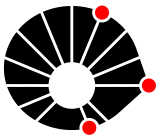

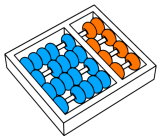

Generating Complete Test Suites for Reactive Systems

Adilson Luiz Bonifacio *Arnaldo Vieira Moura*

Technical Report - IC-17-10 - Relatório Técnico
July - 2017 - Julho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Generating Complete Test Suites for Reactive Systems

Adilson Luiz Bonifacio* Arnaldo Vieira Moura†

Resumo

Model based testing is a well-established approach to test reactive systems described by formal models. In this paper we look at the problem of generating complete test suite for reactive systems formally described as Input Output Labeled Transition Systems (IOLTSs). In this work we propose a new notion of conformance relation for IOLTS models based on formal languages and automata. We show that this new conformance relation is more general than the well-studied **ioco** conformance relation. We then describe how to generate test suites that are sound and exhaustive for a given specification model, according to the new conformance relation. We impose no restrictions on the structure of the specification model, a distinct advantage when compared to other recent works.

1 Introduction

Software testing has always been an important part in many systems development processes, in order to improve the quality of te final product. The use of systematic formal methods and techniques has taken prominence when accuracy and critical guarantees are of paramount importance to the development process when failures can cause severe damages.

Model based testing [6, 15, 17] has been widely used to cope with such difficulties when checking whether an implementation under test (IUT) is in conformance, according to a fault model, to a desired behavior of a specification [1]. Testing approaches based on formal models have the added advantage test suites generation, and establishing completeness guarantees, can be effectively and precisely automated, for certain classes of specification models. Conformance testing [4, 13, 18] is a well-established approach for model based testing that can be used for the validation of reactive systems. Observable behaviors specified by a formal model can be compared with behaviors of a given implementation in order to gain a definite verdict of conformance.

Some formalisms, such as Finite State Machines (FSMs) [2, 3, 5, 7, 9, 11], can capture some aspects of reactive systems. But in this formalism, input actions from the environment and output actions produced by 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) and its close associate class of Input Output Labeled Transition Systems (IOLTSs).

A well-known conformance relation, called **ioco** conformance, has already been proposed for testing IOLTS models [19]. Other 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

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

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

in this direction [16]. 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.

We propose a different and more general notion of conformance relation for testing IOLTS models, one that is based on formal languages and automata. As will be shown, it turns out that the classical **ioco** conformance relation is a special case of this more general notion of conformance. Moreover, with this new notion of conformance, it is possible to generate complete test suites for a much wider class of IOLTS models, thus removing some of the structural restrictions imposed on the models by other approaches.

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 is described in Section 4. Section 5 concludes with some final remarks.

2 Notation and preliminary results

This section defines labeled transition systems (LTSs), the classical theory of finite state automata (FSAs), and also introduces input/output labeled transition systems (IOLTSs). We present some preliminary results stating the relationship between the underlying LTS associated to an IOLTS and the associated FSA. But first we start with the formal models and some notation.

2.1 Basic Notation

Let X and Y be sets. We indicate by $\mathcal{P}(X) = \{Z \mid Z \subseteq X\}$ the power set of X, and $X - Y = \{z \mid z \in X \text{ and } z \notin Y\}$ indicates set difference. We will let $X_Y = X \cup Y$. When $Y = \{y\}$ is a singleton we may also write X_y for $X_{\{y\}}$, that is, $X_y = X \cup \{y\}$. 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. The length of a finite sequence of symbols α over A is indicated by $|\alpha|$. Hence, $|\varepsilon| = 0$. 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. 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 $G_1 \cdot G_2$, or just by $G_1 G_2$, is the language $\{\sigma\rho \mid \sigma \in G_1, \rho \in G_2\}$. 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 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 extend $\widehat{h}$ 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.

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 2.2. *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 the set of* transitions*.*

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

Example 2.3. 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,

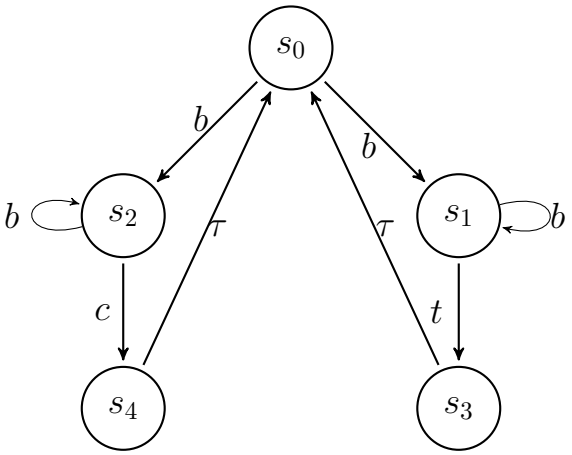

Figura 1: A LTS with finite state and label sets.

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

$$T = \{(s_0, \ell_i, s_{k,}) \,|\, k \geq 1\} \cup \{(s_{k,j-1}, \ell_j, s_{k,j}) \,|\, k \leq 2, 1 \leq j \leq k\}.$$

□

Example 2.4. In Figure 2 we have a LTS $\mathcal{S} = \langle S, s_0, L, T \rangle$ with an infinite set of states $S = \{s_{i,j} : 1 \leq i, 1 \leq j \leq i\}$ and an infinite set of labels is $\{\ell_i : i \geq 1\}$.

