

# A test generation framework for *quiescent* real-time systems (Extended Version)

Laura Brandán Briones and Ed Brinksma

Faculty of Computer Science, University of Twente,  
P.O.Box 217, 7500AE Enschede,  
The Netherlands. Fax - (31 53)-489-3247  
{brandanl,brinksma}@cs.utwente.nl

**Abstract.** We present an extension of Tretmans' theory and algorithm for test generation for input-output transition systems to real-time systems. Our treatment is based on an operational interpretation of the notion of *quiescence* in the context of real-time behavior. This gives rise to a family of implementation relations parameterized by observation durations for *quiescence*. We define a nondeterministic (parameterized) test generation algorithm that generates test cases that are sound with respect to the corresponding implementation relation. Also, the test generation is exhaustive in the sense that for each non-conforming implementation a test case can be generated that detects the non-conformance.

## 1 Introduction

Although testing has always been the most important technique for the validation of software systems it has only become a topic of serious academic research in the past decade or so. In this period research on the use of formal methods for model-driven test generation and execution of functional test cases has led to a number of promising methods and tools for systematic black-box testing of systems, e.g. [BFV<sup>+</sup>99,TB03,FJJV96,FJJV97]. Most of these approaches are limited to the qualitative behaviour of systems, and exclude quantitative aspects such as real-time properties. The explosive growth of embedded software, however, has also caused a growing need to extend existing testing theories to the testing of real-time reactive systems. In this paper we present an extension of Tretmans' *ioco* theory for test generation [Tre96] for input-output transition systems that includes real-time behaviour.

A central concept in the non-timed theory is the notion of *quiescence*, which characterizes systems states that will not produce any output response without the provision of a new input stimulus. By treating *quiescence* as a special sort of system output the notion of behavioural trace can be generalized to include observations of *quiescence*. In turn, this leads to an implementation relation that defines unambiguously if implemented behaviour conforms to a given specification model, viz. if after all specified generalized traces of the implementation all possible generalized outputs are allowed according to the specification. Or, more informally, if all outputs and *quiescence* are correctly predicted by the specification.

In practice, the above implementation criterion means that implementations can be more deterministic than their specifications. Although it is good engineering practice to not introduce unnecessary nondeterminism in reactive systems, it is often unavoidable in the context of testing, and it should therefore be part of a sensible testing theory. The reason for this is twofold:

- Although the implementation under test may be deterministic, it can often only be tested through a testing environment that includes operating system features, communication media, etc. that typically introduce nondeterminism into the observed behaviour.

- An implementation under test often consists of concurrent components in an asynchronous parallel composition. The loss of information about the relative progress of components results in nondeterministic properties of their integrated behaviour.

Our proposed extension of the **ioco** theory to real-time systems is based on an operational interpretation of the notion of *quiescence*. This gives rise to a family of implementation relations parameterized by observation durations for *quiescence*. We define a nondeterministic (parameterized) test generation algorithm that generates test cases that are sound with respect to the corresponding implementation relation. This means that if an implementation fails any of the generated tests, it must be non-conforming. The algorithm is also exhaustive in the sense that for every non-conforming implementation a test case can be generated that will detect its non-conformance.

The rest of this paper is organized as follows. Section 2 is a summary of ioco-testing. Section 3 introduces the model of timed input-output transition systems and our conformance relation. Section 4 presents the real-time test generation algorithm. Section 5 illustrates the theory with an example in the setting of timed automata. Section 6 compares our achievements to related work. Finally, Section 7 presents the conclusions and future work.

## 2 Untimed ioco-testing

In this section we give a brief summary of input-output transition systems and the implementation relation **ioco**, from [Tre96]. An input-output transition system (IOTS) is a labelled transition system whose action set is partitioned into input actions (whose occurrence is controlled by the environment of the system) and output actions (whose occurrence is controlled by the system). We follow the convention that input actions are identified by names followed by a  $?$ -symbol, and output actions by names followed by a  $!$ -symbol.

**Definition 1.** An *Input-Output Transition System (IOTS)* is a 4-tuple  $\langle S, s_0, L, \rightarrow \rangle$ , where

- $S$  is a countable, non-empty set of states. With  $s_0 \in S$  as the initial state.
- $L$  is a countable set of labels, partitioned into input ( $L_I$ ) and output ( $L_U$ ) actions, with  $L_I \cap L_U = \emptyset$  and  $L_I \cup L_U = L$ .
- $\rightarrow \subseteq (S \times (L \cup \{\tau\}) \times S)$  is the transition relation.

To obtain the desired control over the occurrence of input and output actions, in case of an implementation, a *IOTS* must be able to accept all input actions in any state, i.e. the *IOTS* is *input-enabled*. Given that input actions are always enabled in *input-enabled IOTS* systems, traditional deadlock states cannot exist. Then a weaker notion becomes relevant: states that cannot produce (further) output actions without the supply of an input actions; such states are called *quiescent states*.

**Definition 2.** Let  $s$  be a state in an *IOTS*( $L$ ), then

$$s \text{ is quiescent} \quad \text{if and only if} \quad \forall a \in (L_U \cup \{\tau\}) : s \not\stackrel{a}{\rightarrow}$$

where the notation  $s \not\stackrel{a}{\rightarrow}$  denotes that there is no transition from  $s$  labelled with  $a$ , and with  $\delta(s)$  we denote that the state  $s$  is quiescent.

The idea is to treat *quiescence* as an observable event. This can be formalized by extending the transition relation adding self-loops to *quiescents* states. Then, a transition  $s \xrightarrow{\delta} s$  is added for each *quiescent* state  $s$  such that  $\delta(s)$ :

$$\delta(s) \text{ if and only if } s \xrightarrow{\delta} s$$

Using the *quiescent* information we introduce the notion of a suspension trace, i.e. a sequence of input actions, output actions and *quiescences* as they can occur when observing an *IOTS*.

**Definition 3.** Let  $p$  be a *IOTS* with the transition relation  $(\rightarrow)$  extended with quiescence transitions  $s \xrightarrow{\delta} s$  in case  $s$  is quiescent. Then, the suspension traces of process  $p$  are:

$$\text{Straces}(p) \triangleq \{\sigma \in L_{\delta}^* \mid p \xrightarrow{\sigma}\}$$

where  $L_{\delta}$  means  $L \cup \{\delta\}$ .

We use the well-known notation  $p \xrightarrow{\sigma}$  to say that there exists a state  $s$  that is reachable from  $p$  by performing  $\sigma$  abstracting from the internal actions (see next section for a formal definition of the  $\Rightarrow$ -relation). Moreover, we do not always distinguish between a input-output transition system and its initial state: if  $p = \langle S, s_0, L, \rightarrow \rangle$  we will often identify the process  $p$  with its initial state  $s_0$ , e.g. we write  $p \xrightarrow{\sigma}$  instead of  $s_0 \xrightarrow{\sigma}$ .

