}$, and which contains only deterministic and input-enabled test purposes.

Lemma 5.7. *Let $\mathcal{S} = \langle S_\mathcal{S}, s_0, L_I, L_U, R_\mathcal{S}\rangle \in \mathcal{IO}(I,U)$ be a specification. Then we can effectively construct a fault model $TP = \{\mathcal{T}\}$ which is* **ioco***-complete for $\mathcal{S}$. Moreover, $\mathcal{T}$ is a deterministic input-enabled IOLTS with a single* **fail** *state and no* **pass** *state.*

Demonstração. We will devise a fault model TP that is a singleton, that is, $TP = \{\mathcal{T}\}$ where $\mathcal{T} = \langle S_\mathcal{T}, t_0, L_U, L_I, R_\mathcal{T}\rangle$ is some test purpose to be constructed. In order for TP to be **ioco**-complete for $\mathcal{S}$ we need that, for all implementations $\mathcal{I}$, it holds that $\mathcal{I}$ passes $\mathcal{T}$ if and only if $\mathcal{I}$ **ioco** $\mathcal{S}$. From Lemma 3.7 we know that $\mathcal{I}$ **ioco** $\mathcal{S}$ if and only if $\mathcal{I}\,\mathbf{conf}_{D,\emptyset}\,\mathcal{S}$, with $D = otr(\mathcal{S})L_U$. Recall that $\overline{otr}(\mathcal{S})$ indicates the complement of $otr(\mathcal{S})$, that is, $\overline{otr}(\mathcal{S}) = L^\star - otr(\mathcal{S})$. From Proposition 3.2 we get $\mathcal{I}\,\mathbf{conf}_{D,\emptyset}\,\mathcal{S}$ if and only if $otr(\mathcal{I}) \cap T = \emptyset$, where $T = \overline{otr}(\mathcal{S}) \cap (otr(\mathcal{S})L_U)$. Putting it together, we see that we need a test purpose $\mathcal{T}$ such that, for all implementations $\mathcal{I}$, we have $\mathcal{I}$ passes $\mathcal{T}$ if and only if $otr(\mathcal{I}) \cap T = \emptyset$.

Let $\mathcal{I} = \langle S_\mathcal{I}, q_0, L_I, L_U, R_\mathcal{I}\rangle \in \mathcal{IO}(I,U)$ be an arbitrary implementation. We have that $\mathcal{I}$ passes $\mathcal{T}$ if and only if in the cross-product LTS $\mathcal{T} \times \mathcal{I}$ we never have $(s_0,q_0) \overset{\sigma}{\Rightarrow} (\mathbf{fail}, q)$, for any $\sigma \in L^\star$ and any $q \in S_\mathcal{I}$. Since we want $\mathcal{I}$ passes $\mathcal{T}$ if and only if $otr(\mathcal{I}) \cap T = \emptyset$, we conclude that we need $otr(\mathcal{I}) \cap T \neq \emptyset$ if and only if $(t_0,q_0) \overset{\sigma}{\Rightarrow} (\mathbf{fail}, q)$ in $\mathcal{T} \times \mathcal{I}$ for some $\sigma \in L^\star$ and some $q \in S_\mathcal{I}$.

We start by using Proposition 2.26 and the underlying LTS of $\mathcal{S}$ to effectively construct a deterministic LTS $\mathcal{B}'$ such that $otr(\mathcal{S}) = tr(\mathcal{B}') = otr(\mathcal{B}')$. Let $\mathcal{B}$ be the IOLTS whose underlying LTS is $\mathcal{B}'$. Clearly, $\mathcal{B}$ can be effectively constructed, is deterministic, and $otr(\mathcal{S}) = tr(\mathcal{B}) = otr(\mathcal{B})$. Let $\mathcal{B} = \langle S_\mathcal{B}, b_0, L_I, L_U, R_\mathcal{B}\rangle$. We use $\mathcal{B}$ to construct the desired test purpose $\mathcal{T} = \langle S_\mathcal{T}, t_0, L_U, L_I, R_\mathcal{T}\rangle$ extending the state set $S_\mathcal{B}$ and the transition set $R_\mathcal{B}$ as follows: we add a **fail** state to $S_\mathcal{B}$ and add a transition (s,ℓ,p) to $R_\mathcal{B}$ whenever state s has no outgoing transition labeled ℓ in $\mathcal{B}$, where $\ell \in L_U$ is an output symbol. More precisely, we construct $\mathcal{T}$ by defining $t_0 = b_0$, $S_\mathcal{T} = S_\mathcal{B} \cup \{\mathbf{fail}\}$ where $\mathbf{fail} \notin S_\mathcal{B}$, and letting

$$R_\mathcal{T} = R_\mathcal{B} \cup \big\{(s,\ell,\mathbf{fail}) \,|\, \ell \in L_U \text{ and } (s,\ell,p) \notin R_\mathcal{B} \text{ for any } p \in S_\mathcal{B}\big\} \cup \big\{(\mathbf{fail},\ell,\mathbf{fail}) \,|\, \ell \in L_U\big\}.$$

According to Definition 5.1, $\mathcal{T}$ is input-enabled by construction. Also, clearly, $\mathcal{T}$ is a deterministic IOLTS with a single **fail** state and no **pass** states. In order to complete the proof we have to establish that $otr(\mathcal{I}) \cap T \neq \emptyset$ if and only if $(t_0,q_0) \overset{\sigma}{\Rightarrow} (\mathbf{fail}, q)$ in $\mathcal{T} \times \mathcal{I}$ for some $\sigma \in L^\star$ and some $q \in S_\mathcal{I}$. We start with the following claim.

Claim:

(i) If $t \neq \mathbf{fail}$, then $t_0 \overset{\sigma}{\Rightarrow} t$ in $\mathcal{T}$ if and only if $b_0 \overset{\sigma}{\Rightarrow} b$ in $\mathcal{B}$.

(ii) $t_0 \overset{\sigma}{\Rightarrow} \mathbf{fail}$ in $\mathcal{T}$ if and only if $\sigma = \alpha\ell$, with $\alpha \in L^\star$, $\ell \in L_U$, $b_0 \overset{\alpha}{\Rightarrow} b$ in $\mathcal{B}$, and $b \overset{\ell}{\to}$ not in $\mathcal{B}$, for some $b \in S_\mathcal{B}$, with $b \neq \mathbf{fail}$.

Proof of the Claim: Since $\mathcal{T}$ is deterministic, it has no τ-labeled transitions. Hence, $t_0 \overset{\sigma}{\Rightarrow} t$ if and only if $t_0 \overset{\sigma}{\to} t$. An easy induction on $|\sigma| \geq 0$, with $t \neq \mathbf{fail}$, gives that $t_0 \overset{\sigma}{\to} t$ in $\mathcal{T}$ if and only if $b_0 \overset{\sigma}{\rightarrow} b$ in $\mathcal{B}$. Since $\mathcal{B}$ is also deterministic, we immediately get $b_0 \overset{\sigma}{\rightarrow} b$ if and only if $b_0 \overset{\sigma}{\Rightarrow} b$ in $\mathcal{B}$, thus establishing item (i).

Since $\mathcal{T}$ has no τ-labeled transitions, we know that $t_0 \overset{\sigma}{\Rightarrow}$ **fail** if and only if $t_0 \overset{\sigma}{\rightarrow}$ **fail**. By the construction of $\mathcal{T}$ from $\mathcal{B}$ we know that it has no transitions starting at the **fail** state. Hence, $t_0 \overset{\sigma}{\rightarrow}$ **fail** holds if and only if $t_0 \overset{\alpha}{\rightarrow} t_1 \overset{\ell}{\rightarrow}$ **fail**, where $t_1 \neq$ **fail**, $\sigma = \alpha\ell$, $\alpha \in L^\star$, and $\ell \in L_U$. Again, $t_0 \overset{\alpha}{\rightarrow} t_1$, with $t_1 \neq$ **fail**, holds in $\mathcal{T}$ if and only if we have $b_0 \overset{\alpha}{\rightarrow} b_1$ in $\mathcal{B}$. Since $\mathcal{B}$ is also deterministic, we readily get $b_0 \overset{\alpha}{\rightarrow} b_1$ in $\mathcal{B}$ if and only if $b_0 \overset{\alpha}{\Rightarrow} b_1$ in $\mathcal{B}$. Moreover, by construction of $\mathcal{T}$, $t_1 \overset{\ell}{\rightarrow}$ **fail** if and only if $b_1 \overset{\ell}{\rightarrow}$ does not hold in $\mathcal{B}$. This verifies item (ii), and completes the proof of the claim.

Assume that $(s_0, q_0) \overset{\sigma}{\Rightarrow} (\textbf{fail}, q)$ in $\mathcal{T} \times \mathcal{I}$ for some $\sigma \in L^\star$ and some $q \in S_\mathcal{I}$. Using Proposition 5.4 we obtain $t_0 \overset{\sigma}{\Rightarrow}$ **fail** in $\mathcal{T}$ and $q_0 \overset{\sigma}{\Rightarrow} q$ in $\mathcal{I}$. Hence, $\sigma \in otr(\mathcal{I})$. The Claim gives $\sigma = \alpha\ell$, with $\alpha \in L^\star$, $\ell \in L_U$, $b_0 \overset{\alpha}{\Rightarrow} b$ in $\mathcal{B}$, and $b \overset{\ell}{\rightarrow}$ not in $\mathcal{B}$, for some $b \in S_\mathcal{B}$ with $b \neq$ **fail**. Hence, $\alpha \in otr(\mathcal{B})$ and, since we already have $otr(\mathcal{S}) = otr(\mathcal{B})$, we get $\alpha \in otr(\mathcal{S})$. Because $\ell \in L_U$ and $\sigma = \alpha\ell$, we conclude that $\sigma \in otr(\mathcal{S})L_U$. This gives $\sigma \in otr(\mathcal{I}) \cap (otr(\mathcal{S})L_U)$. Thus we must have $\sigma \notin otr(\mathcal{S})$, otherwise we would also have $\sigma \in otr(\mathcal{B})$, since $otr(\mathcal{S}) = otr(\mathcal{B})$. This would give $b_0 \overset{\alpha}{\Rightarrow} r \overset{\ell}{\Rightarrow} q$ in $\mathcal{B}$, for some r, $q \in S_\mathcal{B}$. Since $\mathcal{B}$ is deterministic and we already have $b_0 \overset{\alpha}{\Rightarrow} b$ in $\mathcal{B}$, we conclude that $r = b$. Hence, $r \overset{\ell}{\Rightarrow} q$ implies $b \overset{\ell}{\rightarrow}$ in $\mathcal{B}$, which is a contradiction. So, $\sigma \notin otr(\mathcal{S})$, that is, $\sigma \in \overline{otr}(\mathcal{S})$. We can now write $\sigma \in otr(\mathcal{I}) \cap \left[\overline{otr}(\mathcal{S}) \cap (otr(\mathcal{S})L_U)\right]$, that is, $\sigma \in otr(\mathcal{I}) \cap T$, showing that $otr(\mathcal{I}) \cap T \neq \emptyset$, as desired.