The set of transitions is given by the arrows depicted in the figure. □

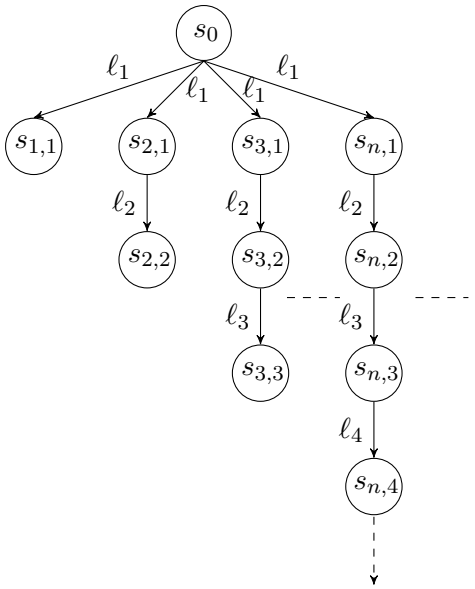

Figura 2: A LTS with infinite states an labels.

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. We first need the notion of a path in a LTS.

Definition 2.5. *Let $\mathcal{S} = \langle S, s_0, L, T\rangle$ be a LTS and let $p, q \in S$. Let $\sigma = l_1, \ldots, l_n$ be a word in $L_\tau^\star$.*

1. *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. *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.

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 $p \stackrel{\sigma}{\to} q$. We may also write $p \stackrel{\sigma}{\to}$ to indicate that there is some $q \in S$ such that $p \stackrel{\sigma}{\to} q$; likewise, $p \to q$ means that there is some $\sigma \in L_\tau^*$ such that $p \stackrel{\sigma}{\to} q$. Also $p \to$ means $p \stackrel{\sigma}{\to} q$ for some $q \in S$ and some $\sigma \in L_\tau^*$. When σ is an observable path from p to q we may write $p \stackrel{\sigma}{\Rightarrow} q$, 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 $p \stackrel{\sigma}{\underset{\mathcal{S}}{\to}} q$, or $p \stackrel{\sigma}{\underset{\mathcal{S}}{\Rightarrow}} q$.

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 of any desired length from s_2 to s_1. □

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 2.7. *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 $tr(p) = \{\sigma \mid p \stackrel{\sigma}{\to}\}$. The set of* observable traces *of p is $otr(p) = \{\sigma \mid p \stackrel{\sigma}{\Rightarrow}\}$.*

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

Remark 2.8. *If* $\mathcal{S}$ *has no* τ*-labeled transitions, then* $otr(\mathcal{S}) = tr(\mathcal{S})$.

Example 2.9. 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 now 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.

Remark 2.10. *In Definition 2.2 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* $s_0 \Rightarrow s$.

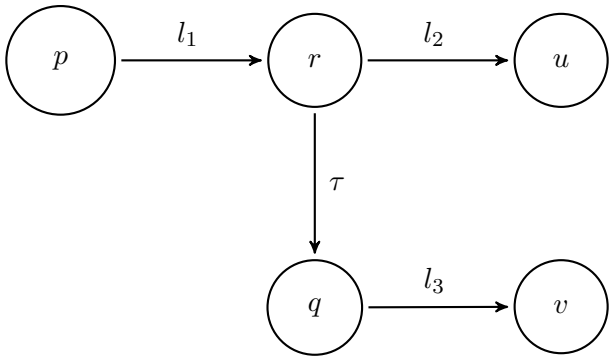

Figura 3: An LTS $\mathcal{S}_1$ with label τ.

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 3. 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 4.

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

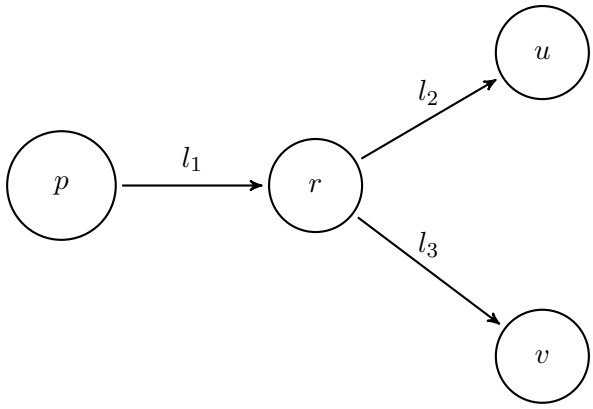

Figura 4: An equivalent LTS to $\mathcal{S}_1$ without τ.

observable behavior leading to two or more distinct states. This motivates a special variant of LTSs.

Definition 2.11. *Let $\mathcal{S} = \langle S, s_0, L, T\rangle$ be a LTS. We say that $\mathcal{S}$ is* deterministic *if $s_0 \stackrel{\sigma}{\Rightarrow} s_1$ and $s_0 \stackrel{\sigma}{\Rightarrow} s_2$ imply $s_1 = s_2$, for all s_1, $s_2 \in S$, and all $\sigma \in L^\star$.*

As a consequence, we have the following result.

