Refusal testing for classes of transition systems with inputs and outputs

Lex Heerink and Jan Tretmans
Tele-Informatics and Open Systems group, Dept. of Computer Science
University of Twente, 7500 AE Enschede, The Netherlands
{heerink,tretmans}@cs.utwente.nl

Abstract

This paper presents a testing theory that is parameterised with assumptions about the way implementations communicate with their environment. In this way some existing testing theories, such as refusal testing for labelled transition systems and (repetitive) quiescence testing for I/O automata, can be unified in a single framework. Starting point is the theory of refusal testing. We apply this theory to classes of implementations which communicate with their environment via clearly distinguishable input and output actions. These classes are induced by making assumptions about the geographical distribution of the points of control and observation (PCO's) and about the way input actions of implementations are enabled. For specific instances of these classes our theory collapses with some well-known ones. For all these classes a single test generation algorithm is presented that is able to derive sound and complete test suites from a specification.

1 INTRODUCTION

An important aspect in the design and construction of systems is to validate whether an implementation operates as it has been specified. This can be done using conformance testing: experiments are conducted on an implementation under test (IUT) and from the observation of responses of the IUT it is concluded whether it behaves correctly. A formalisation of the conformance testing process hence requires formal models for the specification, for the implementation under test, and for experiments and observations, and the

formal definition of a correctness criterion, which is done by means of an implementation relation between models of implementations and specifications.

In formal conformance testing it is assumed that the formal specification is apriori known, and that the behaviour of an implementation can be formally modelled, but its model is not apriori known. The latter is called a test assumption. A well-known test assumption is that implementations can be modelled as labelled transition systems [2, 3, 14] that communicate in a symmetric and synchronous way with their environment; no notion of initiative of actions is present. However, it has been recognised that such symmetric communication is not very realistic. Most implementations communicate in practice with their environment via actions that are clearly initiated by one partner, and accepted by the other [9, 11, 13, 14]. This has triggered research in models that make an explicit distinction between actions that are controlled by the environment (input actions of the implementation) and actions that are controlled by the IUT (output actions of the implementation), e.g., input/output automata (IOA) [9, 13], input/output state machines [11], and input/output transition systems (IOTS) [15]. Many of these models additionally require that input actions are continuously enabled.

As indicated in [4, 15] many implementation relations for labelled transition systems can be defined extensionally in terms of a set of experimenters $\mathcal{U}$, a set of observations $obs(u, i)$ that experimenter $u \in \mathcal{U}$ causes when system i is tested, and a relation $\otimes$ between $obs(u, i)$ and $obs(u, s)$. Formally, this amounts to

$$i \textbf{ conforms-to } s \;=_{def}\; \forall u \in \mathcal{U} : obs(u, i) \otimes obs(u, s) \qquad (1)$$

One such testing relation is refusal testing [12], where experimenters are not only able to detect whether actions can occur, but also able to detect whether actions can fail. Another example is the extensional characterisation of quiescent trace preorder for input/output automata in [13].

This paper continues the track of extensionally defined implementation relations and testing for models that distinguish between input and output actions. What we add in this paper is the distinction between the different *locations* where these actions may occur on an interface, that is, we explicitly take the distributed nature of interfaces into account. Moreover, we weaken the requirement imposed on IOA and IOTS that input actions for implementations must always be enabled, thereby conciliating the criticism in [13] that this requirement is too restrictive. We obtain classes of models of implementations, one for each possible distribution of the interface. For these classes we apply refusal testing [12] where observers are able to observe the success and failure of actions conducted at each different location separately. We will show that refusal testing for transition systems without inputs and outputs [12] and refusal testing for IOTS [15] are just instances of our parameterised model (for the finest and the coarsest distribution of locations, respectively). In that way the worlds of testing transition systems with, and without, inputs and outputs are unified in a single testing framework. Furthermore, we define

an intuitive correctness criterion in the same way as [15] which is manageable for test generation in realistic situations. A single test generation algorithm is given which can cope with each of these classes, and which derives tests that can distinguish between correct and incorrect implementations.

This paper is organised as follows. Section 2 fixes notation and recapitulates refusal testing for labelled transition systems [12]. In section 3 classes of implementation models, parameterised with assumptions about the distribution of the interfaces, are defined and co-related. Refusal testing theory for these classes is discussed in section 4 in a single framework. Section 5 presents a test generation algorithm which is proved to be sound and complete. Section 6 illustrates the operation of the algorithm, and section 7 wraps up with conclusions and further work.

2 REFUSAL TESTING FOR TRANSITION SYSTEMS

We use labelled transition systems to specify and model the behaviour of systems. This section recalls the basics of transition systems and refusal testing without distinguishing between inputs and outputs.

DEFINITION 1 *A (labelled) transition system over L is a quadruple $\langle S, L, \rightarrow, s_0\rangle$ where S is a (countable) set of states, L is a (countable) set of observable actions, $\rightarrow \subseteq S \times L \times S$ is a set of transitions, and $s_0 \in S$ is the initial state.*

We denote the class of all transition systems over L by $\mathcal{LTS}(L)$. For the notation of transition systems, we use some standard process-algebraic operators, which are defined in the usual way (cf. LOTOS [5]). For this paper it suffices to use action-prefix $\alpha; B$, which can perform action α and then behave as B, and unguarded choice $\sum \mathcal{B}$ which can behave as any of its members $B \in \mathcal{B}$. We abbreviate $\sum\{B_1, B_2\}$ by $B_1 + B_2$ and $\sum \emptyset$ by **stop**.

The behaviour of a labelled transition system, starting from a particular state (usually s_0), is expressed using sequences consisting of actions and sets of refused actions, i.e., sequences in $(\mathcal{P}(L) \cup L)^*$ ($\mathcal{P}(L)$ denotes the set of all subsets of L). Such sequences are called *failure traces* [1].