For the other direction, assume that $otr(\mathcal{I}) \cap T \neq \emptyset$. We then get some $\sigma \in L^\star$ such that $\sigma \in otr(\mathcal{I})$, $\sigma \notin otr(\mathcal{S})$ and $\sigma \in otr(\mathcal{S})L_U$. Hence, $\sigma \in otr(\mathcal{I})$ gives $q_0 \overset{\sigma}{\Rightarrow} q$ in $\mathcal{I}$, for some $q \in S_\mathcal{I}$. Also $\sigma \in otr(\mathcal{S})L_U$ implies that $\sigma = \alpha\ell$ for some $\ell \in L_U$ and $\alpha \in otr(\mathcal{S})$. Then, $\alpha \in otr(\mathcal{B})$, so that $b_0 \overset{\alpha}{\Rightarrow} b$ in $\mathcal{B}$ for some $b \neq$ **fail**. If $b \overset{\ell}{\rightarrow}$ in $\mathcal{B}$, then $b_0 \overset{\alpha\ell}{\Rightarrow}$ in $\mathcal{B}$ because we already have $b_0 \overset{\alpha}{\Rightarrow} b$ in $\mathcal{B}$. This gives $\alpha\ell \in otr(\mathcal{B})$, and so $\sigma \in otr(\mathcal{S})$ because $\sigma = \alpha\ell$ and $otr(\mathcal{S}) = otr(\mathcal{B})$. But this is a contradiction since we already have $\sigma \notin otr(\mathcal{S})$. Hence, $b \overset{\ell}{\rightarrow}$ does not hold in $\mathcal{B}$. The Claim then gives $t_0 \overset{\sigma}{\Rightarrow}$ **fail** in $\mathcal{T}$. Using Proposition 5.4 we obtain $(t_0, q_0) \overset{\sigma}{\Rightarrow} (\textbf{fail}, q)$ with $\sigma \in L^\star$, establishing the reverse direction.

We have reached the desired conclusion, namely, $otr(\mathcal{I}) \cap T \neq \emptyset$ if and only if $(t_0, q_0) \overset{\sigma}{\Rightarrow} (\textbf{fail}, q)$ in $\mathcal{T} \times \mathcal{I}$ for some $\sigma \in L^\star$ and some $q \in S_\mathcal{I}$. The proof is, thus, complete. □

We illustrate the construction of a TP **ioco**-complete for a given specification as described at Lemma 5.7.

Example 5.8. Let the IOLTS $\mathcal{S}$ of Figure 6 be the specification model. Since $\mathcal{S}$ is already deterministic we start the construction of a test purpose $\mathcal{T}$ as described in Lemma 5.7. We add transitions $(t_0, x, fail)$, $(t_2, x, fail)$ and $(t_3, x, fail)$ to $R_\mathcal{T}$. The construction is illustrated at Figure 11. Notice that, among others, the observable behavior $\alpha = axbx$ leads to the **fail** state in $\mathcal{T}$, that is $t_0 \overset{\alpha}{\Rightarrow}$ **fail** in $\mathcal{T}$. Now let the IOLTS $\mathcal{I}$ of Figure 8 be an implementation. We now have $q_0 \overset{\alpha}{\Rightarrow} q_2$ in $\mathcal{I}$. Hence, we get $(t_0, q_0) \overset{\alpha}{\Rightarrow} (\textbf{fail}, q_2)$ in $\mathcal{T} \times \mathcal{I}$ and so $\mathcal{I}$ does not pass the fault model $\{\mathcal{T}\}$.

According to Lemma 5.7, $\{\mathcal{T}\}$ is a complete fault model for the specification $\mathcal{S}$. Therefore, $\mathcal{I}$ **ioco** $\mathcal{S}$ should not hold also. We check that this is indeed the case. Consider, for example, the observable behavior $\beta = ba$. We have $q_0 \overset{\beta}{\Rightarrow} q_3$ in $\mathcal{I}$, and there is a transition (q_3, x, q_2) in $\mathcal{I}$. According to Definition 3.6, we have $x \in out(q_0 \textbf{ after } \beta)$. On the other hand, we have $s_0 \overset{\beta}{\Rightarrow} s_3$ in $\mathcal{S}$, but we have no transition out of s_3 on the output label x in $\mathcal{S}$. Hence, $x \notin out(s_0 \textbf{ after } \beta)$, and we conclude that $out(q_0 \textbf{ after } \beta) \not\subseteq out(s_0 \textbf{ after } \beta)$. Since β is clearly an observable behavior of $\mathcal{S}$, Definition 3.6 says that $\mathcal{I}$ **ioco** $\mathcal{S}$ does not hold, as was expected. □

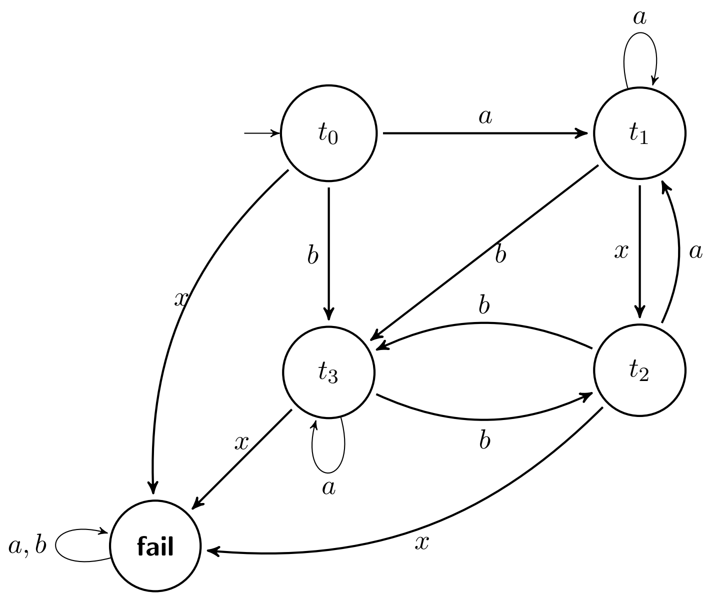

Figura 11: Test purpose $\mathcal{T}$ **ioco**-complete for $\mathcal{S}$ in Figure 6.

5.2 A Specific Family of Test Cases

A number of other specific conditions can be imposed on the structure of a tester. In this subsection, we examine conditions that will lead to the specific class of testers considered by Tretmans [27]. First we would like to remove cycles in all test purposes present in a given fault model. This will guarantee that in any test run only finite words are exchanged between the tester and the IUT. But this goal is not tenable, in general, unless some bound on the size of implementations is enforced as shown in next proposition.

Proposition 5.9. *There is a simple deterministic input-enabled specification* $\mathcal{S} \in \mathcal{IOE}(\{a\},\{x\})$ *for which there is no fault model* TP *which is* **ioco***-complete for* $\mathcal{S}$ *relatively to* $\mathcal{IOE}(\{a\},\{x\})$*, and such that all test purposes in* TP *are acyclic.*

Demonstração. Consider the specification IOLTS $\mathcal{S}$ with a simple self-loop described by the transition (s_0, a, s_0), where $L_I = \{a\}$ and $L_U = \{x\}$. Clearly, $\mathcal{S} \in \mathcal{IOE}(\{a\},\{x\})$. Let $\alpha = a^n$, with $n \geq 0$. Clearly, $\alpha x \in \overline{otr}(\mathcal{S}) \cap (otr(\mathcal{S})\ L_U)$. Consider now the implementation $\mathcal{I}_n$ described by the transitions (q_{i-1}, a, q_i) $(1 \leq i \leq n)$, and (q_n, x, q), together with the self-loops (q_n, a, q_n) and (q, a, q). Clearly, we also have $\mathcal{I}_n \in \mathcal{IOE}(\{a\},\{x\})$. Let q_0 be the initial state in $\mathcal{I}_n$. Easily, $\alpha x \in otr(\mathcal{I}_n)$. Hence, for all $n \geq 0$ we have an implementation $\mathcal{I}_n$ such that

$$otr(\mathcal{I}_n) \cap \big[\overline{otr}(\mathcal{S}) \cap otr(\mathcal{S})\ L_U\big] \neq \emptyset.$$

Now, assume we had a fault model TP that was **ioco**-complete for $\mathcal{S}$, and such that any test purpose $\mathcal{T} \in TP$ is acyclic. Because TP is finite and all test purposes in TP are acyclic, we can define

$$k = \max_{\mathcal{T} \in TP} \big\{|\sigma| \,|\, \sigma \in otr(\mathcal{T})\big\},$$

as the maximum length of a trace in any test purpose in TP. Now, consider the implementation $\mathcal{I}_k$. Since $otr(\mathcal{I}_k) \cap \big[\overline{otr}(\mathcal{S}) \cap otr(\mathcal{S})\ L_U\big] \neq \emptyset$, Proposition 3.2 and Lemma 3.7 say that $\mathcal{I}_k$ **ioco** $\mathcal{S}$ does not hold. Because TP is **ioco**-complete for $\mathcal{S}$ relatively to $\mathcal{IOE}(I, U)$ and $\mathcal{I}_k \in \mathcal{IOE}(I, U)$, we must then have that $\mathcal{I}_k$ does not pass TP, that is, there is a test purpose $\mathcal{T} \in TP$ with $(t_0, q_0) \overset{\sigma}{\Rightarrow} (\textsf{fail}, q)$ in $\mathcal{T} \times \mathcal{I}_k$, for some $\sigma \in \{a, x\}^*$ and some state q of $\mathcal{I}_k$. But then Proposition 5.4 gives $t_0 \overset{\sigma}{\Rightarrow} \textsf{fail}$ in $\mathcal{T}$ and $q_0 \overset{\sigma}{\Rightarrow} q$ in $\mathcal{I}_k$. By the maximality of k we obtain $\sigma = a^m$ for some $m \le k$. Then, $t_0 \overset{a^m}{\Rightarrow} \textsf{fail}$ in $\mathcal{T}$. Since, clearly, $s_0 \overset{a^m}{\Rightarrow} s_0$ in $\mathcal{S}$, Proposition 5.4 gives $(t_0, s_0) \overset{a^m}{\Rightarrow} (\textsf{fail}, s_0)$ in $\mathcal{T} \times \mathcal{S}$. Hence, $\mathcal{S}$ does not pass TP, and so we do not have $\mathcal{S}$ **ioco** $\mathcal{S}$, which is a clear contradiction. □

It is not hard to see that, for the specification IOLTS in the proof of Proposition 5.9 we could as well have taken any IOLTS $\mathcal{S}$ with a state s that is reachable from the initial state of $\mathcal{S}$, and with a self-loop at state s on any input symbol. This observation makes the result at Proposition 5.9 much more widely applicable.