Proposition 2.12. *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 $\mathcal{S}$. From Remark 2.10 we get $p \neq q$ and $\sigma \in L^\star$ such that $s_0 \stackrel{\sigma}{\Rightarrow} p$. Then, we also have $s_0 \stackrel{\sigma}{\Rightarrow} q$, contradicting the fact that $\mathcal{S}$ is deterministic, according to Definition 2.11. □

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 2.13. *A* Finite State Automaton with ε-moves *(an ε-FSA, or just a 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 2.14. *Let $\mathcal{A} = \langle S, s_0, A, \rho, F\rangle$ be a ε-FSA. Then we say that $\mathcal{A}$ induces the move relation $\big|\!\!\frac{}{\mathcal{A}}$ defined in the set $S \times A^\star$ of* configurations *of $\mathcal{A}$, where $(p, \sigma) \big|\!\!\frac{}{\mathcal{A}} (q, \mu)$ if either*

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

Since $\big|\!\!\frac{}{\mathcal{A}}$ is defined in the set $S \times A^\star$ we write $\big|\!\!\frac{}{\mathcal{A}}^n$ for its n-th power ($n \geq 0$), and we indicate by $\big|\!\!\frac{}{\mathcal{A}}^\star$ its reflexive and transitive closure, that is $\big|\!\!\frac{}{\mathcal{A}}^\star$ if and only if $\big|\!\!\frac{}{\mathcal{A}}^n$ for some $n \geq 0$. In order to simplify the notation, we may also write $p \overset{\sigma}{\mapsto} q$ when $(p, \sigma) \big|\!\!\frac{}{\mathcal{A}}^\star (q, \varepsilon)$. 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 2.15. *Let $\mathcal{A} = \langle S, s_0, A, \rho, F\rangle$ be an ε-FSA. The* language *accepted by $\mathcal{A}$ is the set*

$$L(\mathcal{A}) = \{\sigma \in A^\star \mid (s_0, \sigma) \big|\!\!\frac{}{\mathcal{A}}^\star (p, \varepsilon) \text{ and } p \in F\}.$$

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

Definition 2.16. *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 2.17. *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 2.18. *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. *$G \cdot H$, $G \cap H$, and $G \cup H$ are regular languages.*
2. *$\overline{G} = A^\star - G$ is a regular language.*
3. *$h(G) \subseteq B^\star$ and $h^{-1}(L) \subseteq A^\star$ are regular languages.*

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

Demonstração. See, *e.g.*, [10, 8]. □

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

Definition 2.19. *Let $\mathcal{A} = \langle S, s_0, A, \rho, F\rangle$ be a FSA. We say that $\mathcal{A}$ is* deterministic *if $\mathcal{A}$ has no ε-moves and 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, if $\mathcal{A} = \langle S, s_0, A, \rho, F\rangle$ 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 [14, 10], 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 2.20. *Let $\mathcal{A} = \langle S_\mathcal{A}, a_0, A, \rho_\mathcal{A}, F_\mathcal{A}\rangle$ be an ε-FSA. Then, there is a deterministic FSA $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$. Further, $\mathcal{B}$ can always be algorithmically constructed from $\mathcal{A}$.*

Demonstração. Consider the sets $E(s) = \{p \mid s \overset{\varepsilon}{\mapsto} p\} \subseteq S$, defined for all $s \in S$, that is $p \in E(s)$ if and only if $(s, \varepsilon) \left|\frac{\star}{\mathcal{A}}\right. (p, \varepsilon)$. 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

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

If $Q_2 \neq \emptyset$, then: (i) include Q_2 in Z_{i+1}, if it is not already 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.

An easy induction of $k \geq 0$ shows that if $(Q_1, \sigma_1) \left|\frac{k}{\mathcal{B}}\right. (Q_2, \sigma_2)$ and $q_2 \in Q_2$, then there is $q_1 \in Q_1$ such that $(q_1, \sigma_1) \left|\frac{\star}{\mathcal{A}}\right. (q_2, \sigma_2)$, for all σ_1, $\sigma_2 \in A^\star$. So, if $\sigma \in L(\mathcal{B})$ then $(b_0, \sigma) \left|\frac{\star}{\mathcal{B}}\right. (Q, \varepsilon)$ 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 $(q, \sigma) \left|\frac{\star}{\mathcal{A}}\right. (f, \varepsilon)$. But $b_0 = E(a_0)$ and so we have $(a_0, \varepsilon) \left|\frac{\star}{\mathcal{A}}\right. (q, \varepsilon)$. Composing, we get $(a_0, \sigma) \left|\frac{\star}{\mathcal{A}}\right. (q, \sigma) \left|\frac{\star}{\mathcal{A}}\right. (f, \varepsilon)$ 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 $(b_0, \varepsilon) \left|\frac{\star}{\mathcal{B}}\right. (Q, \varepsilon)$ 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, $(a_0, \varepsilon) \left|\frac{\star}{\mathcal{A}}\right. (f, \varepsilon)$, and so $\varepsilon \in L(\mathcal{A})$ because $f \in F_\mathcal{A}$. Next, let $\sigma \in A^\star$, with $|\sigma| \geq 1$ and $(q_1\sigma) \left|\frac{\star}{\mathcal{A}}\right. (q_2, \varepsilon)$. A simple induction on $|\sigma|$ show 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 $(Q_1, \sigma) \left|\frac{\star}{\mathcal{B}}\right. (Q_2, \varepsilon)$. Now, if $|\sigma| \geq 1$ and $\sigma \in L(\mathcal{A})$ we get $(a_0, \sigma) \left|\frac{\star}{\mathcal{A}}\right. (f, \varepsilon)$, 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 $(b_0, \sigma) \left|\frac{\star}{\mathcal{B}}\right. (Q, \varepsilon)$. 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. □

It will be useful to examine the construction when 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 $\mathcal{B}$. For later use, we write this conclusion as a remark.