NOTATION 1 Let $p = \langle S, L, \rightarrow, s_0\rangle$ be a transition system such that $s, s' \in S$, $\alpha, \alpha_i \in \mathcal{P}(L) \cup L$ and $\sigma \in (\mathcal{P}(L) \cup L)^*$. Then

$$s \xrightarrow{\alpha} s' \quad =_{def} \begin{cases} (s, \alpha, s') \in \rightarrow & \text{if } \alpha \in L \\ s = s' \text{ and } \forall \mu \in \alpha : s \not\xrightarrow{\mu} & \text{if } \alpha \in \mathcal{P}(L) \end{cases}$$

$$s \xrightarrow{\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_n} s' \quad =_{def} \exists s_0, s_1, \ldots, s_n : s = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} s_n = s'$$

$$s \xrightarrow{\sigma} \quad =_{def} \exists s' : s \xrightarrow{\sigma} s'$$

The self-loop transitions of the form $s \xrightarrow{A} s$ where $A \subseteq L$ are called refusal transitions; A is called a refusal of s. Such a refusal transition explicitly encodes the inability to perform any action in A from state s. Refusal transitions can be serialised: $s \xrightarrow{A_1 \cup A_2} s$ iff $s \xrightarrow{A_1} s \xrightarrow{A_2} s$.

DEFINITION 2 *Let $p \in \mathcal{LTS}(L)$, then*
1. $init(p) =_{def} \{\alpha \in L \mid \exists p' : p \xrightarrow{\alpha} p'\}$
2. $f\text{-}traces(p) =_{def} \{\sigma \in (\mathcal{P}(L) \cup L)^* \mid p \xrightarrow{\sigma}\}$
3. $traces(p) =_{def} f\text{-}traces(p) \cap L^*$
4. p **after** σ **deadlocks** $=_{def} \exists p' : p \xrightarrow{\sigma} p'$ *and* $init(p') \cap L = \emptyset$
5. $der(p) =_{def} \{p' \mid \exists \sigma \in L^* : p \xrightarrow{\sigma} p'\}$
6. P **after** $\sigma =_{def} \{p' \mid \exists p \in P : p \xrightarrow{\sigma} p'\}$ *where P is a set of states*
7. *p is deterministic iff* $\forall \sigma \in L^* : |\{p\}$ **after** $\sigma| \leq 1$
8. *p has finite behaviour iff* $\exists N \in \mathbb{N} : \forall \sigma \in traces(p) : |\sigma| \leq N$

Observers are, just like specifications and implementations, modelled as transition systems. In order for an observer to observe the refusals of a system p we equip an observer u with a special deadlock detection label θ ($\theta \notin L$) [8]. The occurrence of θ indicates that the synchronised behaviour of u and p is not able to continue with any other action than θ, i.e., p refuses all other actions offered by u.

DEFINITION 3 *Let $p \in \mathcal{LTS}(L)$ and $u \in \mathcal{LTS}(L \cup \{\theta\})$, then $\rceil|: \mathcal{LTS}(L \cup \{\theta\}) \times \mathcal{LTS}(L) \to \mathcal{LTS}(L \cup \{\theta\})$ is defined by the following inference rules.*

$$\frac{u \xrightarrow{a} u', p \xrightarrow{a} p'}{u \rceil| p \xrightarrow{a} u' \rceil| p'} \ (a \in L) \qquad \frac{u \xrightarrow{\theta} u', init(u) \cap init(p) = \emptyset}{u \rceil| p \xrightarrow{\theta} u' \rceil| p}$$

Observations that can be made using an observer u interacting with p by means of $\rceil|$ now may include the action θ. This makes it is possible to detect when p was unable to perform any other action offered by u.

DEFINITION 4 *Let $p \in \mathcal{LTS}(L)$ and $u \in \mathcal{LTS}(L \cup \{\theta\})$.*
1. *The set of completed trace observations obs_c^θ is*

$$obs_c^\theta(u,p) =_{def} \{\sigma \in (L \cup \{\theta\})^* \mid (u \rceil| p) \textbf{ after } \sigma \textbf{ deadlocks}\}$$

2. *The set of trace observations obs_t^θ is*

$$obs_t^\theta(u,p) =_{def} \{\sigma \in (L \cup \{\theta\})^* \mid (u \rceil| p) \xrightarrow{\sigma}\}$$

Based on the ability to distinguish processes by means of observations a preorder on processes can be defined extensionally (cf. equation (1)). This preorder, called *refusal preorder*, is known to correspond to inclusion of failure traces [12].

DEFINITION 5 *Refusal preorder $\leq_{rf} \subseteq \mathcal{LTS}(L) \times \mathcal{LTS}(L)$ is defined by*

$$i \leq_{rf} s =_{def} \forall u \in \mathcal{LTS}(L \cup \{\theta\}) : \begin{array}{l} obs_c^\theta(u,i) \subseteq obs_c^\theta(u,s) \text{ and} \\ obs_t^\theta(u,i) \subseteq obs_t^\theta(u,s) \end{array}$$

PROPOSITION 1 $i \leq_{rf} s$ *iff* $f\text{-}traces(i) \subseteq f\text{-}traces(s)$

3 CLASSES OF TRANSITION SYSTEMS

In [9] it is argued that the symmetric synchronisation mechanism between a system and its environment used in e.g., [5] exhibits the counter intuitive property that the system is able to block actions that are supposed to be controlled by its environment, and vice versa. Therefore, models have been developed that distinguish between the initiative (or direction) of actions. In these models either an action is initiated by the environment and accepted by the system (i.e., it is an *input action*), or an action is initiated by the system and accepted by its environment (i.e., it is an *output action*). Output actions for the environment are input actions for the system, and vice versa. By requiring that implementations must always be prepared to accept input actions and the environment must always be prepared to accept output actions, counter-intuitive blocking can no longer occur. Transition systems in which the labelset L is partitioned in a set of inputs L_I and a set of outputs L_U (i.e., $L = L_I \cup L_U$ and $L_I \cap L_U = \emptyset$), and which are always prepared to accept any input are called I/O automata (input/output automata) [9]. Figure 1 depicts the synchronisation between a coffee machine and its environment, both modelled as I/O automata ($L_I = \{coin\}$ and $L_U = \{coffee, tea\}$).

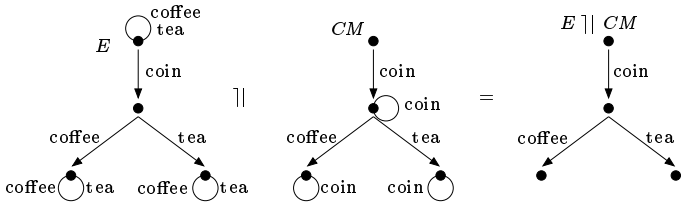

Figure 1 Coffee machine CM and environment E as I/O automata.