For input-output transition systems all output actions that are enabled in state  $p$ , including the *quiescence* action  $\delta$ , are collected into the set  $out(p)$ .

**Definition 4.** Let  $s$  be a state and let  $S$  be a set of states in a *IOTS*, then

$$\begin{aligned} out(s) &\triangleq \{a \in L_U \mid s \xrightarrow{a}\} \cup \{\delta \mid \delta(s)\} \\ out(S) &\triangleq \bigcup_{s \in S} out(s) \end{aligned}$$

The correctness of an implementation  $q$  (an *input-enabled IOTS*) with respect to a specification  $p$  (an *IOTS*) is given by the *implementation relation* **ioco**, defined by:

$$q \text{ ioco } p \triangleq \forall \sigma \in \text{Straces}(p) : out(q \text{ after } \sigma) \subseteq out(p \text{ after } \sigma)$$

where  $(p \text{ after } \sigma)$  denotes the set of states that is reachable from  $p$  by performing the *Trace*  $\sigma$ .

Informally, this means that the implementation  $q$  is **ioco**-correct with respect to a specification  $p$ , if and only if, after all possible behaviours of the specification ( $\forall \sigma \in \text{Straces}(p)$ ), any output action  $a$  produced by the implementation ( $a \in out(q \text{ after } \sigma)$ ) can also occur as an output of the specification ( $a \in out(p \text{ after } \sigma)$ ). In particular, this should also hold for the special action *quiescence* ( $\delta$ ), which models the absence of outputs.

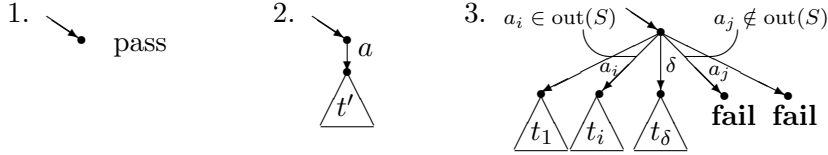
Using this theory, the following algorithm for test derivation is presented in [Tre96]. A test case is understood as the specification of the behaviour of a deterministic and finite testing process that can be carried out with an implementation under test. Also the behaviour of test cases can be described by input-output transition systems, where the occurrence of a  $\delta$ -action<sup>1</sup> in a test case corresponds to the detection of *quiescence* in an implementation, i.e. the observation that no output is produced. In practice, the observation of  $\delta$  is implemented using a time-out of sufficiently long duration.

A state of a test case is either a terminal state labelled (**fail** or **pass**), or a state that offers one particular input (to the implementation), or a state accepting all possible outputs (from the implementation), including the  $\delta$ -action. The class of test cases is denoted by *TEST*.

<sup>1</sup> In [Tre96] the action symbol  $\theta$  is used for the observation of *quiescence*. We prefer to use  $\delta$  for both *quiescence* and its observation, in line with the philosophy that identical actions synchronize.

Let  $p \in IOTS$  be a specification with initial state  $s_0$  and  $S$  be a non-empty set of states, with  $S$  initially equal to the  $\{s_0 \text{ after } \epsilon\}$  set (where  $\epsilon$  denotes the empty trace). Then,  $S$  represents the set of all possible states in which the implementation can be at the current stage of the test case execution. As in [Tre96], we present the test using a syntax inspired by LOTOS [ISO89]; the concrete semantics for the timed case will be explained in section 4.

The algorithm for the generation of test cases  $t \in TEST$  from  $S$  consists of a finite number of recursive applications of a nondeterministic selection between one of the following three alternatives:



1. The single state test case **pass**, which stops the recursion in the algorithm and thus terminates the test case:

$$t := \text{pass}$$

2. Test case  $t$  supplies the input  $a$  and behaves as test case  $t'$ :

$$t := a; t'$$

where  $a \in L_I, (S \text{ after } a) \neq \emptyset$ , and  $t'$  is obtained by recursively applying the algorithm to  $(S \text{ after } a)$ .

3. Test case  $t$  checks the next output from implementation; if it is a valid response (i.e.  $a \in \text{out}(S)$ ) the test case continues recursively; if it is an invalid response (i.e.  $a \notin \text{out}(S)$ ) then the test case terminates in **fail**. The observation of *quiescence* ( $\delta$ ) is treated separately:

$$\begin{aligned} t := & \Sigma \{a_i; t_i \mid a_i \in L_U \wedge a_i \in \text{out}(S)\} \\ & + \Sigma \{\delta; t_\delta \mid \delta \in \text{out}(S)\} \\ & + \Sigma \{a_j; \text{fail} \mid a_j \in L_U \wedge a_j \notin \text{out}(S)\} \\ & + \Sigma \{\delta; \text{fail} \mid \delta \notin \text{out}(S)\} \end{aligned}$$

where  $t_i$  and  $t_\delta$  are obtained by recursively applying the algorithm for  $(S \text{ after } a_i)$  and  $(S \text{ after } \delta)$ , respectively.

This algorithm is implemented in the test generation tool TORX. Indeed, the developed theory plus its tool support provided by TORX have proven quite useful and successful approach to the functional testing of reactive systems [TB03].

*Example 1.* The example present here is an adaptation of an example due to Langerak [Lan90]. Figure 1 shows two coffee machines with peculiar behaviour. They illustrate the importance of being able to observe *quiescence* and act on the basis of this information, as formalized by the **io** relation. We follow the convention, even not explicitly shown in the graph, that all implementation systems (and in this particular example also the specification) are saturated with input-action transitions. Therefore, we consider in each state the adding of input self-loops for all input transitions that are not explicitly given. We have that after accepting a *coin?* both machines can end up in state  $q_1$  ( $q_2$ ) where only the action *tea?* (*coffee?*) leads to

the desired effect, viz. the corresponding output of  $tea!$  ( $coffee!$ ). In the left-hand side machine the user can switch between the  $tea$ - and  $coffee$ -mode by  $bang?$ -ing the machine after noticing that the desirable drink is not produced. In the right-hand side machine such switches are not possible. The **io** relation distinguish these two machines, because:

$$\text{out}(\text{left-hand machine after } coin? \cdot coffee? \cdot \delta \cdot bang? \cdot coffee?) = \{coffee!\}$$

whereas

$$\text{out}(\text{right-hand machine after } coin? \cdot coffee? \cdot \delta \cdot bang? \cdot coffee?) = \{\delta\}$$

However, if the special action  $\delta$  is drop from the definition of suspension traces and *out*-sets, it leads to the input-output testing pre-order [Tre96], both machines would be indistinguishable because we could not use the *quiescence* from states  $q_1$  and  $q_2$ .