Remark 2.21. *In Proposition 2.20, if we start with a FSA for which all states are final states, then in the equivalent FSA there constructed all states will be final states too.*

From a deterministic FSA we can 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 $S_{\mathcal{B}} = S_A \cup \{e\}$, 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 $\sigma \in A^\star$. □

The following simple result 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 $|S_{\mathcal{T}}| \leq |S_{\mathcal{A}}| \times |S_{\mathcal{B}}|$.

Demonstração. Define also $t_0 = (a_0, b_0)$, 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 $a_0 \left|\frac{\star}{\mathcal{A}}\right. s_1$ and $b_0 \left|\frac{\star}{\mathcal{A}}\right. s_2$ for some $\sigma \in A^\star$. Clearly, $|S_{\mathcal{T}}| \leq |S_{\mathcal{A}}| \times |S_{\mathcal{B}}|$. Now, 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 $|\sigma| \geq 0$ gives $\big((s_1, q_1), \sigma\big) \left|\frac{\sigma}{\mathcal{T}}\right. \big((s_2, q_2), \varepsilon\big)$ if and only if $(s_1, \sigma) \left|\frac{\sigma}{\mathcal{A}}\right. (s_2, \varepsilon)$ and $(q_1, \sigma) \left|\frac{\sigma}{\mathcal{B}}\right. (q_2, \varepsilon)$, for all $\sigma \in A^\star$.

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

The following result will prove very useful later on.

Lemma 2.24. *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 $A_2 \cap C = \{\sigma\} \not\subseteq A_1$. □

Example 2.25. Let $A = \{a, b\}$ be an alphabet and let $A_1 = (aa + bb)^\star$, $A_2 = (aa)^\star b^\star$ be regular languages over A. Now let $B = a^\star(bb)^\star$ be a regular language over A and also let $C \subseteq B$ be a finite language with $C = \{\epsilon, a, aa, bb, abb, aabb, abbbb, aaabb\}$. We can check that

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

One can see that $B' = A_2 \cap B = (aa)^\star(bb)^\star$. Clearly $B' \subseteq A_1$ when pairs of a's come before pairs of b's. On the other hand, $C' = A_2 \cap C = \{\epsilon, aa, bb, aabb\}$, which easily gives $C' \subseteq A_1$. Note that if $abbb \in C'$ then $abbb$ should be in B which is not true. Otherwise we will get, in this case, $C \not\subseteq B$. □

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 2.26. *We have the following two associations:*

1. *Let $\mathcal{S} = \langle S, s_0, L, T\rangle$ be a LTS. The ε-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 an ε-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 2.27. *Let $\mathcal{S} = \langle S, s_0, L, T\rangle$ be a LTS and let $\mathcal{A}_\mathcal{S}$ be the ε-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 an ε-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 2.7, 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 2.16, 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 2.28. *Let $\mathcal{S}$ be an 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 ε-FSA $\mathcal{A}_{\mathcal{S}}$ induced by $\mathcal{S}$. From Proposition 2.27 we know that $L(\mathcal{A}_{\mathcal{S}}) = otr(\mathcal{S})$. Use Proposition 2.20 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 2.26 we know that all states of $\mathcal{A}_{\mathcal{S}}$ are final states and so, from Remark 2.21 it follows that the same is true of $\mathcal{B}$. Thus, from Definition 2.26 again, we can get the LTS $\mathcal{S}_{\mathcal{B}}$, induced by $\mathcal{B}$. By Proposition 2.27 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 2.26, 2.19, and 2.11 that $\mathcal{S}_{\mathcal{B}}$ is also deterministic. Hence, $\mathcal{S}_{\mathcal{B}}$ has no τ-labeled transitions, and so Remark 2.8 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 2.20 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 2.28, then $p \stackrel{\sigma}{\Rightarrow} q$ in $\mathcal{S}$ if and only if $p \stackrel{\sigma}{\rightarrow} q$ in $\mathcal{T}$ if and only if $p \stackrel{\sigma}{\Rightarrow} q$ in $\mathcal{T}$. Moreover, $\mathcal{T}$ has no τ-labeled transitions, and if $p \stackrel{\sigma}{\Rightarrow} q$ 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 symbols as symbols that the LTS "receives" from the environment, and some other set of action symbols as symbols that the LTS "sends back" to the environment. The next LTS variation differentiates between input action symbols and output action symbols.

Definition 2.29. *An Input Output Labeled Transition System (IOLTS) $\mathcal{I}$ is defined by a tuple $\langle S, s_0, L_I, L_U, T\rangle$, 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 an 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 $\mathcal{IOLTS}(L_I, L_U)$. In order to ease the notation, in specific examples a symbol in L_I will always be denoted by following it with a "?" mark, and a symbol in L_U will 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.*, [19], 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 2.30. *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 Meally machine. 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 a IOLTS model, input labels and output labels are exchanged in an asynchronous way. This facilitates the specification of more complex behaviors.