In [13, 15] testing theories based on the I/O automaton model (in terms of IOA and IOTS) are presented. These theories suffer from some deficiencies. First, the requirement imposed on IOA and IOTS to always accept input actions is quite strong. Secondly, [13, 15] implicitly assume that the environment is always capable of observing *every* output produced by the system, even if these outputs occur at geographically dispersed places (which is frequently the case if the system under test is distributed), and thereby ignores a possible distribution of the environment itself. In order to overcome these deficiencies we refine the I/O automaton model by taking the distribution of the interaction points (PCO's, points of control and observation [6]) on the interface of a system with its environment into account, and we weaken the requirement that inputs must be always enabled. This is accomplished by partitioning the inputs L_I in pairwise disjunct sets $L_I^1, \ldots, L_I^n$, and, similarly, L_U in pairwise disjunct sets $L_U^1, \ldots, L_I^m$. We shall refer to such sets as *channels*. The idea behind this partitioning is that each set L_I^j (or L_U^k) reflects the location on an

interface where these actions may occur. Furthermore, we weaken the requirement on input enabling to 'if some action in channel L_I^j can be performed, then all actions in channel L_I^j can be performed".

DEFINITION 6 *A multi input-output transition system p over partitioning $\mathcal{L}_I$ of L_I and partitioning $\mathcal{L}_U$ of L_U is a transition system with inputs and outputs, $p \in \mathcal{LTS}(L_I \cup L_U)$, such that for all $L_I^j \in \mathcal{L}_I$*

$$\forall p' \in der(p),\ \textit{if}\ \ \exists a \in L_I^j : p' \xrightarrow{a} \quad \textit{then}\ \ \forall b \in L_I^j : p' \xrightarrow{b}$$

The universe of multi input-output transition systems over $\mathcal{L}_I$ and $\mathcal{L}_U$ is denoted by $\mathcal{MIOTS}(\mathcal{L}_I, \mathcal{L}_U)$.

Each particular partitioning $\mathcal{L}_I$ and $\mathcal{L}_U$ induces a class of transition systems $\mathcal{MIOTS}(\mathcal{L}_I, \mathcal{L}_U) \subseteq \mathcal{LTS}(L_I \cup L_U)$. In order to compare these we define an ordering $\trianglelefteq$ on partitionings of a set S, where we restrict to finite partitionings.

DEFINITION 7 *Let $Parts(S)$ be the set of all partitionings of S and let $\mathcal{X}, \mathcal{Y} \in Parts(S)$, then $\mathcal{X} \trianglelefteq \mathcal{Y}\ =_{def}\ \forall X \in \mathcal{X}, \exists Y \in \mathcal{Y} : X \subseteq Y$*

The relation $\trianglelefteq$ reflects the ordering on the granularity of the partitioning involved, and defines a lattice on partitionings. The minimal element is the partitioning that contains only singleton sets consisting of elements of S: $\min_{\trianglelefteq}(Parts(S)) = \{\{s\} \mid s \in S\}$. The maximal element is the partitioning that consists of a single set containing all elements of S: $\max_{\trianglelefteq}(Parts(S)) = \{\{S\}\}$.

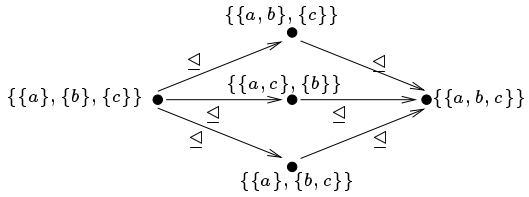

Figure 2 The partial order $\trianglelefteq$ applied to partitionings of $\{a, b, c\}$.

The granularity of the partitioning $\mathcal{L}_I$ uniquely defines the class of potential system models of implementations: the finer the partitioning of $\mathcal{L}_I$, the larger the class $\mathcal{MIOTS}(\mathcal{L}_I, \mathcal{L}_U)$. The granularity of the partitioning of the set of output actions L_U does not influence the class of potential system models of implementations (cf. definition 6).

PROPOSITION 2 *Let $\mathcal{X}_1, \mathcal{X}_2$ be partitionings of L_I, and let $\mathcal{Y}$ be a partitioning of L_U, then $\mathcal{X}_1 \trianglelefteq \mathcal{X}_2$ implies $\mathcal{MIOTS}(\mathcal{X}_1, \mathcal{Y}) \supseteq \mathcal{MIOTS}(\mathcal{X}_2, \mathcal{Y})$*

Specific instances of these partitionings yield some well-known classes. In particular, for the finest partitioning on L_I and L_U the requirement imposed on multi input-output transition systems trivially becomes true, and the set $\mathcal{MIOTS}(\min_{\trianglelefteq}(Parts(L_I)), \min_{\trianglelefteq}(Parts(L_U)))$ equals the set $\mathcal{LTS}(L_I \cup L_U)$. The set $\mathcal{MIOTS}(\max_{\trianglelefteq}(Parts(L_I)), \max_{\trianglelefteq}(Parts(L_U)))$ collapses with IOTS [15] in case input actions are always enabled.

4 TESTING MULTI INPUT-OUTPUT TRANSITION SYSTEMS

We give an extensional comparison criterion, cf. equation (1), that decides which implementations, modelled as MIOTS, can be distinguished by external observers, and which cannot. The set of external observers $\mathcal{U}$ are assumed to be modelled as MIOTS, too: an observer is able to accept all outputs at a specific location that are produced by the implementation as long as the observer is able to accept only one of them. Furthermore, in order to observe the inability to accept an input action, or the inability to produce an output action, observers are (analogous to definition 3) equipped with deadlock detection labels. This time we use different deadlock detection labels for each channel: $\theta_i^j \notin L_I \cup L_U$ to observe the inability of the implementation to accept an input action in channel L_I^j, and $\theta_u^k \notin L_I \cup L_U$ to observe the inability to produce outputs in channel L_U^k.