Now we examine the case when we have an upper bound on the number of states in implementations.

Definition 5.10. *Let $\mathcal{IMP} \subseteq \mathcal{IO}(I, U)$, and let $m \ge 1$. We denote by $\mathcal{IMP}[m]$ the subfamily of $\mathcal{IMP}$ comprised by all $\mathcal{S} \in \mathcal{IMP}$ with at most m states. Let $\mathcal{S} \in \mathcal{IO}(I, U)$. We say that a fault model TP over $L_I \cup L_U$ is m-**ioco**-complete for $\mathcal{S}$ relatively to $\mathcal{IMP}$ if, for any $\mathcal{I} \in \mathcal{IMP}[m]$ it holds that $\mathcal{I}$ passes TP if and only if $\mathcal{I}$ **ioco** $\mathcal{S}$.*

Since it is not possible to construct an acyclic finite fault model that is **ioco**-complete in general, we turn to the problem of obtaining finite acyclic fault models that are m-**ioco**-complete.

Proposition 5.11. *Let $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle \in \mathcal{IO}(I, U)$ be deterministic, and let $m \ge 1$. Then, there is a fault model TP which is **ioco**-complete for $\mathcal{S}$ relatively to $\mathcal{IO}(I, U)[m]$, and such that all test purposes in TP are deterministic and acyclic.*

Demonstração. Let $|S| = n$, and let $s_0, s_1, \ldots, s_{n-1}$ be an enumeration of the states in $\mathcal{S}$. We construct a direct acyclic multi-graph D, based on the transitions in T.

We make D with $mn + 1$ levels, where at each level i, $0 \le i \le mn$, we list all nodes in S, in the given order, from left to right, labeling them $s_{0,i}$, $s_{1,i}$,…, $s_{n-1,i}$. Next, we add transitions to D. Consider a node $s_{j,k}$, $0 \le j \le n$, at level k, $0 \le k < mn$. For each transition (s_j, ℓ, s_i) of $\mathcal{S}$: (i) if $i > j$, add a left to right arc to D from node $s_{j,k}$ to node $s_{i,k}$, both at level k; and (ii) if $i \le j$ add an arc to D from node $s_{j,k}$ at level k to node $s_{i,k+1}$ at level $k + 1$. In both cases, label the new arc with the symbol ℓ. We complete the construction of D by adding to it an extra node, labeled **fail**. Further, for any node $s_{i,k}$ in D, $s_{i,k} \neq \textsf{fail}$, and any $\ell \in L_U$, if $(s_i, \ell, p) \notin T$, for every $p \in S$, then we add an arc from node $s_{i,k}$ to node **fail** in D, and label it ℓ. Let $s_{0,0}$ be the root node. Finally, any node in D that is not reachable from the root node is discarded.

Since all arcs in D are directed top-down or from left to right, it is clear that D is acyclic. Hence, there is a finite number of distinct maximal paths starting at the root node of D. Also, since at any node $s_{i,k}$ we add arcs to D corresponding to all transitions in $\mathcal{S}$ from node s_i, it is clear that, for all $\sigma \in L^*$, with $|\sigma| \le mn$, we have that $\sigma \in otr(\mathcal{S})$ if and only if σ is a path from the root of D.

Once D is constructed, we obtain the desired acyclic fault model TP which is m-**ioco**-complete for $\mathcal{S}$. For each path $s_{i_0} \overset{x_1}{\rightarrow} s_{i_1} \overset{x_2}{\rightarrow} \cdots \overset{x_{i_r}}{\rightarrow} s_{i_r}$ ($r \ge 1$ and $0 \le i \le n-1$) in D where $s_{i_0} = s_{0,0}$ is the root node and $s_{i_r} = \textsf{fail}$, we add the acyclic test purpose

$$\mathcal{T} = (\{s_{i_0}, \ldots, s_{i_r}\}, s_{i_0}, L_U, L_I, \{(s_{i_{j-1}}, x_j, s_{i_j} \,|\, 1 \le j \le r\})$$

to TP. It is also clear that $\mathcal{T}$ is deterministic.

Now we want to argue for the m-**ioco**-completeness of TP. First, assume that $\mathcal{I}$ **ioco** $\mathcal{S}$ does not hold, where $\mathcal{I} = (Q, q_0, L_I, L_U, R)$ is an implementation IOLTS with $|Q| = h \le m$ states. Then, by

Proposition 3.2 and Lemma 3.7 we know that there is some $\sigma \in L^*$ such that $\sigma \in otr(\mathcal{I})$, $\sigma \notin otr(\mathcal{S})$ 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, $s_0 \overset{\alpha}{\Rightarrow} s$ in $\mathcal{S}$. From $\alpha\ell \in otr(\mathcal{I})$ we also have $q_0 \overset{\alpha}{\Rightarrow} q \overset{\ell}{\Rightarrow} q'$ in $\mathcal{I}$. By Proposition 5.4, we have $(s_0, q_0) \overset{\alpha}{\Rightarrow} (s, q)$ in $\mathcal{S} \times \mathcal{I}$. With loss of generality, let $\alpha = x_1 \dots x_r$ $(r \geq 0)$. Then,

$$(s_0, q_0) \overset{x_1}{\Rightarrow} (s_1, q_1) \overset{x_2}{\Rightarrow} (s_2, q_2) \overset{x_3}{\Rightarrow} s \overset{x_r}{\Rightarrow} (s_r, q_r),$$

with $s = s_r$ and $q = q_r$. If $r \geq mn \geq hn$ we get $(s_i, q_i) = (s_j, q_j)$ for some $0 \leq i < j \leq r$. Then, $(s_0, q_0) \overset{\mu}{\Rightarrow} (s_r, q_r)$ in $\mathcal{S} \times \mathcal{I}$, and with $\mu = x_1 \dots x_i x_{j+1} \dots x_r$. Again, $s_0 \overset{\mu}{\Rightarrow} s_r$ and $q_0 \overset{\mu}{\Rightarrow} q_r$, which gives $\mu \in otr(\mathcal{S})$ and $\mu \in otr(\mathcal{I})$. Moreover, $(s_r, \ell, p) \notin T$, otherwise we would get $\sigma = \alpha\ell \in otr(\mathcal{S})$, a contradiction. Hence, $\mu\ell \notin otr(\mathcal{S})$. Also, since $\mu \in otr(\mathcal{S})$, we get $\mu\ell \in otr(\mathcal{S})L_U$. Further, we also have $q_0 \overset{\mu}{\Rightarrow} q_r$ in $\mathcal{I}$, $q_0 \overset{\alpha}{\Rightarrow} q \overset{\ell}{\Rightarrow} q'$ in $\mathcal{I}$, and $q = q_r$. Hence, $q_0 \overset{\mu}{\Rightarrow} q \overset{\ell}{\Rightarrow} q'$ in $\mathcal{I}$, and so $\mu\ell \in otr(\mathcal{I})$. Thus, $\mu\ell \in otr(\mathcal{I}) \cap \big[\overline{otr}(\mathcal{S}) \cap otr(\mathcal{S})L_U\big]$. But $|\mu| < |\alpha|$ and so $|\mu\ell| < |\alpha\ell| = |\sigma|$, violating the minimality of $|\sigma|$. Hence, $r < mn$. Thus, from $s_0 \overset{\alpha}{\Rightarrow} s_i$ in $\mathcal{S}$ we now get a path labeled α from the root node $s_{0,0}$ to a node $s_{i,k}$ in D. Because (s_i, ℓ, p) is not in T for any $\ell \in L_U$ and any $p \in S$ (otherwise we would have $\sigma = \alpha\ell \in otr(\mathcal{S})$), by the construction we have an arc labeled ℓ from $s_{i,k}$ to **fail** in D. Hence, here is a path labeled $\alpha\ell$ from the root node to **fail** in D. Thus, we have a test purpose $\mathcal{T} = (S_\mathcal{T}, t_0, L_U, L_I, T_\mathcal{T})$ in TP with $t_0 \overset{\alpha\ell}{\Rightarrow}$ **fail** in $\mathcal{T}$, and so we also have $(t_0, q_0) \overset{\sigma}{\Rightarrow}$ (**fail**, q') in $\mathcal{T} \times \mathcal{I}$, thus showing that $\mathcal{I}$ does not pass TP.