Example 2.31. Figure 5 represents an IOLTS, adapted from [12]. The model represents the

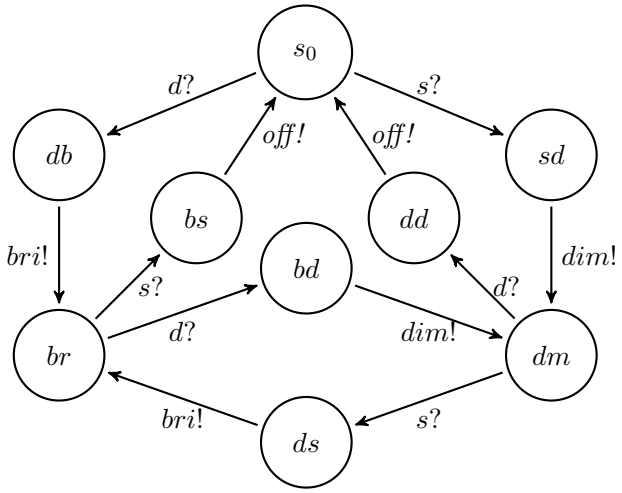

Figura 5: An IOTS.

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 2.32. Figure 6 represents another IOLTS, adapted from [19]. 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. □

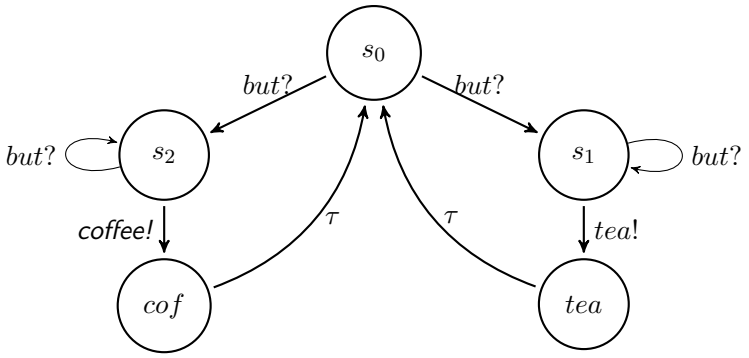

Figura 6: Another IOTS.

2.5 Quiescent States

When treating the **ioco** conformance relation, some works [19] distinguish the so called quiescent states. Informally, those are states from which there are no transitions labeled by some output action symbol.

Definition 2.33. *Let $\mathcal{S} = \langle S, s_0, L, T\rangle$ be a LTS. A state $s \in S$ is said to be* quiescent *if $out(s) = \emptyset$.*

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 "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 $\mathcal{T} = \langle Q, q_0, L_U, L_I, R\rangle$.

However, 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 [19] is to imagine that the implementation has a special output symbol $\delta \notin L_I \cup L_U$, 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 $q \xrightarrow{\delta} q'$ between specific states q, $q' \in Q$[1]. 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.

[1] In [19], a different symbol, θ, was used to signal the acceptance of quiescence on the tester side, but that for our formalism, that makes little difference, if any.

Remark 2.34. *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 conformance relations using regular languages. We also relate such a generalization with the classical notion of ioco conformance relation [19].

3.1 The General Conformance Relation

Informally, we consider a language D, the set of "desirable", or "allowed", behaviors, and a language F, the set of "forbidden", or "undesirable", behaviors. 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 os 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, $\overline{otr}(\mathcal{S}) = L^\star - otr(\mathcal{S})$.

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 and $otr(\mathcal{I}) \cap F \cap otr(\mathcal{S}) = \emptyset$ and $otr(\mathcal{I}) \cap D \cap \overline{otr}(\mathcal{S}) = \emptyset$. And this holds if and only if

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

□

Example 3.3. Let $\mathcal{S}$ be a specification depicted in Figure 7 and also let $\mathcal{I}$ be an implementation as depicted in Figure 9 with $L = \{a, b, x\}$. Take the regular languages $D = (a + b)^\star ax$ and $F = (a + b)^\star bx$. We check that $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$.

The condition $F \cap otr(\mathcal{S}) = \emptyset$ since F represents words that should not be accepted by $\mathcal{S}$. In this case any word ending with bx. On the other hand, $\overline{otr}(\mathcal{S})$ is the language that is accepted bye the FSA $\overline{\mathcal{S}}$ such that $L(\overline{\mathcal{S}}) = \overline{otr}(\mathcal{S})$. For instance, we have that $x, bx, bax, axx, bbx, abx, abax \in L(\overline{\mathcal{S}})$. We illustrate the FSA $\overline{\mathcal{S}}$ at Figure 8.

Next we see that the language D contains any word of $L^\star$ ending with ax. Now we need to find out the condition $D \cap \overline{otr}(\mathcal{S})$. By a simple inspection we can obtain that any word $\alpha = \sigma ax$ such that $\bar{s_0} \xrightarrow{\sigma} \bar{s_3}$ belongs to the $D \cap \overline{otr}(\mathcal{S})$. For instance, we note that $bax, abax \in D \cap \overline{otr}(\mathcal{S})$. But we easily see that the words $bax, abax$ are also words of language $L(\mathcal{I})$. Then 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.

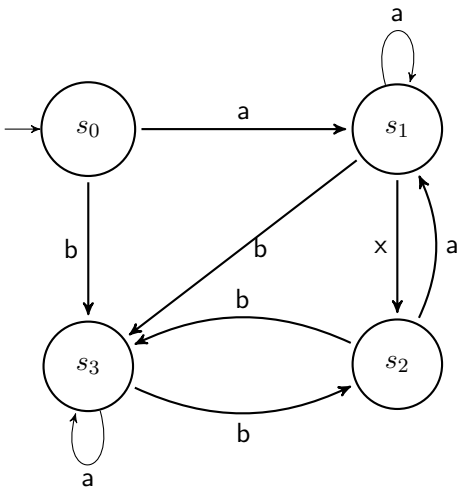

Figura 7: LTS specification $\mathcal{S}$.