Now, implementations that are modelled as members of $\mathcal{MIOTS}(\mathcal{L}_I, L_U)$ are observed by observers modelled in $\mathcal{MIOTS}(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta)$, where $\mathcal{L}_U^\theta =_{def} \{L_U^1 \cup \{\theta_u^1\}, \ldots, L_U^m \cup \{\theta_u^m\}\}$ and $\mathcal{L}_I^\theta =_{def} \{L_I^1 \cup \{\theta_i^1\}, \ldots, L_I^n \cup \{\theta_i^n\}\}$. Communication between observer and system is modelled by operator $]\![$. The set Θ denotes the set of all deadlock detection labels: $\Theta =_{def} \{\theta_i^1, \ldots, \theta_i^n, \theta_u^1, \ldots, \theta_u^m\}$.

DEFINITION 8 *Let $\mathcal{L}_I = \{L_I^1, \ldots, L_I^n\}$ be a finite partitioning of L_I and let $\mathcal{L}_U = \{L_U^1, \ldots, L_U^m\}$ be a finite partitioning of L_U, then operator $]\![$: $\mathcal{MIOTS}(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta) \times \mathcal{LTS}(L_I \cup L_U) \rightarrow \mathcal{LTS}(L_I \cup L_U \cup \Theta)$ is defined by the following inference rules.*

$$\frac{u \xrightarrow{a} u', p \xrightarrow{a} p'}{u\,]\![p \xrightarrow{a} u'\,]\![p'} \ (a \in L_I \cup L_U) \qquad \frac{u \xrightarrow{\theta_i^j} u', init(p) \cap L_I^j = \emptyset}{u\,]\![p \xrightarrow{\theta_i^j} u'\,]\![p} \ (j \in \{1, \ldots, n\})$$

$$\frac{u \xrightarrow{\theta_u^k} u', init(p) \cap L_U^k = \emptyset}{u\,]\![p \xrightarrow{\theta_u^k} u'\,]\![p} \ (k \in \{1, \ldots, m\})$$

An observer $u \in \mathcal{MIOTS}(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta)$ that communicates with a system $p \in \mathcal{LTS}(L_I \cup L_U)$ may perform sequences of actions in $L_I \cup L_U$, possibly interleaved with deadlock detection labels θ_i^j and θ_u^k. Similar to definition 4, we define such sequences as the observations that can be made of such a system, thereby overloading the notations obs_c^θ and obs_t^θ.

DEFINITION 9 *Let* $p \in \mathcal{LTS}(L_I \cup L_U)$ *and* $u \in \mathcal{MIOTS}(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta)$.

1. *The set of completed trace observations* obs_c^θ *is*

$$obs_c^\theta(u,p) \;=_{def}\; \{\sigma \in (L_I \cup L_U \cup \Theta)^* \mid (u][p) \textbf{ after } \sigma \textbf{ deadlocks } \}$$

2. *The set of trace observations* obs_t^θ *is*

$$obs_t^\theta(u,p) \;=_{def}\; \{\sigma \in (L_I \cup L_U \cup \Theta)^* \mid (u][p) \xrightarrow{\sigma} \}$$

Now, following equation (1), and in the same line as definition 5, refusal preorder is defined under the assumption that implementations are modelled as members in $\mathcal{MIOTS}(\mathcal{L}_I, \mathcal{L}_U)$. However, we do not require this for specifications; specifications are just labelled transition systems over $L_I \cup L_U$.

DEFINITION 10 *The relation* $\leq_{mior} \subseteq \mathcal{MIOTS}(\mathcal{L}_I, \mathcal{L}_U) \times \mathcal{LTS}(L_I \cup L_U)$, *called multi input-output refusal preorder, is defined by*

$$i \leq_{mior} s \;=_{def}\; \forall u \in \mathcal{MIOTS}(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta) : \; obs_c^\theta(u,i) \subseteq obs_c^\theta(u,s) \;\; and \;\; obs_t^\theta(u,i) \subseteq obs_t^\theta(u,s)$$

Conceptually, when an observer experiments on an implementation that is modelled as MIOTS it can either provide inputs at an input channel (e.g., press a button), or observe outputs from an output channel (e.g., view a display). For each input channel the observer is equipped with a "finger" to perform a button-push experiment, and for each output channel the observer is equipped with an "eye" that notices the output actions occurring on the display (figure 3). By assumption, output actions at a specific location cannot be selectively perceived by observers: if one output can be observed, then all output actions at the same location can potentially be observed. Furthermore, it is assumed that unsuccessful input experiments and output experiments are noticed by the observer. Figure 3 depicts an interface for MIOTS.

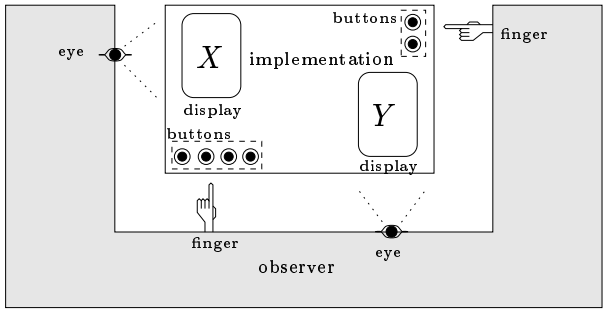

Figure 3 Observers of multi input-output transition systems.

A special class of observers are the *singular observers*. They consist of finite, serial compositions of providing a single input action at some channel L_I^j and detection of its acceptance or rejection, and observing some channel L_U^k and detection of the occurrence, or absence, of outputs produced at this channel. It turns out that it suffices to restrict to singular observers in order to establish whether implementations are $\leq_{mior}$-correct or not.

DEFINITION 11 *A singular observer u over $\mathcal{L}_U$ and $\mathcal{L}_I$ is a finite, deterministic multi input-output transition system $u \in \mathcal{MIOTS}(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta)$ such that*

$$\forall u' \in der(u): \ init(u') = \emptyset \quad or \quad init(u') = L_U^k \cup \{\theta_u^k\} \quad or \quad init(u') = \{a, \theta_i^j\}$$

for some $j \in \{1, \ldots, n\}, k \in \{1, \ldots, m\}$ and $a \in L_I^j \in \mathcal{L}_I$. The set of all singular observers over $\mathcal{L}_U$ and $\mathcal{L}_I$ is denoted by $\mathcal{SOBS}(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta)$.