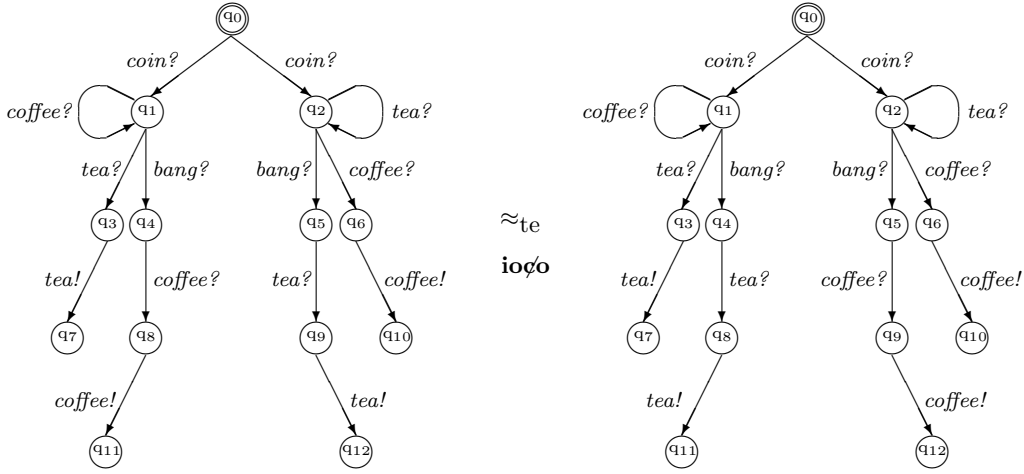


Fig. 1. The quirky coffee machine, a modified version of [Lan90]

### 3 Implementation relations for real-time *quiescence* systems

#### 3.1 Timed input-output transition systems

In this section we present the *Timed Labelled Transition Systems* (*TLTS*), their properties and notation. Later, we specialize the *TLTS* to obtain the model of *Timed Input-Output Transition Systems* (*TIOTS*). Furthermore, we introduce a conformance relation between a specification defined as a *TLTS* and an implementation defined as *TIOTS*, analogous to the **io** relation for the untimed case.

We distinguish three types of actions: *time-passage actions*, *visible labelled actions* and the special *internal action*  $\tau$ . All except the time-passage actions are thought of as occurring instantaneously, i.e. without consuming time. To specify time, a dense time domain is used (i.e. the nonnegative reals  $\mathbb{R}^{\geq 0}$ ). No lower *a priori* bounds are imposed on the delays between events.

**Definition 5.** A *Timed Labelled Transition System* (*TLTS*) is a 4-tuple  $\langle S, s_0, L_{\tau\mathcal{T}}, \rightarrow \rangle$ , where

- $S$  is a non-empty set of states. With  $s_0 \in S$  as the initial state.
- $L_{\tau\mathcal{T}} \triangleq L \cup \{\tau\} \cup \mathcal{T}$  are the actions  $L$  including the internal action  $\tau$  and time-passage actions; where  $\mathcal{T}$  is  $\{d \mid d \in \mathbb{R}^{\geq 0}\}$
- $\rightarrow \subseteq (S \times L_{\tau\mathcal{T}} \times S)$  is the transition relation with the following consistency constraints:
  - **Time Determinism** whenever  $s \xrightarrow{a} s'$  and  $s \xrightarrow{a} s''$  then  $s' = s''$
  - **Time Additivity**  $\forall s, s'' \in S \wedge \forall d_1, d_2 \geq 0 : (\exists s' \in S : s \xrightarrow{d_1} s' \xrightarrow{d_2} s'')$  if and only if  $s \xrightarrow{d_1+d_2} s''$
  - **Null Delay**  $\forall s, s' \in S : s \xrightarrow{0} s'$  if and only if  $s = s'$

The labels in  $L_{\mathcal{T}}$  ( $L_{\mathcal{T}} \triangleq L \cup \mathcal{T}$ ) represent the observable actions of a system, i.e. labelled actions and passage of time; the special label  $\tau$  represents an unobservable internal action. A transition  $(s, a, s') \in \rightarrow$  is denoted as  $s \xrightarrow{a} s'$ . A *computation* is a finite or infinite sequence of transitions:

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \dots \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s_n (\rightarrow \dots)$$

A *timed trace* captures the observable aspects of a computation, is the sequence of observable actions ( $L_{\mathcal{T}}$ ). The set of all *finite* sequences of actions over  $L_{\mathcal{T}}$  is denoted by  $L_{\mathcal{T}}^*$ , while  $\epsilon$  denotes the empty sequence. If  $\sigma_1, \sigma_2 \in L_{\mathcal{T}}^*$  then with  $\sigma_1 \cdot \sigma_2$  we denote the concatenation of  $\sigma_1$  and  $\sigma_2$ .

The class of all timed labelled transition systems over  $L$  is denote by  $TLTS(L)$ . Some additional notations and properties are introduced in the next definitions.

**Definition 6.** Let  $p = \langle S, s_0, L_{\tau\mathcal{T}}, \rightarrow \rangle$  be a  $TLTS(L)$  with  $s, s', s_i \in S; d, d', e \in \mathbb{R}^{\geq 0}; a_i \in L_{\tau\mathcal{T}}; \beta \in L; \alpha_i \in L_{\mathcal{T}}; \alpha \in L_{\mathcal{T}}^*$ , then

$$\begin{aligned}
s \xrightarrow{a_1 \dots a_n} s' &\triangleq \exists s_0, \dots, s_n : s = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n = s' \\
s \xrightarrow{a_1 \dots a_n} &\triangleq \exists s' : s \xrightarrow{a_1 \dots a_n} s' \\
s \xrightarrow{a_1 \dots a_n} &\triangleq \nexists s' : s \xrightarrow{a_1 \dots a_n} s' \\
s \xrightarrow{\epsilon} s' &\triangleq s = s' \text{ or } s \xrightarrow{\tau \dots \tau} s' \\
s \xrightarrow{\beta} s' &\triangleq \exists s_1, s_2 : s \xrightarrow{\epsilon} s_1 \xrightarrow{\beta} s_2 \xrightarrow{\epsilon} s' \\
s \xrightarrow{d} s' &\triangleq (\exists s_1, s_2 : s \xrightarrow{\epsilon} s_1 \xrightarrow{d} s_2 \xrightarrow{\epsilon} s') \text{ or } (\exists s_1, d', d'' : d' + d'' = d : s \xrightarrow{\epsilon} s_1 \xrightarrow{d'} s' \xrightarrow{d''} s') \\
s \xrightarrow{\alpha_1 \dots \alpha_n} s' &\triangleq \exists s_0 \dots s_n : s = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} s_n = s' \\
s \xrightarrow{\alpha} &\triangleq \exists s' : s \xrightarrow{\alpha} s' \\
s \xrightarrow{\alpha} &\triangleq \nexists s' : s \xrightarrow{\alpha} s'
\end{aligned}$$

Similarly as in the untimed case, we do not always distinguish between a timed labelled transition system and its initial state: if  $p = \langle S, s_0, L_{\tau\mathcal{T}}, \rightarrow \rangle$  we will often identify the process  $p$  with its initial state  $s_0$ , e.g. we write  $p \xrightarrow{\alpha}$  instead of  $s_0 \xrightarrow{\alpha}$ .