For the converse, if an implementation $\mathcal{I} = (Q, q_0, L_I, L_U, R)$ does not pass TP, we have $(t_0, q_0) \overset{\sigma}{\Rightarrow}$ (**fail**, q) in $\mathcal{T} \times \mathcal{I}$, for some test purpose $\mathcal{T} = (S_\mathcal{T}, s_{0,0}, L_U, L_I, T_\mathcal{T})$ in TP. So, $s_{0,0} \overset{\sigma}{\Rightarrow}$ **fail** in $\mathcal{T}$ and $q_0 \overset{\sigma}{\Rightarrow} q$ in $\mathcal{I}$. Thus, $\sigma \in otr(\mathcal{I})$. By construction of $\mathcal{T}$, we have $s_{0,0} \overset{\alpha}{\Rightarrow} s_{i,k} \overset{\ell}{\Rightarrow}$ **fail** in $\mathcal{T}$, and we had known that (s_i, ℓ, p) is not a transition in $\mathcal{S}$, for all $p \in S$ and all $\ell \in L_U$. From $s_{0,0} \overset{\alpha}{\Rightarrow} s_{i,k}$ in $\mathcal{T}$ and the construction of $\mathcal{T}$, we have a path labeled α from the root node in D, and so we also get $s_0 \overset{\alpha}{\Rightarrow} s_i$ in $\mathcal{S}$, that is, we have $\sigma = \alpha\ell \in otr(\mathcal{S})L_U$. If $\alpha\ell \in otr(\mathcal{S})$ we would have $s_0 \overset{\alpha}{\Rightarrow} s' \overset{\ell}{\Rightarrow} s''$ in $\mathcal{S}$. Since $\mathcal{S}$ is deterministic, we get $s_i = s'$, 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 Lemma 3.7 and Proposition 3.2 we have that $\mathcal{I}$ **ioco** $\mathcal{S}$ does not hold.

We conclude that, for any implementation $\mathcal{I}$ with at most m states, $\mathcal{I}$ does not pass TP if and only if $\mathcal{I}$ **ioco** $\mathcal{S}$ does not hold, that is, TP is m-**ioco**-complete for $\mathcal{S}$. □

As a couple of final restrictions in the structure of test purposes [27], consider any test purpose $\mathcal{T} = \langle S_\mathcal{T}, t_0, L_U, L_I, T_\mathcal{T} \rangle$. First one would like it to be input-enabled, so that the tester does not block upon receiving any symbol emitted by the implementation. Recall Definition 5.1. Also, because we do not want the tester to block at **fail** and **pass** states, self-lops on all input-symbols are also arranged for those states. Further, since a tester drives the IUT, one would like that, at any $t \in S_\mathcal{T}$, the tester to be output-deterministic, that is, it can emit only one action symbol from L_I to the implementation. Next, we frame this notion precisely. Recall Definition 2.31.

Definition 5.12. *Let $\mathcal{S} \in \mathcal{IO}(I, U)$. We say that $\mathcal{S}$ is* output-deterministic *if $|\,\mathbf{out}(s)| = 1$, for all $s \in S$.*

It is a simple matter to 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 5.13. *Let $\mathcal{S} \in \mathcal{IO}(I, U)$ be deterministic, and let $m \geq 1$. Then, there is a fault model TP which is* **ioco***-complete for $\mathcal{S}$ relatively to $\mathcal{IO}(I, U)[m]$, and such that all test purposes in TP are deterministic, input-enabled, output-deterministic, and acyclic except for self-loops at special* **fail** *and* **pass** *states.*

Demonstração. From Proposition 5.11 we get a fault model TP that is m-**ioco**-complete for $\mathcal{S}$, and such that all test purposes in TP are acyclic. Consider any test purpose $\mathcal{T} = \langle S_\mathcal{T}, t_0, L_U, L_I, T_\mathcal{T}\rangle$ in TP, and take any state t of $\mathcal{T}$.

In order to secure input-enabledness, we add a new **pass** state to $S_\mathcal{T}$. For all $\ell \in L_U$ we proceed as follows. If we do not have a transition (t, ℓ, t') in $T_\mathcal{T}$ for any $t' \in S_\mathcal{T}$, we add the transition $(t, \ell, \textsf{pass})$ to $T_\mathcal{T}$. Also, for the special states, we add transitions $(\textsf{pass}, \ell, \textsf{pass})$ and $(\textsf{fail}, \ell, \textsf{fail})$ to $T_\mathcal{T}$, for all $\ell \in L_U$. This will transform $\mathcal{T}$ into an input-enabled acyclic IOLTS $\mathcal{T}'$, except for self-loops at **pass** and **fail** states. It is clear that, for any implementation $\mathcal{I} = \langle S_\mathcal{I}, q_0, L_I, L_U, T_\mathcal{I}\rangle \in \mathcal{IO}(I, U)$ if we have $(t_0, q_0) \overset{\sigma}{\Rightarrow} (\textsf{fail}, q)$ in $\mathcal{T} \times \mathcal{I}$ then we also have $(t_0, q_0) \overset{\sigma}{\Rightarrow} (\textsf{fail}, q)$ in $\mathcal{T}' \times \mathcal{I}$, for all observable behaviors σ. And conversely, if we have $(t_0, q_0) \overset{\sigma}{\Rightarrow} (\textsf{fail}, q)$ in $\mathcal{T}' \times \mathcal{I}$ for some observable behavior σ, then we also have $(t_0, q_0) \overset{\alpha}{\Rightarrow} (\textsf{fail}, q')$ in $\mathcal{T} \times \mathcal{I}$, where $\sigma = \alpha\beta$, $\beta \in (L_I \cup L_U)^\star$. We conclude that $\mathcal{I}$ passes $\mathcal{T}$ if and only if $\mathcal{I}$ passes $\mathcal{T}'$. Hence, after adjusting all test purposes in TP we get a fault model TP' that is m-**ioco**-complete for $\mathcal{S}$, and such that all test purposes in TP are input-enabled and acyclic, except for the self-loops at states **pass** and **fail**.

We now argue for output-determinism. A simple re-examination of the proof of Proposition 5.11 reveals that we have at most one outgoing transition (t, ℓ, t') in $\mathcal{T}'$. Since in the previous step we added no transitions on a symbol from L_I, if we have $\ell \in L_I$ then state t is already output-deterministic. If $\ell \notin L_I$ we choose any symbol $\ell' \in L_I$ and add a transition $(t, \ell', \textsf{pass})$ to $\mathcal{T}'$. This makes state t output-deterministic. After we apply this transformation to all states in $\mathcal{T}'$ we get a new test purpose $\mathcal{T}''$ which is output-deterministic. Again, it is easy to see that for any implementation $\mathcal{I} \in \mathcal{IO}(I, U)$, we have that $\mathcal{I}$ passes $\mathcal{T}'$ if and only if $\mathcal{I}$ passes $\mathcal{T}''$. We, thus, reach the conclusion we can always get a fault model TP'' which is m-**ioco**-complete for $\mathcal{S}$, and such that all test purposes in TP are input-enabled, output-deterministic, and acyclic, except for the self-loops at the **pass** and **fail** states. □

Putting these partial results together we reach the following conclusion.

Theorem 5.14. *Let $\mathcal{S} \in \mathcal{IO}(I, U)$ be a specification, and let $m \geq 1$. Then we can effectively construct a finite fault model TP which is* **ioco***-complete for $\mathcal{S}$ relatively to $\mathcal{IO}(I, U)[m]$, and such that all test purposes in TP are input-enabled, output-deterministic, and acyclic except for self-loops at special* **fail** *and* **pass** *states.*

Demonstração. First, if $\mathcal{S}$ is not already deterministic, use Proposition 2.26 to transform $\mathcal{S}$ into an equivalent deterministic IOLTS, so that we may assume that $\mathcal{S}$ is deterministic. Now use Proposition 5.13. □

Example 5.15. Let the IOLTS $\mathcal{S}$ of Figure 6 be the specification again. We construct a direct acyclic multi-graph $\mathcal{D}$ as depicted at Figure 12 as described in Lemma 5.11. As $\mathcal{S}$ has $n = 4$ states we then have four states at each level, and all transitions connecting nodes at the same level must go from left to the right. Any other transition on $\mathcal{S}$ whose direction would come out a cycle must go to next level. Hence any observable behavior starting at root $s_{0,0}$ runs from left to right, and from top to bottom. In this example we have considered IUTs with at most $m = n = 4$ states, so that $n^2 + 1$ levels have been constructed in the multi-graph $\mathcal{D}$.

To complete the construction we also added the **fail** state. But, in order to keep the figure uncluttered we replicated the **fail** label, and we did not represent the self-loops $(\textsf{fail}, x, \textsf{fail})$. Also

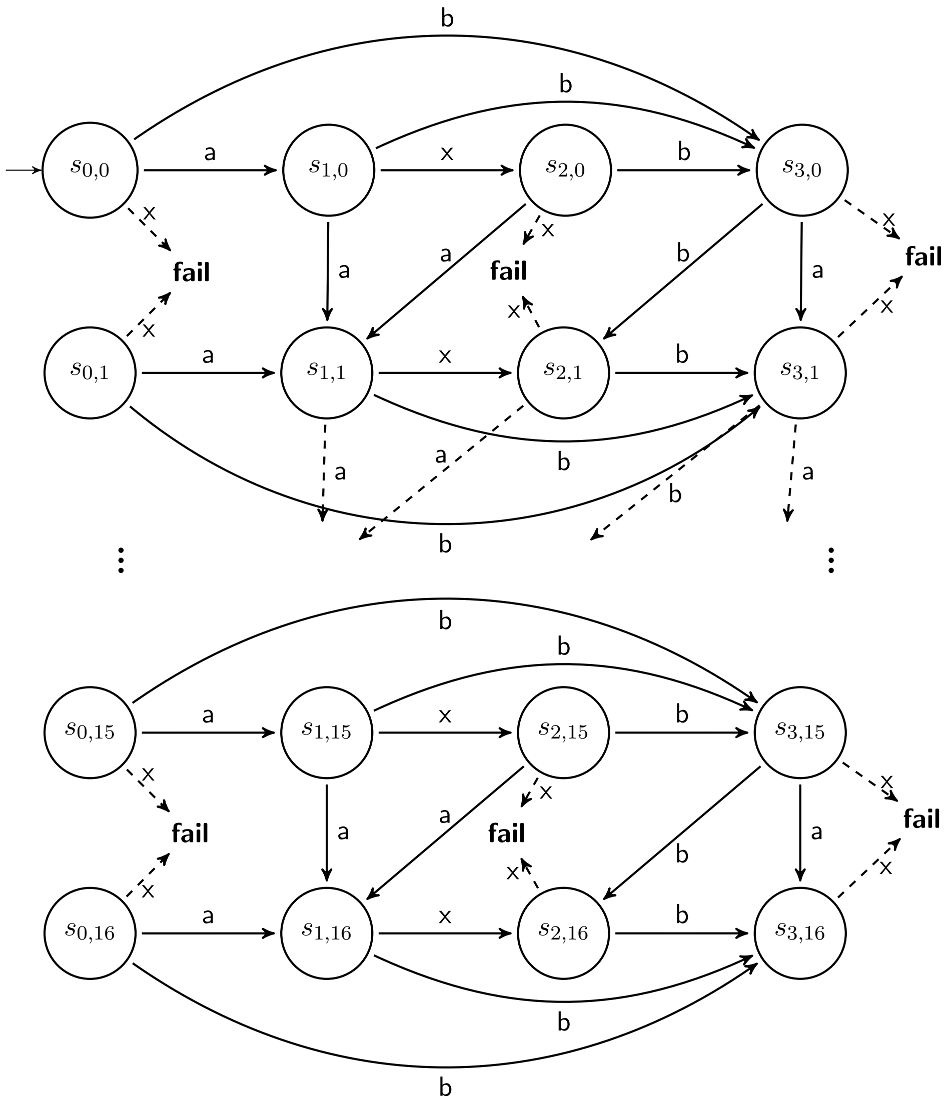

Figura 12: A direct acyclic multi-graph D for the specification depicted at Figure 6.

notice that, in this example, one does not need to add the **pass** state since $\mathcal{S}$ was already deterministic, input-enabled and output-deterministic.