On the other hand, if we assume an implementation $\mathcal{I}$ such that $\mathcal{I}$ is isomorphic to $\mathcal{S}$, $\mathcal{I}$ would not have the transition $q_3 \xrightarrow{x} q_2$ and $bax, abax \in D \cap \overline{otr}(\mathcal{S})$ would not be accepted by $\mathcal{I}$. Actually, we notice that $otr(\mathcal{I}) \cap D \cap \overline{otr}(\mathcal{S}) = \emptyset$, and in this case the condition holds. So we get that $otr(\mathcal{I}) \cap \left[(D \cap \overline{otr}(\mathcal{S})) \cup (F \cap otr(\mathcal{S}))\right] = \emptyset$ therefore $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ as expected. □

Depending on 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 $s_0 \stackrel{\sigma}{\Rightarrow} q$ 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 $D = L^\star \cdot H$.

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

We can now show that if an implementation $\mathcal{I}$ conforms to a specification $\mathcal{S}$ according to 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 2.24 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, we now get $\mathcal{I}\,\mathbf{conf}_{D',F'}\,\mathcal{S}$, as desired. □

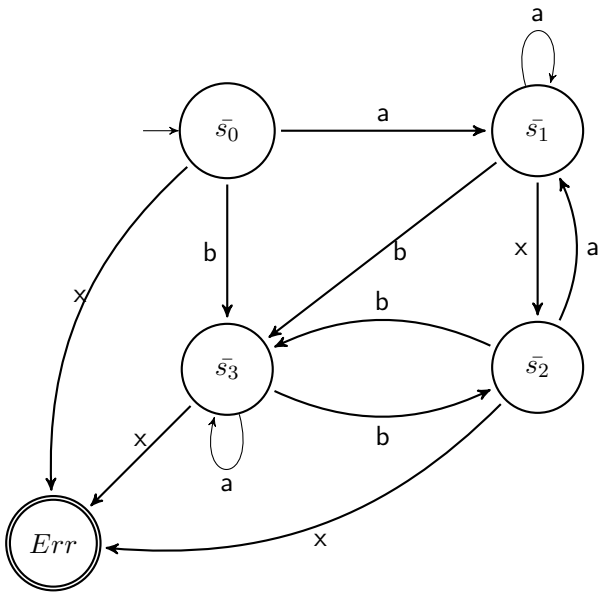

Figura 8: FSA complement $\overline{\mathcal{S}}$ of LTS specification $\mathcal{S}$.

Example 3.5. Let $\mathcal{S}$ and $\mathcal{I}$ be a specification and an implementation, again as depicted in Figures 7 and 9, respectively. Also assume the same languages $D = (a+b)^\star ax$ and $F = (a+b)^\star bx$. Now let

$$D' = \{ax, bax, abax, aaax, abbax, ababax, bbax, babax, bababax\}$$
$$F' = \{bx, aabx, abbbxababbx, bbbx, babbx\}$$

Clearly, $D' \subseteq D$ and $F' \subseteq F$. From Example 3.3we know that had $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ does not hold. We check that $\mathcal{I}\,\mathbf{conf}_{D',F'}\,\mathcal{S}$ does not hold also. First, we get that $D' \cap \overline{otr}(\mathcal{S}) = \{bax, abax, bababax\}$. Since $F \cap otr(\mathcal{S}) = \emptyset$ we also have $F' \cap otr(\mathcal{S}) = \emptyset$. Next we obtain $otr(\mathcal{I}) \cap \big[(D' \cap \overline{otr}(\mathcal{S})) \cup (F' \cap otr(\mathcal{S}))\big] = \{bax, abax, bababa\}$, and so, by Proposition 3.2, $\mathcal{I}\,\mathbf{conf}_{D',F'}\,\mathcal{S}$ does not hold as desired.

On the other hand, if we consider an implementation $\mathcal{I}$ that is isomorphic to $\mathcal{S}$, then it is clear that $otr(\mathcal{I}) \cap \big[(D \cap \overline{otr}(\mathcal{S})) \cup (F \cap otr(\mathcal{S}))\big] = F \cap otr(S)$. Likewise, $otr(\mathcal{I}) \cap \big[(D' \cap \overline{otr}(\mathcal{S})) \cup (F' \cap otr(\mathcal{S}))\big] = F' \cap otr(S)$. Since, $F \cap otr(\mathcal{S}) = F' \cap otr(\mathcal{S}) = \emptyset$, we conclude that both $\mathcal{I}\,\mathbf{conf}_{D',F'}\,\mathcal{S}$ and $\mathcal{I}\,\mathbf{conf}_{D,F}\,\mathcal{S}$ hold. □

3.2 The ioco Conformance Relation

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 classical **ioco** conformance relation [19] 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. We formalize these ideas next.