**Definition 7.** Let  $p$  be an  $TLTS(L)$ ,  $s$  be a state of  $p$  and  $S$  be a subset of states of  $p$ , then

- $ttraces(p) \triangleq \{\sigma \in L_{\mathcal{T}}^* \mid p \xrightarrow{\sigma}\}$
- $init(s) \triangleq \{a \in L_{\tau\mathcal{T}} \mid s \xrightarrow{a}\}$
- $der(s) \triangleq \{s' \mid \exists \sigma \in L_{\mathcal{T}}^* : s \xrightarrow{\sigma} s'\}$
- $s \text{ after } \sigma \triangleq \{s' \mid s \xrightarrow{\sigma} s'\}$
- $S \text{ after } \sigma \triangleq \bigcup_{s \in S} (s \text{ after } \sigma)$ , where  $S$  is a set of states

- $p$  is deterministic if  $\forall \sigma \in L_{\mathcal{T}}^* : (s \text{ after } \sigma)$  has at most one element. If  $\sigma \in \text{ttraces}(p)$ , then  $(p \text{ after } \sigma)$  is overloaded to denote this element.

As an example, the above definitions applied to the left-hand side machine of Figure 1 give us:  $\text{init}(q_9) = \{tea?, coffee?, bang?, tea!\}$  (remembering that all states are saturated with all inputs),  $\text{der}(q_2) = \{q_2, q_5, q_6, q_9, q_{10}, q_{12}\}$  and  $(q_0 \text{ after } coin?) = \{q_1, q_2\}$ .

In the context of timed systems there are some further important properties.

**Definition 8.** Let  $p = \langle S, s_0, L_{\mathcal{T}}, \rightarrow \rangle$  be a  $TLTS(L)$ , then  $p$  is **time divergent**: if for all state there exists an infinite computation with infinite cumulative delay:

$$\forall s \in S : \exists \sigma \in L_{\mathcal{T}}^\omega : \sigma = a_1 a_2 a_3 \dots : s \xrightarrow{\sigma} \wedge \Sigma \{d_i \mid a_i = d_i\} = \infty$$

$p$  has **Zeno behaviour**: if there exists a state and an infinite computation from it with infinitely many non-delay actions and finite cumulative delay:

$$\begin{aligned} \exists s \in S : \exists \sigma \in L_{\mathcal{T}}^\omega : \sigma = a_1 a_2 a_3 \dots : s \xrightarrow{\sigma} \wedge |\{i \mid a_i \neq d_i\}| = \infty \\ \wedge \Sigma \{d_i \mid a_i = d_i\} < \infty \end{aligned}$$

We assume that for all  $p \in TLTS$  we are working with,  $p$  is **time divergent**, and does not have **Zeno behaviour**.

In case a  $TLTS$  has the labels  $L$  partitioned in subset  $L_I$  and  $L_U$  we called them  $TLTS(L_I \cup L_U)$ . To exclude the possibility that the flow of time in a system can be blocked because the environment does not provide certain input actions, we assume all  $TLTS(L_I \cup L_U)$  are **no forced inputs**. Formally, the **no forced inputs** property is: for all state there exists an infinite computation from it without input actions and with infinite cumulative delay:

$$\forall s \in S : \exists \sigma \in (L_U \cup \{\tau\} \cup \mathcal{T})^\omega : \sigma = a_1 a_2 \dots : s \xrightarrow{\sigma} \wedge \Sigma \{d_i \mid a_i = d_i\} = \infty$$

In order to describe the implementation we introduce timed input-output transition systems ( $TIOTS$ ). A  $TIOTS$  models timed systems in which the set of actions is partitioned into *output actions* and *input actions*, moreover a  $TIOTS$  model is *input enabling*. The *input enabling* assumption is: if an input action is initiated by the environment, the system is always prepared to participate in such an interaction: all the inputs can always be accepted without letting time pass.

**Definition 9.** A *Timed Input-Output Transition System* ( $TIOTS$ ) is a *timed labelled transition system*  $\langle S, s_0, L_{\mathcal{T}}, \rightarrow \rangle$  with  $L$  partitioned into input actions,  $L_I$ , and output actions,  $L_U$ , ( $L_I \cup L_U = L, L_I \cap L_U = \emptyset$ ), that has the property of **weak input enabling**:

$$\forall s \in S : \forall a \in L_I : s \xrightarrow{a}$$

The class of timed input-output transition systems with input actions in  $L_I$  and output actions in  $L_U$  is denoted by  $TIOTS(L_I, L_U) \subseteq TLTS(L_I \cup L_U)$ .

A timed trace  $\sigma$  is a sequence of actions and delays, e.g.  $\sigma = a?d_1d_2b!$ . Obviously, it would be more natural to avoid consecutive delays, as in  $\sigma = a?d_1 + d_2b!$ . Such traces could alternatively be written as sequences of actions with relative time stamps, viz.  $\sigma = a?(0)b!(d_1 + d_2)$ . This idea motivates the definition of *normalized timed traces*.

**Definition 10.** Let  $p$  be a  $TLTS(L_I \cup L_U)$  then we define the normalized timed trace, denoted  $nttraces(p)$ , as

$$nttraces(p) \triangleq \{\sigma \in (\mathcal{T} \cdot L)^* \cdot (\epsilon + \mathcal{T}) \mid p \xrightarrow{\sigma}\}$$

Moreover, we define the function  $\hat{\cdot} : ttraces \rightarrow nttraces$  as the inductive function:

$$\hat{x} = \begin{cases} x & x = d \\ 0x0 & x = a \end{cases} \quad \widehat{xx'\sigma} = \begin{cases} x + x' \cdot \sigma & x = d \wedge x' = d' \\ xx' \cdot \widehat{0\sigma} & x = d \wedge x' = a \\ 0x \cdot x' \widehat{\sigma} & x = a \wedge x' = d \\ 0x0x' \cdot \widehat{\sigma} & x = a \wedge x' = a' \end{cases}$$

For *normalized timed traces*  $\hat{\sigma} = d_0a_0d_1a_1 \cdots d_na_nd_{n+1}$  we also write  $\hat{\sigma} = a_0(d_0)a_1(d_1) \cdots a_n(d_n)d_{n+1}$ . If a timed trace begins with an action it can always be converted to a *normalized timed trace* by combining delays, and adding zero delays 0 in the appropriate places. In a similar way it is possible to convert all *timed trace* to a *normalized time trace*, this result is shown in Lemma 1.