Now a simple algorithm can extract test purposes $\mathcal{T}$ by traversing the graph from the root $s_{0,0}$ to a **fail** state. For instance, take the observable behavior $\alpha = aaxabbbax$ with $|\alpha| \leq mn$. We can easily check that α leads $\mathcal{D}$ from $s_{0,0}$ to the **fail** state, that is $s_{00} \overset{\alpha}{\Rightarrow}$ **fail**, by the sequence of states $s_{0,0}, s_{10}, s_{11}, s_{21}, s_{12}, s_{32}, s_{23}, s_{33}, s_{34},$ **fail** in $\mathcal{D}$. Assume the IOLTS $\mathcal{I}$ of Figure 8 as an IUT. By a simple inspection we see that α leads $\mathcal{I}$ from q_0 to q_2. In this case, we check that $\mathcal{I}$ does not pass the fault model induce by α in $\mathcal{D}$ and so $\mathcal{I}$ **ioco** $\mathcal{S}$ does not hold. □

We note that Theorem 5.14 implies that we can always effectively construct fault models that are **ioco**-complete for any given specification, and when implementations to be put under test have at most m states. Moreover, all test purposes that comprise the fault model will satisfy all properties required of a test case, as specified in [27]. We also remark that, in [27], IOLTSs are required to be input-enabled, that is, they belong to the class $\mathcal{IOE}(I, U)$. This is a further restriction that just

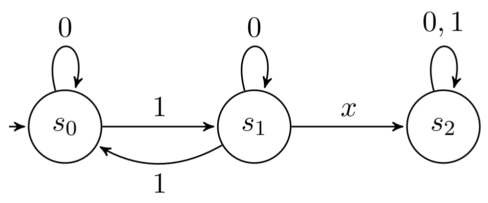

Figura 13: A specification IOLTS.

makes this kind of IOLTS a special flavor of the more general IOLTS models we treated here. If this restriction is to be observed, we have the following related result.

Corollary 5.16. *Let $\mathcal{S} \in \mathcal{IOE}(I, U)$ be a specification, and let $m \geq 1$. Then we can effectively construct a finite fault model TP which is* **ioco***-complete for $\mathcal{S}$ relatively to $\mathcal{IOE}(I, U)[m]$, and such that all test purposes in TP are input-enabled, output-deterministic, and acyclic except for self-loops at special* **fail** *and* **pass** *states.*

Demonstração. Follows from Theorem 5.14, since $\mathcal{IOE}(U, I)$ is a subclass of $\mathcal{IO}(I, U)$. □

If, on the other hand, one has a somewhat different set of characteristics that test purposes should satisfy, as compared to those in [27], one might try to proceed as discussed in this subsection, and transform each basic test purpose so as to make it adhere to that specific set of desired characteristics.

5.3 On the Complexity of Test Purposes

We now 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 IOLTS, with $|S| = n$. First, we return to the construction of the acyclic multigraph D, described in the proof of Proposition 5.11, and that was used to obtain acyclic test cases 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 the traces in D, the fault model that is so generated might also be of super-polynomial size on n and m. We argue now that, in general, this situation is unavoidable.

Theorem 5.17. *Let $m \geq 3$, $L_I = \{0, 1\}$, and $L_U = \{x\}$. There is a simple deterministic specification $\mathcal{S} \in \mathcal{IOE}(I, U)$ for which any fault model TP that is* **ioco***-complete for $\mathcal{S}$ relatively to $\mathcal{IOE}(I, U)[m]$, and such that any test purpose $\mathcal{T}$ in TP is deterministic and output-deterministic, must be of size $\Omega(\Phi^m)$, where $\Phi = (1 + \sqrt{5})/2 \approx 1.61803$.*

Demonstração. Consider the simple specification IOLTS $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle$ of Figure 13. Let $L = L_I \cup L_U$ and define R as the regular language $R = (0 + 11)^*$. To ease the notation, if $\sigma = 0$ we write $\overline{\sigma} = 1$, and when $\sigma = 1$ we let $\overline{\sigma}$ denote 0.

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 $\alpha x \notin otr(\mathcal{S})$.

Now let $\mathcal{I}_\alpha = \langle S_\mathcal{I}, q_o, L_I, L_U, T_\mathcal{I}\rangle$ be the implementation IOLTS comprised by the transitions (q_{i-1}, y_i, q_i) for $1 \leq i \leq r$, and (q_r, x, q_{r+1}). We want to guarantee that $\mathcal{I}_\alpha$ is input-enabled. So, we add a new state, **pass**, to $\mathcal{I}_\alpha$, and add the transitions $(q_{i-1}, \overline{y_i}, \textsf{pass})$ $(1 \le i \le r)$; lastly we add $(q_r, \sigma, \textsf{pass})$ and the self-loops $(q_{r+1}, \sigma, q_{r+1})$ and $(\textsf{pass}, \sigma, \textsf{pass})$, for $\sigma \in \{0,1\}$. See Figure 14. Clearly, $\mathcal{I}_\alpha$ has $r+2 \le m$ states and is input-enabled, that is $\mathcal{I}_\alpha \in \mathcal{IOE}(I,U)[m]$.

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 that $\mathcal{I}_\alpha$ **ioco** $\mathcal{S}$ does not hold, by Corollary 3.10. Because TP is m-**ioco**-complete for $\mathcal{S}$ relatively to $\mathcal{IOE}(I,U)$, it follows from Definition 5.10 that $\mathcal{I}_\alpha$ does not pass TP. Hence, there is some test purpose $\mathcal{T}_\alpha = \langle S_\alpha, t_0, L_U, L_I, T_\alpha\rangle$ in TP such that $\mathcal{I}_\alpha$ does not pass $\mathcal{T}_\alpha$. From Definition 5.5 we get some $\sigma \in L^*$ such that $(t_0, q_0) \overset{\sigma}{\Rightarrow} (\textsf{fail}, q)$ in $\mathcal{T}_\alpha \times \mathcal{I}_\alpha$, for some $q \in S_\mathcal{I}$. By Proposition 5.4 we obtain $t_0 \overset{\sigma}{\Rightarrow}$ **fail** in $\mathcal{T}_\alpha$ and $q_0 \overset{\sigma}{\Rightarrow} q$ in $\mathcal{I}_\alpha$. Suppose that $\sigma \in \{0,1\}^\star$. This would give $s_0 \overset{\sigma}{\Rightarrow} s$ in $\mathcal{S}$, and Proposition 5.4 would yield $(t_0, s_0) \overset{\sigma}{\Rightarrow} (\textsf{fail}, s)$ in $\mathcal{T}_\alpha \times \mathcal{S}$. But this is a contradiction because it would imply that $\mathcal{S}$ does not pass TP and, $\mathcal{S}$ being in $\mathcal{IOE}(I,U)$, would imply that $\mathcal{S}$ **ioco** $\mathcal{S}$ does not hold, a contradiction. Since $\sigma \in otr(\mathcal{I}_\alpha)$, we must then have $\sigma = \alpha x \alpha'$ where $\alpha' \in \{0,1\}^*$.

Now let $\beta \in R$ with $|\beta| = |\alpha|$ and $\beta \neq \alpha$. Using a similar reasoning to the one just given, we get a test purpose $\mathcal{T}_\beta = \langle S_\beta, t'_0, L_U, L_I, T_\beta\rangle$, with $t'_0 \overset{\beta x \beta'}{\Rightarrow}$ **fail** in $\mathcal{T}_\beta$. Assume that $\mathcal{T}_\alpha = \mathcal{T}_\beta$. Then, we get $t_0 \overset{\alpha x \alpha'}{\Rightarrow}$ **fail** and $t_0 \overset{\beta x \beta'}{\Rightarrow}$ **fail** in $\mathcal{T}_\alpha$. Now, $\alpha \neq \beta$ and $|\alpha| = |\beta|$ give $\alpha = \mu x_1 \alpha_1$ and $\beta = \mu x_2 \beta_1$ with x_1, $x_2 \in \{0,1\}$, $x_1 \neq x_2$, $|\alpha_1| = |\beta_1|$, and μ, α_1, $\beta_1 \in \{0,1\}^\star$. Since prefixes of α and of β are in $\{0,1\}^*$, w can not reach the **fail** state in $\mathcal{T}_\alpha$ with such prefixes. Hence, we must have $t_0 \overset{\mu}{\Rightarrow} t_1 \overset{x_1}{\Rightarrow} t_2 \overset{\alpha_1 x \alpha'}{\Rightarrow}$ **fail** and $t_0 \overset{\mu}{\Rightarrow} t'_1 \overset{x_2}{\Rightarrow} t'_2 \overset{\beta_1 x \beta'}{\Rightarrow}$ **fail** in $\mathcal{T}_\alpha$. Since $\mathcal{T}_\alpha$ is deterministic, we get $t_1 = t'_1$ and the transitions

$$(t_1, x_1, t_2) \quad \text{and} \quad (t_1, x_2, t'_2) \tag{3}$$

in $\mathcal{T}_\alpha$, with $x_1 \neq x_2$ in $\{0,1\}$, which is the output alphabet for $\mathcal{T}_\alpha$, contradicting the fact that $\mathcal{T}_\alpha$ is output-deterministic. We conclude that, in fact we must have $\mathcal{T}_\alpha \neq \mathcal{T}_\beta$ when $\alpha \neq \beta$ and $|\alpha| = |\beta|$.