Definition 3.6 ([19])**.** *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* **after** $: S \times L^\star \to \mathcal{P}(S)$ *by letting* s **after** $\sigma = \{q \mid s \stackrel{\sigma}{\Rightarrow} q\}$*, for all* $s \in S$*,* $\sigma \in otr(\mathcal{S})$*;*

2. *Define a function out* $: \mathcal{P}(S) \to L_U$ *as* $out(V) = \bigcup_{s \in V} \{\ell \in L_U \mid s \stackrel{\ell}{\to}\}$*, for all* $V \subseteq S$*;*

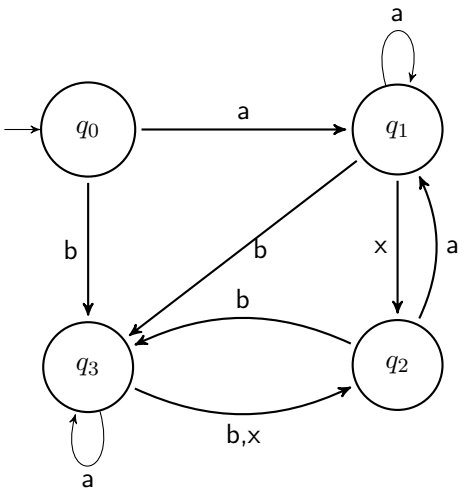

Figura 9: LTS implementation $\mathcal{I}$.

3. *Define a relation* **ioco** $\subseteq \mathcal{IOLTS}(L_I, L_U) \times \mathcal{IOLTS}(L_I, L_U)$ *by letting* $\mathcal{I}$ **ioco** $\mathcal{S}$ *if and only if for all* $\sigma \in otr(\mathcal{S})$ *we have*

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

Again, we may abuse the notation by writing $out(S)$ instead of $out(\{s\})$.

3.3 The ioco relation as a special case

Now we want to show that the **ioco** conformance relation is a special case of Definition 3.1.

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* $D = otr(\mathcal{S}) \cdot L_U$ *is a regular language over* L*, and we have that* $\mathcal{I}$ **ioco** $\mathcal{S}$ *if and only if* $\mathcal{I}\,\textbf{conf}_{D,\emptyset}\,\mathcal{S}$.

Demonstração. From Definition 2.30 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 2.27 and Remark 2.17 we see that $otr(\mathcal{S})$ and L_U are regular languages. Hence, by Proposition 2.18 we conclude that D is also a regular language.

Now, we show that $\mathcal{I}$ **ioco** $\mathcal{S}$ if and only if $\mathcal{I}\,\textbf{conf}_{D,F}\,\mathcal{S}$. First assume that we have $\mathcal{I}\,\textbf{conf}_{D,F}\,\mathcal{S}$. 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 $\mathcal{I}$ **ioco** $\mathcal{S}$, let $\sigma \in otr(\mathcal{S})$ and let $\ell \in out(q_0 \textbf{ after } \sigma)$. We must show that $\ell \in out(s_0 \textbf{ after } \sigma)$. Because $\ell \in out(q_0 \textbf{ after } \sigma)$ we get $\sigma, \sigma\ell \in otr(\mathcal{I})$. Since $\ell \in L_U$, we get $\sigma\ell \in otr(\mathcal{S}) \cdot L_U$ 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, $\ell \in out(s_0 \textbf{ after } \sigma)$, as desired.

Next, assume that $\mathcal{I}$ **ioco** $\mathcal{S}$ and we want to show that $\mathcal{I}\,\textbf{conf}_{D,F}\,\mathcal{S}$ holds. Given that $F = \emptyset$, we immediately have $otr(\mathcal{I}) \cap F \cap otr(\mathcal{S}) = \emptyset$, thus verifying the first condition of Definition 3.1. 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 $D = otr(\mathcal{S}) \cdot L_U$. Clearly, τ does not occur in σ. Also, $\sigma \in otr(\mathcal{I})$ and $\ell \neq \tau$ give $\alpha\ell \in otr(\mathcal{I})$, and so $\alpha \in otr(\mathcal{I})$. Then, because $\ell \in L_U$, we get $\ell \in out(q_0 \textbf{ after } \alpha)$. Because we assumed $\mathcal{I}$ **ioco** $\mathcal{S}$ and we have $\alpha \in otr(\mathcal{S})$,

we also get $\ell \in out(s_0 \textbf{ after } \alpha)$, 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 7 and 9, respectively.

Assume $L_I = \{a, b\}$, $L_U = \{x\}$. Since we need $D = otr(\mathcal{S}) \cdot L_U$ we construct a FSA $\mathcal{D}$ such that $L(\mathcal{D}) = D$ as depicted in Figure 10.

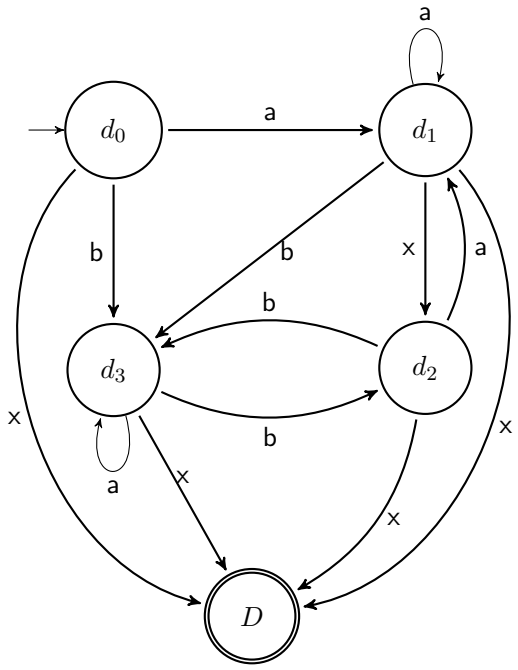

Figura 10: FSA $\mathcal{D}$ for the language $D = otr(\mathcal{S}) \cdot L_U$.