**Lemma 1.** Let  $p$  be a  $TLTS(L_I \cup L_U)$ , then

$$\forall \sigma : \sigma \in ttraces(p) \text{ if and only if } \hat{\sigma} \in nttraces(p)$$

*Proof.*

[ $\Leftarrow$ ] Direct. If  $\hat{\sigma} \in nttraces(p)$  then  $\hat{\sigma} \in ttraces(p)$ .

[ $\Rightarrow$ ] By induction in  $|\sigma|$ . Let  $|\sigma| = 1$ , then

If  $\sigma \in nttraces(p)$ . Done

If  $\sigma \notin nttraces(p)$  then  $\sigma = a$ . Then  $\hat{\sigma} = 0 \cdot a \cdot 0$  or  $\hat{\sigma} = a(0) \cdot 0$ .

Suppose  $\forall \sigma : |\sigma| \leq n : \exists \hat{\sigma} : \hat{\sigma} \in nttraces(p)$ .

Let  $|\sigma| = n + 1$  then  $\sigma = x \cdot \sigma'$  and  $|\sigma'| \leq n$  and for inductive hypothesis exists

$\hat{\sigma}' = d_1a_1 \cdots d_ka_kd_{k+1}$ . Then,

If  $x = d_0$  then  $\hat{\sigma} = (d_0 + d_1)a_1 \cdots d_ka_kd_{k+1}$  or  $\hat{\sigma} = a_1(d_0 + d_1)a_k(d_k)d_{k+1}$

If  $x = a_0$  then  $\hat{\sigma} = 0a_0d_1a_1 \cdots d_ka_kd_{k+1}$  or  $\hat{\sigma} = a_0(0)a_1(d_1) \cdots a_k(d_k)d_{k+1}$

□

From now on we will not distinguish between a timed trace  $\sigma$  and its normalization  $\hat{\sigma}$ .

Similarly to Tretmans' work, we proceed to introduce the notion of *quiescence* in the timed setting. In the presence of time we define a *quiescent* state as one where the system is unable to produce an output immediately or in the future without receiving further input stimuli.

**Definition 11.** Let  $p$  be a  $TLTS(L_I \cup L_U)$ . A state  $s$  of  $p$  is *quiescent*, denoted by  $\delta(s)$ , if and only if

$$\forall a \in L_U : \forall d \in \mathbb{R}^{\geq 0} : s \not\xrightarrow{a(d)}$$

As before in the untimed case, we can start out by representing *quiescence* as a special action  $\delta$  ( $\delta \notin L \cup \{\tau\}$ )<sup>2</sup>, and extending the timed transition relation of  $p$  a  $TLTS$  to include self-loop transitions  $s \xrightarrow{\delta} s$  if and only if  $s$  is a *quiescent* state. Moreover, let  $\Delta(p)$  denotes the extended timed transition system of  $p$  that is obtained in this way.

<sup>2</sup> In [Tre96] the action symbol  $\theta$  is used for the observation of *quiescence*. We prefer to use  $\delta$  for both *quiescence* and its observation, in line with the philosophy that identical actions synchronize.

### 3.2 Timed implementation relations

The extension of the timed transition relation allows us to define the following relation over implementation  $q$  in  $TIOTS(L_I, L_U)$  and specification  $p$  in  $TLTS(L_I \cup L_U)$  as inclusion of  $nttraces$ .

**Definition 12.**

Let  $q$  be a  $TIOTS(L_I, L_U)$  and  $p$  be a  $TLTS(L_I \cup L_U)$ , then

$$q \sqsubseteq_{tiorf} p \text{ if and only if } nttraces(\Delta(q)) \subseteq nttraces(\Delta(p))$$

For specifications  $p \in TLTS$  *quiescent* states can, in principle, be identified by analyzing the timed transition system, i.e. we can assume that  $\Delta(p)$  is at our disposal. However, given our black box framework, for implementations  $q \in TIOTS$  we only can detect *quiescence* by waiting for outputs. But we cannot wait forever, and therefore need to choose a maximal duration  $M$ , which is the time need to recognize that we are in a *quiescent* state. This motivates the following parameterized version of *tiorf*, where  $\delta$  can only appear after  $M$  time-units.

**Definition 13.**

Let  $q$  be a  $TIOTS(L_I, L_U)$  and  $p$  be a  $TLTS(L_I \cup L_U)$ , then

$$q \sqsubseteq_{tiorf}^M p \text{ if and only if } \Delta_M(q) \subseteq \Delta_M(p)$$

where for any  $r \in TLTS$ :  $\Delta_M(r) \triangleq nttraces(\Delta(r)) \cap (\mathcal{T} \cdot L \cup M \cdot \delta)^* \cdot (\epsilon + \mathcal{T})$ .

The above definition takes only into account observations of *quiescence* that are made after a delay of  $M$  time units.

In contrast to the untimed case, time delays can change the system state, which has interesting consequences, as shown by the modification of the quirky coffee machine example of Figure 1.