Now, since 1 occurs only in blocks of two in each word in R, there are $\binom{m-i}{i}$ distinct words of length m with i such blocks in R. So, we have

$$\sum_{i=0}^{\lfloor m/2 \rfloor} \binom{m-i}{i} \tag{4}$$

words of length m in R. But Eq. (4) represents the mth Fibonacci number F_m, where $F_0 = 1$, $F_1 = 1$ and $F_n = F_{n-1} + F_{n-2}$ for $n \ge 2$. It is a known fact that

$$F_m = \frac{1}{\sqrt{5}}\left(\Phi^m + \frac{1}{\Phi^m}\right) \ge \frac{\Phi^m}{\sqrt{5}},$$

where $\Phi = (1+\sqrt{5})/2$. We conclude that we must have at least $\Phi^m/\sqrt{5}$ test cases in TP. Hence, any fault model TP which is deterministic, m-**ioco**-complete for $\mathcal{S}$, and also output-deterministic, must contain at least $\Phi^m/\sqrt{5}$ distinct test purposes. Hence, TP must be, asymptotically, of size at least $\Omega(\Phi^m)$, that is, exponential in m. □

It is also clear that Theorem 5.17 also applies to any specification $\mathcal{S}$ in which Figure 13 occurs, with s_0 being reachable from the initial state of $\mathcal{S}$.

6 Fault Domains as Scheme Sets

In this section we want to show that our proposal also accommodates a more recent approach for generating test suites for IOLTSs [22] based on the **ioco** conformance relation. We will need some simple definitions, following [22]. Recall Definition 2.31.

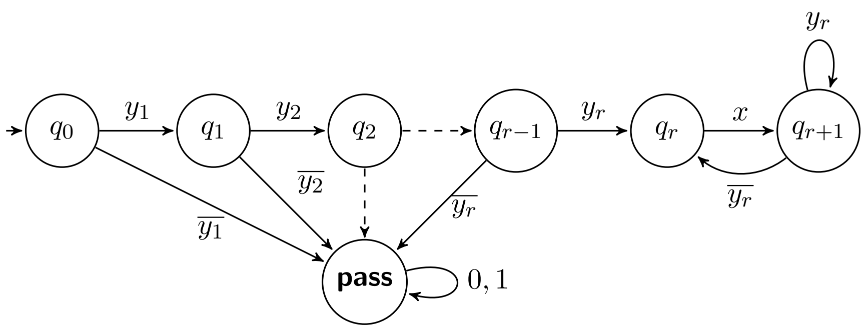

Figura 14: Implementation $\mathcal{I}_\alpha$.

Definition 6.1 ([22])**.** *Let $\mathcal{S} = \langle S, s_0, L_I, L_U, T\rangle \in \mathcal{IO}(I, U)$, with $L = L_I \cup L_U$. A state $s \in S$ is a* sink state *if* $\mathbf{init}(\{s\}) = \emptyset$*;* single-input *when* $|\mathbf{inp}(\{s\})| \leq 1$*; and* input-state *when* $\mathbf{inp}(\{s\}) \neq \emptyset$*. We say that s is* input-complete *when* $\mathbf{inp}(\{s\}) = L_I$ *or* $\mathbf{inp}(\{s\}) = \emptyset$ *and, likewise, that s is* output-complete *when* $\mathbf{out}(\{s\}) = L_U$ *or* $\mathbf{out}(\{s\}) = \emptyset$*. We say that $\mathcal{S}$ is single-input, input-complete, or output-complete, if any state in S is, 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 at least one transition $q_{j-1} \xrightarrow{x_j} q_j$ with $x_j \in L_I$, for some $1 \leq j \leq k$.*

Let $\mathcal{IOIP}(I, U) \subseteq \mathcal{IO}(I, U)$ denote the class of all IOLTSs which are deterministic, input-complete, progressive and initially-connected.

In Definition 5.1, a state s was said to be input-enabled when $\mathbf{inp}(s) = L_I$. Note the slight difference with the notion of s being input-complete just given.

In Definition 4.1, the terms test case and test suite were defined to refer to words and languages, respectively, over $L_I \cup L_U$. In order to not overload the use of these terms, in this section we will define *schemes* and *scheme suites*, respectively, when referring to the notions of test cases and test suites as in Definition 3 of Simao and Petrenko [22]. Note that, contrary to the notion of a test purpose in Definition 5.2, test schemes in [22] do not have their sets of input and output symbols reversed with respect to the sets of input and output symbols specifications and implementations. The next definition reflects this idea.

Definition 6.2. *Let L_I and L_U be sets of symbols 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 $\mathcal{T} \in \mathcal{IO}(I, U)$ 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 5.3, of the cross-product operator $\mathcal{S} \times \mathcal{I}$ for synchronous execution of two IOLTSs $\mathcal{S}$ and $\mathcal{I}$. In [22], the exactly same operator is denoted by $\mathcal{S} \cap \mathcal{I}$, with the proviso that, in that work, internal τ-moves were not considered, by definition. In this section we will continue to use the cross-product to denote synchronous execution. This being noted, we remark now that our Definition 5.5, for when an implementation IOLTS $\mathcal{I}$ passes a test scheme $\mathcal{T}$ and passes a scheme suite SS, exactly matches Definition 4 of Simao and Petrenko [22] and, as a consequence, we also have the very same notion of **ioco**-completeness, as stated in our Definition 5.5 and in Definition 4 of Simao and Petrenko [22].

We first 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 implementations models.

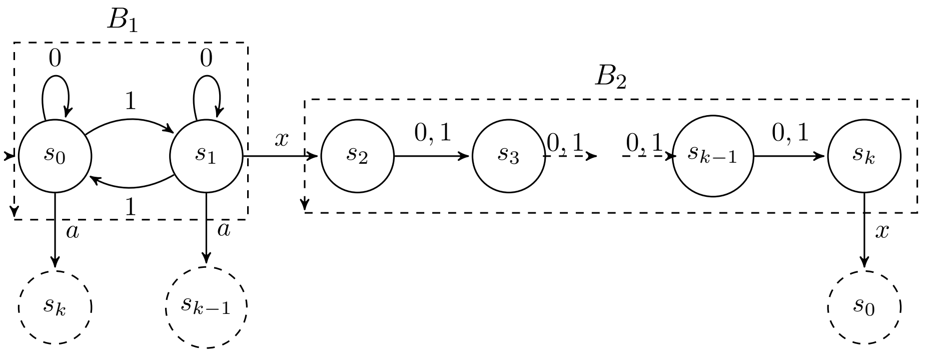

Figura 15: Specification $\mathcal{S}'$, modifying specification $\mathcal{S}$ of Figure 13.

Theorem 6.3. *Let $\mathcal{S} = \langle S_{\mathcal{S}}, s_0, L_I, L_U, R_{\mathcal{S}}\rangle \in \mathcal{IO}(I,U)$ a specification over $L = L_I \cup L_U$, and let $m \geq 1$. Then we can effectively construct a finite scheme suite SS over L which is* **ioco***-complete for $\mathcal{S}$ relatively to $\mathcal{IO}(I,U)[m]$.*

Demonstração. We start with the Proposition 2.26 to transform $\mathcal{S}$ into an equivalent deterministic IOLTS, if $\mathcal{S}$ is not already deterministic. Now we use the Proposition 5.11 and then use an argument similar to the one in Proposition 5.13.

Using Proposition 5.11, we get a scheme suite SS which is **ioco**-complete for $\mathcal{S}$ relatively to $\mathcal{IO}(I,U)[m]$, and such that all schemes in SS are deterministic and acyclic IOLTSs. From the proof of Proposition 5.11, it is clear that all schemes in SS have a single **fail** state, which is also a sink state.

Let $\mathcal{T} = \langle S_{\mathcal{T}}, t_0, L_I, L_U, R_{\mathcal{T}}\rangle$ in SS be any scheme constructed as in the proof of Proposition 5.11, and let $s \in S_{\mathcal{T}}$ be any state of $\mathcal{T}$. 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 $p \in S_{\mathcal{T}}$. There are two cases for $\ell \in L_U \cup L_I$. If $\ell \in L_I$, from Definition 6.1, we immediately get that $|\,\mathbf{inp}(s)| \leq 1$ and $\mathbf{out}(s) = \emptyset$, that is, s is single-input and output-complete. Now, assume $\ell \in L_U$ is an output symbol of $\mathcal{T}$. Then, s is already single-input. In order to make s output-complete, transform scheme $\mathcal{T}$ to a scheme $\mathcal{T}' = \langle S'_{\mathcal{T}}, t_0, L_I, L_U, R'_{\mathcal{T}}\rangle$ by adding a new **pass** state to $S_{\mathcal{T}}$ and, for any other $x \in L_U$, with $x \neq \ell$, add a transition $(s, x, \mathbf{pass})$ to $R_{\mathcal{T}}$. It is clear that s is now output-complete.

Since **pass** is a sink state in $\mathcal{T}'$, for any implementation $\mathcal{I} = \langle S_{\mathcal{I}}, q_0, L_I, L_U, R_{\mathcal{I}}\rangle \in \mathcal{IO}(I,U)$, we have that $(t_0,q_0) \overset{\star}{\Rightarrow} (\mathbf{fail}, q)$ in $\mathcal{T}$ if and only if $(t_0,q_0) \overset{\star}{\Rightarrow} (\mathbf{fail}, q)$ in $\mathcal{T}'$, for any $q \in S_{\mathcal{I}}$. Therefore, we also get that $\mathcal{I}$ passes $\mathcal{T}$ if and only if $\mathcal{I}$ passes $\mathcal{T}'$. Let SS' be the scheme suite obtained from SS with the transformation just discussed, now applied to each scheme in SS. Since SS is **ioco**-complete for $\mathcal{S}$ relatively to $\mathcal{IO}(I,U)[m]$, then SS' is also **ioco**-complete for $\mathcal{S}$ relatively to $\mathcal{IO}(I,U)[m]$.