PROPOSITION 3 *Let $i \in \mathcal{MIOTS}(\mathcal{L}_I, \mathcal{L}_U)$ and $s \in \mathcal{LTS}(L_I \cup L_U)$, then*

$$\begin{aligned}
& i \leq_{mior} s \\
\textit{iff} \ & \forall u \in \mathcal{SOBS}(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta): obs_c^\theta(u,i) \subseteq obs_c^\theta(u,s) \ \textit{and} \ obs_t^\theta(u,i) \subseteq obs_t^\theta(u,s) \\
\textit{iff} \ & \forall u \in \mathcal{SOBS}(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta): obs_c^\theta(u,i) \subseteq obs_c^\theta(u,s) \\
\textit{iff} \ & \textit{f-traces}(i) \cap (L_I \cup L_U \cup \mathcal{L}_I \cup \mathcal{L}_U)^* \subseteq \textit{f-traces}(s)
\end{aligned}$$

Since each singular observer is composed of actions that are able to detect whether an input at channel L_I^j is accepted or not, and observations that are able to detect whether outputs are produced at some channel L_U^k or not, it follows that execution of singular observers only ends in case no more actions can be conducted; the only way for a test execution $u]\![i$ to deadlock is by deadlock of u.

PROPOSITION 4 *Let $u \in \mathcal{SOBS}(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta)$ and $p \in \mathcal{MIOTS}(\mathcal{L}_I, \mathcal{L}_U)$, then*

$$(u]\![p) \ \textbf{after} \ \sigma \ \textbf{deadlocks} \quad \textit{implies} \quad u \ \textbf{after} \ \sigma \ \textbf{deadlocks}$$

The observation that can be made from observer u communicating with system p uniquely determines the failure trace that was performed by p. This is possible because every observation of θ_i^j and θ_u^k in $u]\![p$ corresponds to refusal of L_I^j and L_U^k, respectively. We denote with $\overline{\sigma}$ the trace σ where each occurrence of a refusals L_I^j or L_U^k is replaced by its detection label θ_i^j or θ_u^k, and vice versa.

PROPOSITION 5 *Let $p \in \mathcal{LTS}(L_I \cup L_U)$ and $u \in \mathcal{SOBS}(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta)$, then for any $\sigma \in (L_I \cup L_U \cup \Theta)^*$*

$$u]\![p \xrightarrow{\sigma} u']\![p' \quad \textit{iff} \quad u \xrightarrow{\sigma} u' \ \textit{and} \ p \xrightarrow{\overline{\sigma}} p'$$

Yet another characterisation of the relation $\leq_{mior}$ exists that is based on the responses that the implementation can produce after having performed a specific trace. These responses consist of the output suspension labels (δ^k) indicating that the implementation is in a state that cannot produce an output at channel L_U^k, the input suspension labels (ξ^j) indicating that the implementation is in a state that cannot accept any input from channel L_I^j, and the outputs in L_U that the implementation can produce in the current state. All these responses are collected in the set *out*.

DEFINITION 12 *Let $p \in \mathcal{LTS}(L_I \cup L_U)$ and $\sigma \in (L_I \cup L_U \cup \mathcal{L}_I \cup \mathcal{L}_U)^*$, then the set $out(p \textbf{ after } \sigma)$ is defined by*

$$\begin{aligned} out(p \textbf{ after } \sigma) =_{def} \quad & \{x \in L_U \mid \exists p' : p \xrightarrow{\sigma} p' \xrightarrow{x} \} \\ & \cup \{\xi^j \mid 1 \leq j \leq n, \exists p' : p \xrightarrow{\sigma} p' \text{ and } init(p') \cap L_I^j = \emptyset\} \\ & \cup \{\delta^k \mid 1 \leq k \leq m, \exists p' : p \xrightarrow{\sigma} p' \text{ and } init(p') \cap L_U^k = \emptyset\} \end{aligned}$$

The inability to accept input at channel L_I^j (i.e., input suspension) and the inability to produce output at channel L_U^k (i.e., output suspension) is now explicitly visible in terms of the input suspension labels ξ^j and the output suspension labels δ^k, respectively. It turns out that an implementations is $\leq_{mior}$-related to a specification in case all responses that the implementation can perform after a trace in $(L_I \cup L_U \cup \mathcal{L}_I \cup \mathcal{L}_U)^*$ are specified, i.e., an implementation is not allowed to suspend at some channel in case this is not specified, and the implementation is not allowed to produce unspecified outputs.

PROPOSITION 6 *Let $i \in \mathcal{MIOTS}(\mathcal{L}_I, \mathcal{L}_U)$ and $s \in \mathcal{LTS}(L_I \cup L_U)$, then*

$$i \leq_{mior} s \quad \textit{iff} \quad \forall \sigma \in (L_I \cup L_U \cup \mathcal{L}_I \cup \mathcal{L}_U)^* : out(i \textbf{ after } \sigma) \subseteq out(s \textbf{ after } \sigma)$$

Checking the condition in proposition 6 for all traces in $(L_I \cup L_U \cup \mathcal{L}_I \cup \mathcal{L}_U)^*$ is too time consuming in practice. Therefore, we generalise this condition to an arbitrary (and possible finite) set $\mathcal{F} \subseteq (L_I \cup L_U \cup \mathcal{L}_I \cup \mathcal{L}_U)^*$, and define a corresponding implementation relation $\textbf{mioco}_{\mathcal{F}}$ in the same way as $\textbf{ioco}_{\mathcal{F}}$ in [15]. We will use this relation in the next section as the basis for deriving tests.

DEFINITION 13 *The implementation relation $\textbf{mioco}_{\mathcal{F}} \subseteq \mathcal{MIOTS}(\mathcal{L}_I, \mathcal{L}_U) \times \mathcal{LTS}(L_I \cup L_U)$, where $\mathcal{F} \subseteq (L_I \cup L_U \cup \mathcal{L}_I \cup \mathcal{L}_U)^*$, is defined by*

$$i \textbf{ mioco}_{\mathcal{F}} \; s \;\; =_{def} \;\; \forall \sigma \in \mathcal{F} : \;\; out(i \textbf{ after } \sigma) \subseteq out(s \textbf{ after } \sigma)$$

Furthermore, we define $\textbf{mioco} \;\; =_{def} \;\; \textbf{mioco}_{f\text{-}traces(s) \cap (L_I \cup L_U \cup \mathcal{L}_I \cup \mathcal{L}_U)^*}$.