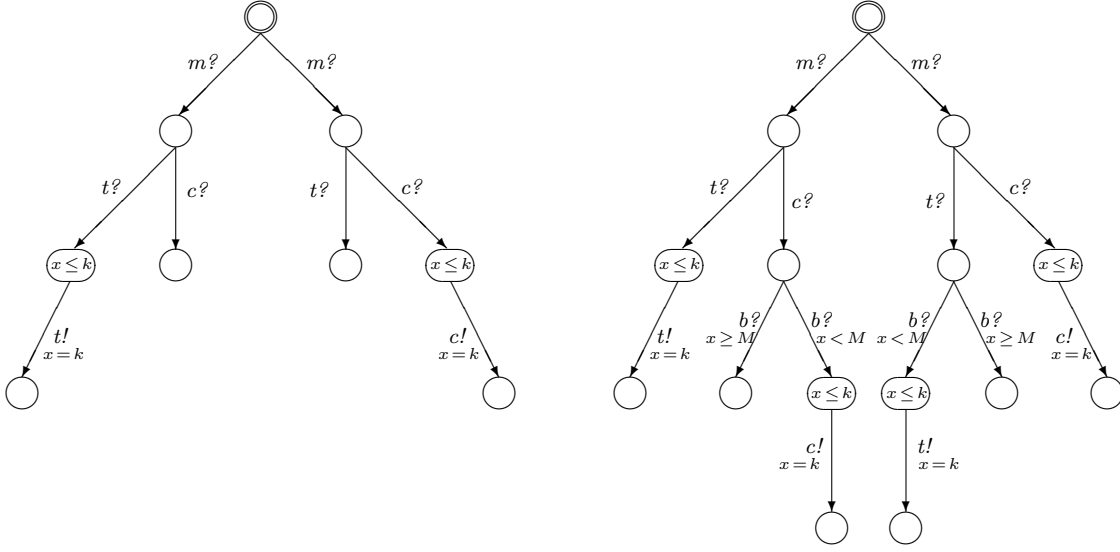
*Example 2.* Figure 2 shows two quirky coffee machines with time. Again, suppose both graphs are saturated with input action transitions in each state by adding input self-loops for all input transitions that are not explicitly given. Note that here after introducing *money?* we can switch between the coffee and tea modes. If we order *coffee?* and *bang?* fast enough we always will have coffee in the right-hand machine and some times in the left-hand machine, but if we *bang?* after waiting for the *quiescence* we will not notice the difference between machines. It follows from the one that cannot switch modes. This is a consequence of the fact that observing *quiescence* takes time. For simplicity, in the figure, we use  $m?$  for money,  $b?$  for bang,  $c?$ ,  $c!$  for coffee, and  $t?$ ,  $t!$  for tea. We also suppose that each action resets the clock  $x$  and that  $k < M$ . We used timed automata representation.

The output set of a given state of a system in  $TLTS(L_I \cup L_U)$  consists of the time stamped output actions that are allowed from that state (abstracting from  $\tau$ -actions), including  $\delta$ -actions after a delay of  $M$  time units.

**Definition 14.** Let  $s$  be a state of an (extended) timed transition system  $p$  in  $TLTS(L_I \cup L_U)$ , then:

$$out_M(s) \triangleq \{a(d) \mid a \in L_U \wedge s \xrightarrow{a(d)}\} \cup \{\delta(M) \mid s \xrightarrow{\delta(M)}\}$$

and for  $S$  a set of states, then:



**Fig. 2.** The quirky coffee machine with time

$$out_M(S) \triangleq \bigcup_{s \in S} out_M(s)$$

**Lemma 2.** Let  $q$  be a  $TIOTS(L_I, L_U)$  and  $p$  be a  $TLTS(L_I \cup L_U)$ , then  
 $q \sqsubseteq_{tiorf}^M p$  if and only if  $\forall \sigma \in (\mathcal{T} \cdot L \cup M \cdot \delta)^* \cdot (\epsilon + \mathcal{T})$ :

$$out_M(\Delta(q) \text{ after } \sigma) \subseteq out_M(\Delta(p) \text{ after } \sigma)$$

*Proof.*

[ $\Rightarrow$ ] Let  $\sigma \in (\mathcal{T} \cdot L \cup M \cdot \delta)^* \cdot (\epsilon + \mathcal{T})$ , then

if  $\sigma \notin ntraces(\Delta(q))$ , then  $out_M(\Delta(q) \text{ after } \sigma) = \emptyset$

if  $\sigma \in ntraces(\Delta(q))$ , then

$\forall a \in out_M(\Delta(q) \text{ after } \sigma) : \sigma \cdot a \in \Delta_M(q)$  and  $\sigma \cdot a \in \Delta_M(p)$ , then  
 $a \in out_M(\Delta(p) \text{ after } \sigma)$

[ $\Leftarrow$ ] Let  $\sigma \in \Delta_M(q)$ , i.e.,  $\sigma \in ntraces(\Delta(q)) \cap (\mathcal{T} \cdot L \cup M \cdot \delta)^* \cdot (\epsilon + \mathcal{T})$

then,  $\Delta(q) \text{ after } \sigma \neq \emptyset$

using the *no forced input* property:  $out_M(\Delta(q) \text{ after } \sigma) \neq \emptyset$

then,  $out_M(\Delta(p) \text{ after } \sigma) \neq \emptyset$

which tell us that:  $\sigma \in ntraces(\Delta(p)) \wedge \sigma \in (\mathcal{T} \cdot L \cup M \cdot \delta)^* \cdot (\epsilon + \mathcal{T})$

then,  $\sigma \in \Delta_M(p)$

□

Finally, we are in position to define the relation we use to test real time systems:  $\mathbf{tioco}_M$ . For  $q \in TIOTS(L_I, L_U)$  and  $p \in TLTS(L_I \cup L_U)$ ,  $q$  will be  $\mathbf{tioco}_M$  to  $p$  if the set of outputs of  $q$  after every normalized timed trace  $\sigma$  of  $p$  including observations  $\delta(M)$ , is a subset of the outputs of  $p$  after the same timed trace  $\sigma$ .

**Definition 15.** Let  $q$  be a  $TIOTS(L_I, L_U)$  and  $p$  be a  $TLTS(L_I \cup L_U)$ , then

$$q \sqsubseteq_{tioco}^M p \text{ if and only if } \forall \sigma \in \Delta_M(p) : out_M(\Delta(q) \text{ after } \sigma) \subseteq out_M(\Delta(p) \text{ after } \sigma)$$

we also write  $\sqsubseteq_{tioco}^M$  as  $\mathbf{tioco}_M$ .

### 3.3 An operational model

To obtain an effective theory of *quiescence* in a timed setting we need more than stipulating that observing *quiescence* takes time. Since with physical implementations we can only observe absence of outputs over finite time intervals we must stipulate when such observations will be interpreted as *quiescence*.

**Definition 16.** Let  $q$  be a  $TLOTS(L_I, L_U)$  and  $M \in \mathbb{R}^{\geq 0}$ , then

- a state  $s$  of  $q$  is  $M$ -quiescent if and only if  $\forall s' \in (s \text{ after } M) : s'$  is quiescent
- $q$  is  $M$ -quiescent if and only if all states  $s$  of  $q$  are  $M$ -quiescent.

In line with the above development we now want to formalize how *normalized timed traces* of  $TLOTS$ 's may be enriched directly with  $\delta$ -actions. Whenever the *normalized timed trace* allows an action with a delay of more than  $M$  time-units this creates a possibility to observe *quiescence*. For example, if a system is  $M$ -quiescent and  $M = 4$  with  $\sigma = a?(2)b?(5)c!(3)$  as an observed timed trace then it is also possible to observe  $\sigma' = a?(2)\delta(4)b?(1)c!(3)$ . We formalize the addition of  $\delta$ -observations to *normalized timed traces* as a formal relation  $\delta_M$  between (extended) *normalized timed traces*.