To show that $\mathcal{I}$ **ioco** $\mathcal{S}$ take $\sigma = aba$ and $x \in out(q_0 \textbf{ after } \sigma)$. By a simple inspection we see that $\sigma, \sigma x \in otr(\mathcal{I})$, and since $x \in L_U$ we also get $\sigma x \in otr(\mathcal{S}) \cdot L_U$ and then $\sigma x \in D$. Next we that $\sigma x \in otr(\mathcal{I}) \cap D$, but $\sigma x \notin otr(\mathcal{S})$. We then conclude that $x \notin out(s_0 \textbf{ after } \sigma)$, as required since $otr(\mathcal{I}) \cap D \not\subseteq otr(\mathcal{S})$.

For the sake of completeness, assume an implementation isomorphic to $\mathcal{S}$. Now take $\sigma = bba$ and $x \in out(q_0 \textbf{ after } \sigma)$. By the same previous reasoning we conclude that $x \in out(s_0 \textbf{ after } \sigma)$, as desired since $otr(\mathcal{I}) \cap D \subseteq otr(\mathcal{S})$. Note that the output x is unique in $\mathcal{S}$, so for any σ such that $s_0 \xrightarrow{\sigma} s_1$ with $s_1 \xrightarrow{x}$ we will show that $\mathcal{I}$ **ioco** $\mathcal{S}$.

Now assume $\mathcal{I}$ **ioco** $\mathcal{S}$. We need to show that $\mathcal{I}\,\mathbf{conf}_{D,\emptyset}\,\mathcal{S}$. We take again $\sigma = aba$ such that $\sigma x \in D$ according to $D = otr(\mathcal{S}) \cdot L_U$. We also see that $\sigma x \in otr(\mathcal{I})$, but $\sigma x \notin otr(\mathcal{S})$. Next, because $x \in out(q_0 \textbf{ after } \sigma)$, we should have $\mathcal{I}$ **ioco** $\mathcal{S}$ and $\sigma \in otr(\mathcal{S})$ in order to get $x \in out(s_0 \textbf{ after } \sigma)$ which gives $\beta \in otr(\mathcal{S})$. But we conclude $x \notin out(s_0 \textbf{ after } \sigma)$ and then $otr(\mathcal{I}) \cap D \not\subseteq otr(\mathcal{S})$ as expected.

Again, for the sake of completeness, assume an implementation isomorphic to $\mathcal{S}$. Take $\sigma = bba$ such that $\sigma x \in D$ according to $D = otr(\mathcal{S}) \cdot L_U$. By the previous reasoning, in this case, we have that $\sigma x \in otr(\mathcal{I})$ and $\sigma x \in otr(\mathcal{S})$. We then conclude the $x \in out(s_0 \textbf{ after } \sigma)$ and $otr(\mathcal{I}) \cap D \subseteq otr(\mathcal{S})$ as desired. Again, x is unique in $\mathcal{S}$, and for any σ such that $s_0 \xrightarrow{\sigma} s_1$ with $s_1 \xrightarrow{x}$ we will get that $\mathcal{I}\,\mathbf{conf}_{D,\emptyset}\,\mathcal{S}$. □

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

Corollary 3.9. *Let $\mathcal{S} = \langle S, s_0, L, T\rangle$ be a specification LTS and let $\mathcal{I} = \langle Q, q_0, L, R\rangle$ be an implementation LTS. Then there is a finite language $D \subseteq L^\star$ 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 $T = \overline{otr}(\mathcal{S}) \cap [otr(\mathcal{S}) \cdot L_U]$*

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

In this section we define test suites as sets of words. We can then use test suites to decide implementation conformance with respect to a given specification.

4.1 The Notion of a Test Suite

The general definition of a test suite follows.

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

A test suite T should be geared to detect bad observable behaviors in an implementation, 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 under test.

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, a 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. Further, 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 over L, and let D, $F \subseteq L^\star$ be a pair of conformance languages over L. Then, the set*

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

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 then 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, $\sigma \in otr(\mathcal{I})$. □

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.

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, it suffices to check that we have $otr(\mathcal{I}) \cap T = \emptyset$. Using Proposition 2.27 we see that $otr(\mathcal{S})$ is also a regular language, and so, by Proposition 2.18 so is $\overline{otr}(\mathcal{S})$. Thus, if D and F are regular languages, then Proposition 2.18 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 2.27 and 2.18 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 $\mathcal{S}$.

5 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** relation. The new notion was based on formal languages and finite state automata.

Based on this new notion of conformance, the problem of generating complete test suites for a given specification was studied. It was shown how to generate complete test suites for a wide class of IOLTS models. As a further advantage, very few restrictions over the structure of the IOLTS models must be satisfied when generating complete test suites.

Equipped with the new notion of conformance relation here studied, it should be possible to specialize the test generation process to cover other special cases of conformance relations, such as the classical **ioco** conformance relation. In addition, complexity issues related to the generation process for more specific conformance relations could also be addressed.

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

[4] 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.

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

[6] 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.

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

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

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

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

[11] 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.

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

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

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

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

[16] 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.

[17] 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.

[18] 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.

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