We remark here that, in general, observers in $\mathcal{SOBS}(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta)$ are more

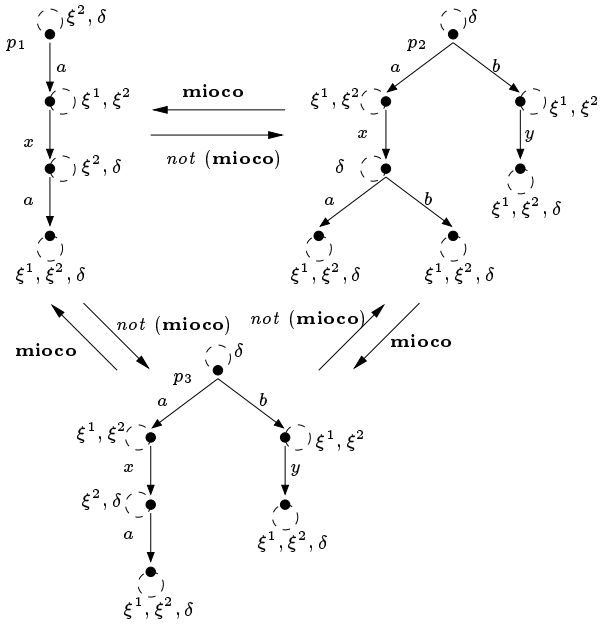

Figure 4 The relation **mioco** with $\mathcal{L}_I = \{\{a\},\{b\}\}$ and $\mathcal{L}_U = \{\{x,y\}\}$.

powerful than observers for input/output automata or IOTS (cf. [13, 15]) due to their ability to observe output suspension at different output channels: singular observers can distinguish between systems that are unable to produce output actions at one channel, while at another channel the system is able to produce output actions. In terms of the relation $\trianglelefteq$ on partitions, this means that the finer the outputs are partitioned (i.e., the more output channels are present), the more selectively observers are able to observe. In particular, for the finest partitioning of inputs and the finest partitioning of outputs our relation $\leq_{mior}$ collapses with $\leq_{rf}$, while we claim that for the coarsest partitioning of the inputs and the outputs the relation $\leq_{mior}$ collapses with **ioco** [15] in case it is assumed that for implementations inputs are always enabled.

PROPOSITION 7 *Let* $i \in \mathcal{MIOTS}(\min_{\trianglelefteq}(Parts(L_I)), \min_{\trianglelefteq}(Parts(L_U)))$ *and* $s \in \mathcal{LTS}(L_I \cup L_U)$ *such that* $\min_{\trianglelefteq}(Parts(L_I))$ *and* $\min_{\trianglelefteq}(Parts(L_U)))$ *are finite, then* $i \leq_{rf} s$ *iff* $i \leq_{mior} s$

5 TEST GENERATION FOR MIOTS

In this section we develop an algorithm to derive tests systematically from a specification such that these tests are able to reject implementations that are

mioco$_{\mathcal{F}}$-incorrect, and accept implementations that are **mioco**$_{\mathcal{F}}$-correct. The algorithm depends on the specification (modelled as a member of $\mathcal{LTS}(L_I \cup L_U)$), the correctness criterion (**mioco**$_{\mathcal{F}}$ for some $\mathcal{F}$), and the test assumption (implementations are modelled as members of $\mathcal{MIOTS}(\mathcal{L}_I, \mathcal{L}_U)$).

Test cases need to incorporate some kind of verdict that can be used to give such an indication about the (in)correctness of implementations when running these test cases against implementations. We distinguish between two kinds of verdicts: **pass** to indicate that the implementation behaved as expected, and **fail** to indicate that the implementation behaved erroneously (cf. [6, 7]). We define a test as a member of $\mathcal{SOBS}(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta)$ where the final states are identified with the verdicts **pass** or **fail**.

DEFINITION 14 *A test t over $\mathcal{L}_I^\theta$ and $\mathcal{L}_U^\theta$ is a singular observer $t \in \mathcal{SOBS}(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta)$ such that for each $t' \in der(t)$*

$$init(t') = \emptyset \quad \textit{iff} \quad t' = \textbf{\textit{pass}} \quad \textit{or} \quad t' = \textbf{\textit{fail}}$$

The universe of tests over $\mathcal{L}_I^\theta$ and $\mathcal{L}_U^\theta$ is denoted by $TESTS(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta)$.

Since tests will always end in a final state of the test (proposition 4) every test run is assigned a verdict, viz., the verdict of the final state of the test. Trace σ is a test run of $t][i$ iff $\sigma \in obs_c^\theta(t, i)$. An implementation i fails test t if there exists a test run of $t][i$ leading to a **fail** state, (i.e., i *fails* t $=_{def}$ $\exists \sigma \in obs_c^\theta(t,i), \exists i' : t][i \xrightarrow{\sigma} \textbf{fail}][i']$), and implementation i passes test t if it does not fail t. Implementation i fails a set of tests T if i fails a test $t \in T$, otherwise it passes T.

Soundness, exhaustiveness and completeness [7] are properties of test suites (i.e., sets of tests) that link the passing or failing of test suites to the correctness of the implementations. A test suite is called sound if this test suite will never reject **mioco**$_{\mathcal{F}}$-correct implementations, and a test suite is called exhaustive if each incorrect implementation always fails this test suite. In practice test suites are required to be sound, but not necessarily exhaustive; any error that is detected by a test suite indeed proves that the implementation under test was incorrect, but not finding an error does not mean that the implementation is error free! A test suite is called complete if it is both sound and exhaustive.

Figure 5 presents a test generation algorithm Π that produces tests that are able to distinguish between **mioco**$^\theta_{\mathcal{F}}$-correct and **mioco**$^\theta_{\mathcal{F}}$-incorrect implementations. The rationale behind the test algorithm is that it construct tests that check the condition

$$out(\, i \textbf{ after } \sigma \,) \subseteq out(\, s \textbf{ after } \sigma \,)$$

for $\sigma \in \mathcal{F}$ (cf. definition 13). The test generation algorithm takes a specification $s \in \mathcal{LTS}(L_I \cup L_U)$ and a set of failure traces $\mathcal{F} \subseteq (L_I \cup L_U \cup \mathcal{L}_I \cup \mathcal{L}_U)^*$, and produces tests in $TESTS(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta)$. The variable S keeps track of the current states in the specification, which initially equals $\{s_0\}$ **after** ϵ, and the

variable $\mathcal{F}$ keeps track of the failure traces that need to be investigated in order to establish correctness. Each time an action is performed the sets S and $\mathcal{F}$ are updated accordingly.