**Definition 17.** Let  $\sigma, \sigma'$  be normal form of  $\sigma, \sigma' \in (\mathcal{T} \cdot (L \cup \delta))^* \cdot (\epsilon + \mathcal{T})$ , then

- $\sigma \delta_M \sigma'$  if and only if

$$\exists \sigma_1, \sigma_2 : \exists a : \exists d \geq M : \sigma = \sigma_1 \cdot a(d) \cdot \sigma_2 \wedge \sigma' = \sigma_1 \delta(M) a(d - M) \sigma_2$$

- let  $\Sigma$  be a set of normalized timed traces, then

$$\delta_M(\Sigma) = \text{pref} \left( \bigcup_{\sigma \in \Sigma} \{ \sigma' \mid \sigma \delta_M^* \sigma' \} \right)$$

where  $\text{pref}(S)$  is interpreted as the prefix-closure of a set of traces  $S$  and  $\delta_M^*$  is the reflexive transitive closure of the relation  $\delta_M$ .

If  $\delta$ -actions are introduced in *normalized timed traces* on the basis observations of delays of (at least)  $M$  time units, we must check for consistency, i.e. we must have the following property: “after a  $\delta(M)$  never appears and output action”, this is express in the next lemma.

**Lemma 3.** Let  $q$  be a  $M$ -quiescent  $TLOTS(L_I, L_U)$ , then

$$\nexists \sigma \in \delta_M(\text{nttraces}(q)) : \exists \sigma' \in \delta_M(\text{nttraces}(q)) : \exists a \in L_U : \sigma = \sigma' \delta(M) a(d)$$

*Proof.* Suppose that such a  $\sigma = \sigma' \delta(M) a(d)$  with  $a \in L_U$  does exist. Then  $\sigma' a(d+M)$  is a trace of the system too. It follows there are reachable states  $s' \in (q \text{ after } \sigma')$  and  $s'' \in (s' \text{ after } M)$  with  $s'' \xrightarrow{a(d)}$ , i.e.  $s''$  is not quiescent. Contradiction.  $\square$

**Lemma 4.** Let  $q$  be a  $M$ -quiescent  $TLOTS(L_I, L_U)$  with  $L_I \neq \emptyset$ , then

$$\delta_M(\text{nttraces}(q)) = \Delta_M(q)$$

*Proof.*

[ $\Rightarrow$ ] This proof is carried out by induction of the number of  $\delta(M)$  occurrences in the traces of  $q$ . It is obvious that for  $\delta$ -free traces  $\sigma \in \delta_M(\text{nttraces}(q))$  if and only if  $\sigma \in \text{nttraces}(q)$  if and only if  $\sigma \in \Delta_M(q)$ . Now suppose that  $\sigma$  is of the form  $\sigma_1\delta(M)a(d)\sigma_2$ , with  $n$  occurrences of  $\delta(M)$  in  $\sigma_1$  and  $\sigma_2$   $\delta$ -free. We will prove that such  $\sigma$  are in  $\delta_M(\text{nttraces}(q))$  if and only if they are  $\Delta_M(q)$ .

$$\begin{aligned} & \sigma_1\delta(M)a(d)\sigma_2 \in \delta_M(\text{nttraces}(q)) \\ \text{(by def. of } \delta_M) & \Leftrightarrow \sigma_1a(d+M)\sigma_2 \in \delta_M(\text{nttraces}(q)) \\ \text{(ind.hyp. on } \sigma_1) & \Leftrightarrow \sigma_1a(d+M)\sigma_2 \in \Delta_M(q) \\ \text{(skip } \delta\text{-observation)} & \Leftarrow \sigma_1\delta(M)a(d)\sigma_2 \in \Delta_M(q) \end{aligned}$$

Clearly, we have that  $\sigma \in \Delta_M(q)$  implies  $\sigma \in \delta_M(\text{nttraces}(q))$ .

[ $\Leftarrow$ ] Let  $\sigma \in \delta_M(\text{nttraces}(q))$ , then by the above implications that  $\sigma = \sigma_1a(d+M)\sigma_2$  is in  $\Delta_M(q)$ . Now suppose that  $\sigma_1\delta(M)a(d)\sigma_2 \notin \Delta_M(q)$ . This means that there is a non-quiescent state in  $(q \text{ after } \sigma_1M)$ , i.e. there is an output action  $a \in L_U$  with  $\sigma_1a(d'+M) \in \Delta_M(q)$ . Again, by the above implication we have  $\sigma_1\delta(M)a(d') \in \delta_M(\text{nttraces}(q))$ . Because of the preceding lemma, this contradicts the  $M$ -quiescence of  $q$ .

If  $\sigma$  is not of the required format, it must be of the form  $\sigma_1\delta(M)$ , and by input-enabledness we can extend it to the form  $\sigma_1\delta(M)a(0)$  by appending some input action  $a$ . The above result, together with the prefix-closure of  $\delta_M(\text{nttraces}(q))$  and  $\Delta_M(q)$ , implies that these sets also coincide for such traces, i.e. the corollary holds.  $\square$

This corollary means that if an implementation  $q$  can be assumed to be  $M$ -quiescent we may use the set of enriched observations  $\delta_M(\text{nttraces}(q))$  to obtain  $\Delta_M(q)$ , whose definition is based on the unobservable timed transition system  $\Delta(q)$ . This will be the basis for our test derivation algorithm.

As a final property, before present the derivation algorithm we present Lemma 5. This lemma reveals that the  $\mathbf{tioco}_M$  relation, in presence of  $M$ -quiescent implementation, implies a pre-order.

**Lemma 5.** *Let  $q$  be a  $M_1$ -quiescent  $TIOTS(L_I, L_U)$ ,  $p$  be a  $TLTS(L_I \cup L_U)$  and  $M_1 < M_2$ , then*

$$\text{if } q \sqsubseteq_{\mathbf{tiorf}}^{M_1} p \text{ then } q \sqsubseteq_{\mathbf{tiorf}}^{M_2} p$$

*Proof.* Assume  $q \sqsubseteq_{\mathbf{tiorf}}^{M_1} p$  and let  $\sigma$  be a trace in  $\Delta_{M_2}(q)$ .

Transform  $\sigma$  into  $\sigma'$  by replacing every subsequence of  $\sigma$  of the form  $\delta(M_2)a(d)$  by  $\delta(M_1)a(d+(M_2-M_1))$ , and possibly  $\delta(M_2)$  by  $\delta(M_1)(M_2-M_1)$  in case  $\delta(M_2)$  is the last action of  $\sigma$ . Moreover, we saturated the trace  $\sigma$  with  $\delta(M_1)$  in case exists in it a time stamp  $d$  bigger than  $M_1$ , recursively.