We conclude that SS' satisfies all the requirements in Definition 6.2, and the proof is complete. □

A similar result follows if we restrict all specifications and implementations to be member of the $\mathcal{IOIP}(I,U)$ class.

Corollary 6.4. *Let $\mathcal{S} \in \mathcal{IOIP}(I,U)$ be a specification, with $L = L_I \cup L_U$, and let $m \geq 1$. Then we can effectively construct a finite scheme suite SS over L which is* **ioco***-complete for $\mathcal{S}$ relatively to the sub-class of $\mathcal{IOIP}(I,U)[m]$ in which every implementation has no more input-states then $\mathcal{S}$.*

Demonstração. Since $\mathcal{IOIP}(I,U) \subseteq \mathcal{IO}(I,U)$, this follows immediately from Theorem 6.3. □

In [22], specifications and implementations are further restricted to be input-state-minimal, in the sense that any two distinct input-states are always distinguishable. In their Corollary 1, they characterize two states r and p as being distinguishable when there are no sink state in the cross-product $\mathcal{S}_{/r} \times \mathcal{S}_{/p}$, where $\mathcal{S}_{/p}$ stands for the same model as $\mathcal{S}$, but now with p being the initial state. We formalize these notions next.

Definition 6.5 ([22])**.** *Let $\mathcal{IOC}(I,U) \subseteq \mathcal{IO}(I,U)$ be a class of IOLTS models, and take two models $\mathcal{S} = \langle S_\mathcal{S}, s_0, L_I, L_U, R_\mathcal{S}\rangle$, $\mathcal{Q} = \langle S_\mathcal{Q}, q_0, L_I, L_U, R_\mathcal{Q}\rangle$ in $\mathcal{IOC}(I,U)$, with $s \in S_\mathcal{S}$ and $q \in S_\mathcal{Q}$. We say that s and q are* distinguishable *in the class $\mathcal{IOC}(I,U)$ if and only if there is a sink state in the cross-product $\mathcal{S}_{/s} \times \mathcal{Q}_{/q}$. Otherwise, we say that r and s are* compatible *in the class $\mathcal{IOC}(I,U)$. We say that $\mathcal{S}$ is* input-state-minimal *in the class $\mathcal{IOC}(I,U)$ if and only if any two distinct input-states r, $s \in S_\mathcal{S}$ are distinguishable. We denote by $\mathcal{IOM}(I,U) \subseteq \mathcal{IOIP}(I,U)$ the subclass of all models in $\mathcal{IOIP}(I,U)$ which are also input-state-minimal in the class $\mathcal{IOIP}(I,U)$.*

When there is no explicit mention to the class $\mathcal{IOC}(I,U)$ we understand that it is the same as the class of models $\mathcal{S}$ and $\mathcal{Q}$. Recall Definition 6.1. In [22], implementations are yet further constrained to have at most as many input-states as the specification model. Let $k \geq 1$, and let $\mathcal{IOC}(I,U) \subseteq \mathcal{IO}(I,U)$ be a family of IOLTS models. We denote by $\mathcal{IOC}(I,U,k)$ the subclass of $\mathcal{IOC}(I,U)$ comprised by all models with at most k input-states. The main result in [22], their Theorem 1, shows that for any specification $\mathcal{S}$ in the class $\mathcal{IOM}(I,U)$ it is possible to construct scheme suites that are **ioco**-complete for implementation models in the class $\mathcal{IOM}(I,U,k)$, where k is the number of input states in $\mathcal{S}$. This result also follows easily from Theorem 6.3.

Corollary 6.6. *Let $m \geq 1$, and let $\mathcal{S} \in \mathcal{IOM}(I,U)$ be a specification with $k \geq 0$ input-states. Then we can effectively construct a finite scheme suite SS over $L_I \cup L_U$ which is* **ioco***-complete for $\mathcal{S}$ relatively to the sub-class of $\mathcal{IOM}(I,U,k)[m]$.*

Demonstração. Note that $\mathcal{IOM}(I,U) \subseteq \mathcal{IO}(I,U)$ and $\mathcal{IOM}(I,U,k)[m] \subseteq \mathcal{IO}(I,U)[m]$, and apply Theorem 6.3. □

As we argued in Subsection 5.3 and in Theorem 5.17, we can not, in general, avoid scheme suites to asymptotically grow very large, even when specifications are confined to the class $\mathcal{IOM}(I,U)$, and implementations are restricted to the class $\mathcal{IOM}(I,U,k)$, where k is the number of input-states in $\mathcal{S}$.

Theorem 6.7. *Let $k \geq m \geq 3$, and let $L_I = \{0,1\}$ and $L_U = \{a,x\}$. There is a specification $\mathcal{S} \in \mathcal{IOM}(I,U)$ with k input-states, and for which any scheme suite SS that is* **ioco***-complete for $\mathcal{S}$, relatively to the class $\mathcal{IOM}(I,U,k)[m]$, must be of size $\Omega(\Phi^m)$, where $\Phi = (1+\sqrt{5})/2 \approx 1.61803$.*

Demonstração. We want to use an argument almost exactly as that used in the proof of Theorem 5.17, with a few adjustments to be considered later on.

First, note that the specification $\mathcal{S}$, used in the proof of Theorem 5.17, and depicted in Figure 13, is deterministic, input-complete, progressive and initially-connected, that is, $\mathcal{S} \in \mathcal{IOIP}(I,U)$. Also, the implementation $\mathcal{I}_\alpha$, constructed in that proof and illustrated in Figure 14, is also in the class $\mathcal{IOIP}(I,U)$. The argument, then, proceeds just as in the proof of Theorem 5.17, and we postulate the existence of a scheme $\mathcal{T}_\alpha = \langle S_\alpha, t_0, L_I, L_U, T_\alpha\rangle$ in SS, and such that $\mathcal{I}_\alpha$ does not pass $\mathcal{T}_\alpha$. The proof continues, imitating the argument in the proof of Theorem 5.17, until we reach Eq. (3). At this point we have

$$(t_1, x_1, t_2) \quad \text{and} \quad (t_1, x_2, t_2') \tag{5}$$

in the scheme $\mathcal{T}_\alpha$, with $x_1 \neq x_2$, and $x_1, x_2 \in \{0,1\} = L_I$. Observe that, now, the input alphabet for $\mathcal{T}_\alpha$ is $L_I = \{0,1\}$. Since, according to Definition 6.2, any scheme must be single-input, we need

$x_1 = x_2$, and so we reach a contradiction again. As, before, this will force $\mathcal{T}_\alpha = \mathcal{T}_\beta$ when $\alpha, \beta \in R$, and $\alpha \neq \beta$, with $|\alpha| = |\beta| = r \leq m - 3$. From this point on, the argument follows the one in the proof of Theorem 5.17, establishing that SS must be of size $\Omega(\Phi^m)$.

The proof would be complete if we had $\mathcal{S} \in \mathcal{IOM}(I, U)$ and $\mathcal{I}_\alpha \in \mathcal{IOM}(I, U, k)[m]$, where k is the number of input-states in $\mathcal{S}$. We will now extend Figures 13 and 14 in such a way that these conditions are met, while preserving the validity of the previous argument.

First note that $\mathcal{S}$ has 3 input states, whereas the implementations $\mathcal{I}_\alpha$ has $r + 2 \leq (m - 2) + 2 = m \leq k$ input-states. We then extend the specification in Figure 13 as shown in Figure 15, with states $s_3, \ldots, s_k$. States s_0, s_{k-1} and s_k are repeated to avoid the clutter. Note that the important 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 $\mathcal{S}'$. In order to assert that $\mathcal{S}'$ is in the class $\mathcal{IOM}(I, U)$, we need to verify that any two input-states in Figure 15 are distinguishable. As indicated in Figure 15 states can be partitioned into the two 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 $\ell = k - (j - i)$, so that $2 \leq i \leq \ell \leq k - 1$. This gives $\mathbf{init}(s_\ell) \cap \mathbf{init}(s_k) = \emptyset$ and we conclude that any two distinct states in B_2 are distinguishable. Then, since $(s_0, s_1) \overset{a}{\Rightarrow} (s_k, s_{k-1})$, 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 $w_0 = 0^{k-1}$ and $w_1 = w_0 x$. We get $(s_0, s_i) \overset{w_0}{\Rightarrow} (s_0, s_k)$ and $(s_1, s_i) \overset{w_1}{\Rightarrow} (s_1, s_0)$. Since we already know that s_0 is distinguishable from s_1 and s_k, we conclude that any pair of states in $B_1 \times B_2$ are distinguishable, and we can now state that any two distinct states in $B_1 \cup B_2$ are distinguishable, that is, $\mathcal{S}'$ is input-state-minimal. Moreover, it is clear that $\mathcal{S}'$ is deterministic, input-complete, progressive, initially-connected, and has k input-states. We conclude that $\mathcal{S}' \in \mathcal{IOM}(I, U)$ with k input-states, as desired.