Input: set of states S
Input: set of failure traces $\mathcal{F} \subseteq (L_I \cup L_U \cup \mathcal{L}_I \cup \mathcal{L}_U)^*$
Output: test case $\Pi_{\mathcal{F},S} \in TESTS(\mathcal{L}_U^\theta, \mathcal{L}_I^\theta)$.

Initial value: $S = \{s_0\}$ **after** ϵ, where s_0 is the initial state of s.

Apply one of the following non-deterministic choices recursively.

1. (* terminate the test case if there are no more specified traces in $\mathcal{F}$ *)
   if $\mathcal{F} = \emptyset$ then
   $$\Pi_{\mathcal{F},S} := \mathbf{pass}$$
2. (* terminate the test case when a trace $\sigma \in \mathcal{F}$ has been performed *)
   if $\epsilon \in \mathcal{F}$ then take some $L_I^j \in \mathcal{L}_I$, and for some $a \in L_I^j$ (* supply input a *)
   $$\Pi_{\mathcal{F},S} := \begin{cases} a;\mathbf{pass} + \theta_i^j;\mathbf{fail} & \text{if } S \textbf{ after } L_I^j = \emptyset \\ a;\mathbf{pass} + \theta_i^j;\mathbf{pass} & \text{if } S \textbf{ after } L_I^j \neq \emptyset \end{cases}$$
3. (* terminate the test case when a trace $\sigma \in \mathcal{F}$ has been performed *)
   if $\epsilon \in \mathcal{F}$ then take some $L_U^k \in \mathcal{L}_U$, then (* observe channel L_U^k *)
   $$\Pi_{\mathcal{F},S} := \sum\{x;\mathbf{pass} \mid x \in L_U^k \cup \{\theta_u^k\} \text{ and } S \textbf{ after } \overline{x} \neq \emptyset\} + \sum\{x;\mathbf{fail} \mid x \in L_U^k \cup \{\theta_u^k\} \text{ and } S \textbf{ after } \overline{x} = \emptyset\}$$
4. (* supply an input for which you want to test deeper *)
   Take some $L_I^j \in \mathcal{L}_I$ and $a \in L_I^j$ such that $\{\sigma \mid a{\cdot}\sigma \in \mathcal{F}\} \neq \emptyset$, then
   $$\Pi_{\mathcal{F},S} := a;\Pi_{\mathcal{F}',S'} + \theta_i^j;\mathbf{pass}$$
   where $S' = S$ **after** a, $\mathcal{F}' = \{\sigma \mid a{\cdot}\sigma \in \mathcal{F}\}$
5. (* supply some input and continue if it is refused *)
   Take some $L_I^j \in \mathcal{L}_I$ such that $\{\sigma \mid L_I^j{\cdot}\sigma \in \mathcal{F}\} \neq \emptyset$, then
   $$\Pi_{\mathcal{F},S} := a;\mathbf{pass} + \theta_i^j;\Pi_{\mathcal{F}'',S''}$$
   where $a \in L_I^j, S'' = S$ **after** $L_I^j, \mathcal{F}'' = \{\sigma \mid L_I^j{\cdot}\sigma \in \mathcal{F}\}$
6. (* Find a channel L_U^k that produces an output for which to test deeper *)
   Take some $L_U^k \in \mathcal{L}_U$ such that $\{\sigma \mid \exists x \in L_U^k \cup \{L_U^k\} : x{\cdot}\sigma \in \mathcal{F}\} \neq \emptyset$, then
   $$\Pi_{\mathcal{F},S} := \sum\{x;\Pi_{\mathcal{F}',S'} \mid x \in L_U^k \cup \{\theta_u^k\} \quad \text{and } \mathcal{F}' = \{\sigma \mid \overline{x}{\cdot}\sigma \in \mathcal{F}\} \text{ and } S' = S \textbf{ after } \overline{x}\}$$

Figure 5 Test generation algorithm.

Step 1 of the algorithm assigns **pass** in case no failure trace in $\mathcal{F}$ was performed (e.g., because the implementation responds with an output action that is not checked for in $\mathcal{F}$). Step 2 of the algorithm checks for all input channels whether the implementation is allowed to suspend input. Note that S **after** $L_I^j = \emptyset$ means that there is no state in S that can perform refusal transition L_I^j. Step 3 checks for all output channels whether all outputs that the

implementation can produce are indeed specified. Step 4 supplies an input to the implementation at some channel L_I^j and continues if the implementation is able to accept this input. Step 5 also supplies an input to the implementation at some channel L_I^j but now the algorithm recursively proceeds if the input is refused. Finally, step 6 awaits an output action or observes an output suspension at output channel L_U^k after which the algorithm recursively proceeds.

Note that the algorithm is guaranteed to finish in case the set $\mathcal{F}$ contains a finite number of failure traces; in every step the length of the failure traces in $\mathcal{F}$ are reduced, and since all failure traces in $\mathcal{F}$ are (by definition) finite eventually step 2 or step 3 will always be applied.

PROPOSITION 8 *Let* $\mathcal{F} \subseteq (L_I \cup L_U \cup \mathcal{L}_I \cup \mathcal{L}_U)^*$ *and* $s \in \mathcal{LTS}(L_I \cup L_U)$

1. *Any test case obtained from algorithm* Π *for* s *and* $\mathcal{F}$ *is sound for* s *with respect to* $\mathbf{mioco}_{\mathcal{F}}$.
2. *The set of all test cases that can be obtained from algorithm* Π *for* s *and* $\mathcal{F}$ *is complete for* s *with respect to* $\mathbf{mioco}_{\mathcal{F}}$.

REMARK 1 The algorithm presented in figure 5 can be seen as an extension of the one presented in [15] in two ways. First of all, [15] considers implementations that are modelled as IOTS, so refusal of input is not considered. Secondly, the algorithm in [15] is not able to deal with the different input channels and different output channels on interfaces of implementations.

Although our algorithm is applicable to different classes of implementations, the algorithm in [15] is (probably) more efficient in deriving tests for IOTS than ours; it is likely that they need less tests to obtain a complete test suite for these kind of systems than we do.

6 ILLUSTRATION OF THE ALGORITHM

Consider the coffee machine CM depicted in figure 6. After insertion of a coin ($coin$) a user may press either the coffee button (cb) or the tea button (tb), which results in the production of coffee (cof) or tea (tea), respectively. There are two distinct input channels (a channel to insert coins and a channel to push buttons) and a single output channel for providing coffee or tea: $CM \in \mathcal{MIOTS}(\{\{coin\}, \{cb, tb\}\}, \{\{cof, tea\}\})$. The dashed arrows labelled ξ^1, ξ^2 and δ^1 denote refusal transitions for the sets $\{coin\}, \{cb, tb\}$ and $\{cof, tea\}$, respectively.

Figure 6 also depicts some tests that are derived from CM for $\mathcal{F} = \{\epsilon, coin \cdot cb\}$ using algorithm Π (see figure 5). For readability the steps of the algorithm that were applied are indicated in the nodes of the tests. Tests (a) is an immediate consequence of step (2) of the algorithm, and test (b) an immediate consequence of step (3). Test (a) checks that implementations initially *must* accept a coin (refusal of a coin gives a **fail** verdict), and test (b) checks

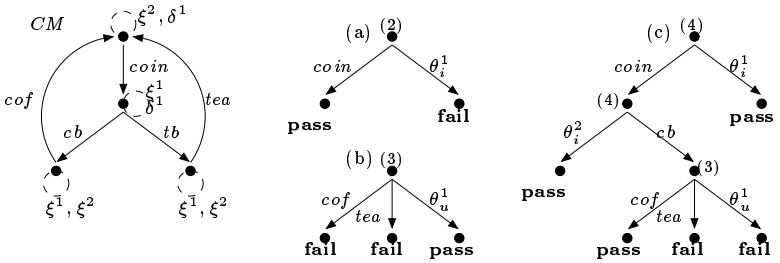

Figure 6 Some tests generated by Π from *CM*.

that implementations are initially not allowed to provide free drinks. Test (c) follows from the successive application of step (4), again step (4), and step (3). It checks that after $coin \cdot cb$ the production of tea, or the suspension of providing a drink, is considered incorrect.

Note that algorithm Π may produce tests that always return the verdict **pass** (e.g., $cb \cdot \mathbf{pass} + \theta_i^2 \cdot \mathbf{pass}$). Execution of such tests is not very sensible. The derivation of such meaningless tests indicates that the algorithm is not optimal and that there is room for improvement.

7 CONCLUSIONS AND FURTHER WORK

Conclusions In this paper the theory of refusal testing [12] has been applied to several classes of transition systems that distinguish between the initiative of actions: either input or output. Each class is induced by the distribution of the locations through which these systems communicate with their environment. In this way a refusal testing theory is obtained that is parameterised by the distribution of the interface of implementations. Specific choices for the interfaces yield the seminal refusal testing theory of [12], and the (repetitive) quiescent trace testing theory for I/O automata [13] and for input/output transition systems [14, 15]. For the large variety of classes of transition systems that can be obtained, a correctness criterion $\mathbf{mioco}_{\mathcal{F}}$ (definition 13) is defined that is explicitly parameterised by a set of failure traces $\mathcal{F}$. For all these classes of systems and the corresponding correctness criteria a single test generation algorithm (figure 5) is defined that is able to produce a sound and complete test suite from a specification. This algorithm is an extension of the one in [15]: that one is applicable to a smaller class of systems and is not parameterised over the distribution of the interface of implementations.

Further work The test generation algorithm Π can produce a large, and possibly infinite, number of tests. Since it is not feasible to execute all of them, techniques have to be developed to measure the relevance of tests (coverage), to select the most relevant tests from a larger set of tests (test selection), or

to avoid the generation of irrelevant tests. Furthermore, the test generation algorithm needs to handle data in a symbolic way in order to avoid explosion of the state space, and keep test generation manageable. Since the correctness criterion $\mathbf{mioco}_{\mathcal{F}}$ is based on traces, i.e., linear sequences only, an explosion due to the branching structure of specifications (e.g., as in [2]) is avoided. Also, mechanisms to observe the suspension of input or output have to be developed, e.g., making use of timers: if no action occurs before the time-out it is assumed that no action can occur anymore. This requires techniques to carefully choose the timer values such that no incorrect suspension can be observed. Furthermore, the relation between MIOTS and input-complete Finite State Machines needs to be investigated (see, e.g., [10]).

8 REFERENCES

[1] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1990.

[2] E. Brinksma. A theory for the derivation of tests. In S. Aggarwal and K. Sabnani, eds., *PSTV VIII*, p. 63–74. North-Holland, 1988.

[3] R. De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, 24:211–237, 1987.

[4] R. De Nicola and M.C.B. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[5] ISO. *LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour.* IS-8807. ISO, Geneve, 1989.

[6] ISO. *Conformance Testing Methodology and Framework.* IS-9646. ISO, Geneve, 1991. Also: CCITT X.290–X.294.

[7] ISO/IEC JTC1/SC21 WG7, ITU-T SG 10/Q.8. *Proposed ITU-T Z.500 and Committee Draft on "Formal Methods in Conformance Testing".* CD 13245-1. ISO – ITU-T, Geneve, 1996.

[8] R. Langerak. A testing theory for LOTOS using deadlock detection. In E. Brinksma et. al., eds., *PSTV IX*, p. 87–98. North-Holland, 1990.

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

[10] A. Petrenko, G. v. Bochmann, and R. Dssouli. Conformance relations and test derivation. In O. Rafiq, ed., *IWPTS VI*, p. 157–176. North-Holland, 1993.

[11] M. Phalippou. *Relations d'Implantation et Hypothèses de Test sur des Automates à Entrées et Sorties.* PhD thesis, L'Université de Bordeaux I, 1994.

[12] I. Phillips. Refusal testing. *Theoretical Computer Science*, 50(2):241–284, 1987.

[13] R. Segala. Quiescence, fairness, testing, and the notion of implementation. In E. Best, ed., *CONCUR'93*, p. 324–338. LNCS 715, Springer-Verlag, 1993.

[14] J. Tretmans. Conformance testing with labelled transition systems: Implementation relations and test generation. *Computer Networks and ISDN Systems*, 29:49–79, 1996.

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