We now turn to the implementation $\mathcal{I}_\alpha$, in Figure 14. Because $\mathbf{init}(q_r) \cap \mathbf{init}(p) = \emptyset$ we see that q_r is distinguishable from any other state $p \neq q_r$ of $\mathcal{I}_\alpha$. Fix some j, $0 \leq j \leq r + 1$, and define $w = \varepsilon$ if $j = r$, $w = \overline{y_r}$ if $j = r + 1$, otherwise let $w = y_{j+1} \cdots y_r$. Clearly, $q_j \overset{w_j}{\Rightarrow} q_r$ and, since **pass** $\overset{w_j}{\Rightarrow}$ **pass**, we get $(q_j, \textbf{pass}) \overset{w_j}{\Rightarrow} (q_r, \textbf{pass})$. Since we already know that q_r and **pass** are distinguishable, 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. Since we already know that q_r is distinguishable from q_j, with no loss of generality, we can take $0 \leq i < j$ and $i \neq r$. We claim that $q_i \overset{w_j}{\Rightarrow}$ **pass** or $q_i \overset{w_j}{\Rightarrow} q_\ell$ with $\ell \leq r - 1$. This, then, would give $(q_i, q_j) \overset{w_j}{\Rightarrow} (p, q_r)$ with $p \neq q_r$, proving that q_i and q_j are also distinguishable. To establish the claim, note that if $i + |w_j| \leq r - 1$ then the result follows immediately from Figure 15, because $\mathbf{init}(q_k) = \{0, 1\}$ for $k = i, i + 1, \ldots, i + |w_j| \leq r - 1$. Assume now that $i + |w_j| > r - 1$. When $j \leq r$, by construction we get $|w_j| = r - j$, and then $i + r - j > r - 1$ implies $i > j - 1$ contradicting $i < j$. When $j = r + 1$ the construction gives $w_j = \overline{y_r}$ and then $|w_j| = 1$, implying $i + 1 > r - 1$, that is $i \geq r - 1$. Because $j = r + 1$, $i < j$ and $i \neq r$, we get $i = r - 1$, and so $q_{r-1} \overset{w_j}{\Rightarrow}$ **pass**, and the claim holds. Putting it together, we conclude that any pair of distinct states of $\mathcal{I}_\alpha$ are distinguishable, that is, $\mathcal{I}_\alpha$ is also input-state-minimal. By inspection of Figure 14 we see that $\mathcal{I}_\alpha$ is deterministic, input-complete, progressive, initially-connected, and has $r + 2 \leq (m - 2) + 2 = m \leq k$ input-states. Hence, $\mathcal{I}_\alpha \in \mathcal{IOM}(I, U, k)$. But $\mathcal{I}_\alpha$ has $r + 3 \leq (m - 3) + 3 = m$ states. Thus, $\mathcal{I}_\alpha \in \mathcal{IOM}(I, U, k)[m]$, completing the proof. □

Theorem 6.7 clearly also applies to any specification $\mathcal{S}$ in which the model depicted in Figure 15 appears as a sub-model with state s_0 being reachable from the initial state of $\mathcal{S}$. This is in contrast to Theorem 4.7 which says that, given a specification $\mathcal{S}$ over an alphabet L, there is an algorithm of asymptotic time complexity $\mathcal{O}(km)$ for checking m-**ioco**-completeness, where $k = n_{\mathcal{S}} n_L$, $n_L = |L|$

and $n_{\mathcal{S}}$ is the number of states in $\mathcal{S}$.

We also remark that in Theorem 1 [22], the authors further restrict implementations to be "input-eager", although they do not precisely define this notion in that text. On the other hand, in none of the proofs in [22] is 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, from the point of view of a tester that is conducting the testing process. Thus, given that the input-eager condition is not strictly necessary to establish Theorem 1 in [22], we conclude that Theorem 6.7 expresses a valid exponential asymptotic lower bound on the size of the test suites claimed by Theorem 1 of [22].

7 Related Works

IOLTS models are largely used to describe the syntax of reactive systems, where input and output actions can occur asynchronously, thus capturing a wide class of systems and communication protocols in practice. 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.

Vries and Tretmans [6] 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. It turns out that such specific test purposes, and their combinations, somewhat limit the fault coverage spectrum. Large, or even infinite, test suites can be produced by their test generation method. Therefore, test selection criteria need to be put in place to avoid this problem, at least when applied in practical situations.

Petrenko et al. [18] studied IOLTS-testing strategies considering implementations that cannot block inputs, and also testers that can never prevent an implementation from producing outputs. This scenario calls for the need for input and output communication buffers, that can be used for the exchange of symbols between the tester and the implementations. This effectively leads to another class of testing strategies, where arbitrarily large buffer memories are allowed.

Tretmans [27] proposed the classic **ioco**-conformance relation for IOLTS models. He developed the foundations of an **ioco**-based -testing theory for IOLTS models, where implementations under test are treated as "black-boxes", a testing architecture where the tester is seen as an artificial environment, driving the exchange of inputs and outputs symbols with the implementation. Some restrictions must be observed, however, by the specification, implementation and tester models, such as input-completeness and output-determinism. Also, the algorithms could lead, in general, to infinite test suites.

Simão and Petrenko [22], in a more recent work, also described an approach to generate finite complete test suites for IOLTS models. They, however, also imposed a number of restrictions upon the specification and the implementation models in order to obtain complete test suites with finite test cases. They assumed test cases are 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.

8 Conclusion

This work addressed the problem of conformance testing for reactive systems using a class of formalisms that allows for the asynchronous occurrence of input and output stimuli. A new notion of conformance relation was studied, one that is more general than the classical **ioco**-conformance relation. The new notion of conformance, opens the possibility for a much wider class of conformance relations, all uniformly treated under the same formalism. As a further advantage, very few restrictions over the structure of the IOLTS models, both specification and implementation, must be satisfied when generating finite and complete test suites. We proved correct a polynomial time algorithm to test general conformance in a "white box" architecture. With a fixed specification, the algorithm runs in linear time on the seize of the IUTs.

Equipped with the new notion of conformance relation, we specialized the test generation process in order to cover other special cases of conformance relations, such as the classical **ioco**-conformance relation. In addition, complexity issues related to the test generation process for more specific conformance relations and their associated test suite generation methods were also discussed. In some cases we showed that the state explosion problem, forcing test suites to grow exponentially with the size of the implementation models, can not be avoided, in general. In these cases, we proved correct algorithms with time complexities that reached such lower bounds. The two distinct families of IOLTSs that were considered suggest that the more general approach can likewise be specialized to treat other families of IOLTSs models of interest, with similar results.

Other areas that might be inspired by ideas presented here might include symbolic test case generation methods, where data variables and parameters are also present in the formalism [20, 9], as well as conformance relations and generation methods for extended IOLTS models that can capture real-time systems [16, 3].

Referências

[1] Saswat Anand, Edmund K. Burke, Tsong Yueh Chen, John Clark, Myra B. Cohen, Wolfgang Grieskamp, Mark Harman, Mary Jean Harrold, Phil McMinn, and others. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, 86(8):1978–2001, 2013.

[2] Adilson L. Bonifacio, Arnaldo V. Moura, and Adenilso Simao. Model Partitions and Compact Test Case Suites. *International Journal of Foundations of Computer Science*, 23(01):147–172, 2012.

[3] Laura Brandán Briones and Ed Brinksma. *A Test Generation Framework for quiescent Real-Time Systems*, pages 64–78. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[4] Rachel Cardell-Oliver. Conformance tests for real-time systems with timed automata specifications. *Formal Aspects of Computing*, 12(5):350–371, 2000.

[5] Steven J. Cunning and Jerzy W. Rozenblit. Automating test generation for discrete event oriented embedded systems. *J. Intell. Robotics Syst.*, 41(2-3):87–112, 2005.

[6] R.G. de Vries and Jean Tretmans. Towards formal test purposes. In G.J. Tretmans and H. Brinksma, editors, *Formal Approaches to Testing of Software 2001 (FATES'01)*, volume NS-01-4 of *BRICS Notes Series*, pages 61–76, Aarhus, Denkmark, August 2001.

[7] Rita Dorofeeva, Khaled El-Fakih, and Nina Yevtushenko. An improved conformance testing method. In Farm Wang, editor, *FORTE*, pages 204–218, 2005.

[8] A. Gargantini. Conformance testing. In M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors, *Model-Based Testing of Reactive Systems: Advanced Lectures*, volume 3472 of *Lecture Notes in Computer Science*, pages 87–111. Springer-Verlag, 2005.

[9] Christophe Gaston, Pascale Le Gall, Nicolas Rapin, and Assia Touil. *Symbolic Execution Techniques for Test Purpose Definition*, pages 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

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

[11] G. Gonenc. A method for the design of fault detection experiments. *IEEE Trans. Comput.*, 19(6):551–558, 1970.

[12] M. A. Harison. *Introduction to Formal Language Theory.* Addison Wesley, 1978.

[13] R. M. Hierons. Separating sequence overlap for automated test sequence generation. *Automated Software Engg.*, 13(2):283–301, 2006.

[14] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Comutation.* Addison Wesley, 1979.

[15] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *Model Checking Software: 11th International SPIN Workshop*, number 2989 in Lecture Notes in Computer Science, pages 109–126, Barcelona, Spain, April 2004.

[16] Moez Krichen. *Model-Based Testing for Real-Time Systems.* PhD thesis, Université Joseph Fourier, 2007.

[17] Brian Nielsen and Arne Skou. Test generation for time critical systems: Tool and case study. *ECRTS*, 00:0155, 2001.

[18] Alexandre Petrenko, Nina Yevtushenko, and Jia Le Huo. Testing transition systems with input and output testers. In *TESTERS, PROC TESTCOM 2003, SOPHIA ANTIPOLIS*, pages 129–145. Springer-Verlag, 2003.

[19] M. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[20] Vlad Rusu, Lydie du Bousquet, and Thierry Jéron. *An Approach to Symbolic Test Generation*, pages 338–357. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[21] Deepinder P. Sidhu and Ting-kau Leung. Formal methods for protocol testing: A detailed study. *IEEE Trans. Softw. Eng.*, 15(4):413–426, 1989.

[22] Adenilso Simão and Alexandre Petrenko. Generating complete and finite test suite for ioco is it possible? In *Ninth Workshop on Model-Based Testing (MBT 2014)*, pages 56–70, 2014.

[23] Gerrit Jan Tretmans. *A formal approach to conformance testing.* PhD thesis, University of Twente, Enschede, December 1992.

[24] J. Tretmans. Testing techniques. Report, University of Twente, The Netherlands, 2004.

[25] Jan Tretmans. Test generation with inputs, outputs, and quiescence. In Tiziana Margaria and Bernhard Steffen, editors, *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings*, volume 1055 of *Lecture Notes in Computer Science*, pages 127–146. Springer, 1996.

[26] Jan Tretmans. Testing concurrent systems: A formal approach. In J.C.M Baeten and S. Mauw, editors, *CONCUR '99: Proceedings of the 10th International Conference on Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 46–65, London, UK, 1999. Springer-Verlag.

[27] Jan Tretmans. Model based testing with labelled transition systems. In *Formal Methods and Testing*, pages 1–38, 2008.