$\sigma'$  must also be a trace of  $\Delta(q)$ , and therefore of  $\Delta_{M_1}(q)$ .

As  $q \sqsubseteq_{\mathbf{tiorf}}^{M_1} p$ , it follows that  $\sigma'$  is a trace in  $\Delta_{M_1}(p)$ .

By postponing all observations of  $\delta$  by  $M_2 - M_1$  we get  $\sigma \in \Delta_{M_2}(p)$ ,

i.e.  $\Delta_{M_2}(q) \subseteq \Delta_{M_2}(p)$  then  $q \sqsubseteq_{\mathbf{tiorf}}^{M_2} p$ .  $\square$

## 4 A real-time test generation framework

In this section we define the concept of real-time test cases, the nature of their execution, and the evaluation of their success or failure.

**Definition 18.**

- A test case  $t$  is a TLTS  $\langle S, s_0, L_T \cup \{\delta\}, \rightarrow \rangle$  such that
  - $t$  is deterministic and has bounded behaviour, i.e.  $\exists N > 0 : \forall \sigma = a_1 a_2 a_3 \dots : |\{i \mid a_i \neq d_i\}| < \infty$  and  $\sum \{d_i \mid a_i = d_i\} < N$
  - $S$  contains the terminal states **pass** and **fail**, with  $\text{init}(\mathbf{pass})$  and  $\text{init}(\mathbf{fail})$  without outgoing transitions except self loops allowing time pass
  - for any state  $t' \in S$  of the test case with  $t' \neq \mathbf{pass}, \mathbf{fail}$ ,  $\exists 0 \leq d$  with

$\text{init}(t' \text{ after } d') = L_U \cup \{e \mid e = d - d'\}$  for all  $d' < d$ , or

$\text{init}(t' \text{ after } d) = \{a\}$  with  $a \in L_I$  or  $a = \delta$

- $t$  does not have  $\tau$ -transitions

The class of test cases over  $L_I$  and  $L_U$  is denoted by  $\mathcal{TTEST}(L_I, L_U)$

but we represent it similarly as a timed automata, only for simplify the notation

- A test suite  $\mathbf{T}$  is a set of test cases:  $\mathbf{T} \subseteq \mathcal{TTEST}(L_I, L_U)$ .

A test run of an implementation with a test case is modelled by the synchronous parallel execution of the test case with the implementation under test. This run continues until no more interactions are possible, i.e. until a deadlock occurs.

**Definition 19.** Let  $t$  be a  $\mathcal{TTEST}(L_I, L_U)$  and  $q$  be a  $M$ -quiescent  $\text{TLOTS}(L_I, L_U)$ , then

- Running a test case  $t$  with an implementation  $q$  is modelled by the parallel operator  $\parallel : \mathcal{TTEST}(L_I, L_U) \times \text{TLOTS}(L_I, L_U) \rightarrow \text{TLOTS}(L_I, L_U)$  which is defined by the following inference rules:

$$\begin{array}{lcl}
 q \xrightarrow{\tau} q' & \vdash & t \parallel q \xrightarrow{\tau} t \parallel q' \\
 t \xrightarrow{\delta} t' & \vdash & t \parallel q \xrightarrow{\delta} t' \parallel q \\
 t \xrightarrow{a} t', q \xrightarrow{a} q', a \in L & \vdash & t \parallel q \xrightarrow{a} t' \parallel q' \\
 t \xrightarrow{d} t', q \xrightarrow{d} q' & \vdash & t \parallel q \xrightarrow{d} t' \parallel q'
 \end{array}$$

- A test run of  $t$  with  $q$ , is a  $\sigma \in \Delta_M$  of  $t \parallel q$  leading to a terminal state of  $t : \sigma$  is a test run of  $t$  and  $q \stackrel{\sigma}{\Rightarrow}$

$$\exists q' : (t \parallel q \stackrel{\sigma}{\Rightarrow} \mathbf{pass} \parallel q') \text{ or } (t \parallel q \stackrel{\sigma}{\Rightarrow} \mathbf{fail} \parallel q')$$

- An implementation  $q$  **passes** test case  $t$ , if all their test runs lead to the **pass** state of  $t$ :

$$q \text{ passes } t \stackrel{\text{def}}{\triangleq} \forall \sigma \in \Delta_M : \forall q' : t \parallel q \not\stackrel{\sigma}{\Rightarrow} \mathbf{fail} \parallel q'$$

- An implementation  $q$  **passes** a test suite  $\mathbf{T}$ , if it **passes** all test cases in  $\mathbf{T}$ :

$$q \text{ passes } \mathbf{T} \stackrel{\text{def}}{\triangleq} \forall t \in \mathbf{T} : q \text{ passes } t$$

If  $q$  does not pass the test suite, it fails if:

$$q \text{ fails } \mathbf{T} \stackrel{\text{def}}{\triangleq} \exists t \in \mathbf{T} : q \text{ passes } t$$

Since an implementation can behave nondeterministically, different test runs of the same test case with the same implementation may lead to different terminal states and hence to different verdicts. An implementation **passes** a test case if and only if all possible test runs lead to the verdict **pass**.

#### 4.1 Nondeterministic test case construction

For the description of test cases we use, as we already did before, a process-algebraic behaviour notation with a syntax inspired by LOTOS [ISO89]:

$$B \triangleq a;B \mid B + B \mid \Sigma \mathcal{B}$$

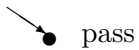
where  $a \in L_{\mathcal{T}}$ ,  $\mathcal{B}$  is a countable set of behaviour expressions, and the axioms and the inference rules are:

$$\begin{array}{ll} a \in L & \vdash a;B \xrightarrow{a} B \\ a = d, d' < d & \vdash a;B \xrightarrow{d'} d - d'; B \\ a = d & \vdash a;B \xrightarrow{d} B \\ B_1 \xrightarrow{a} B'_1, a \in L_{\mathcal{T}} & \vdash B_1 + B_2 \xrightarrow{a} B'_1 \\ B_2 \xrightarrow{a} B'_2, a \in L_{\mathcal{T}} & \vdash B_1 + B_2 \xrightarrow{a} B'_2 \\ B \xrightarrow{a} B', B \in \mathcal{B}, a \in L_{\mathcal{T}} & \vdash \Sigma \mathcal{B} \xrightarrow{a} B' \end{array}$$

Moreover, we use  $a(d)$  as syntactic sugar for  $d; a$ .

**Test case generation procedure** We define a procedure to generate test cases from a given specification timed transition system. Similar to [Tre96] test cases result from the non-deterministic, recursive application of three test generation steps, corresponding to: (1) termination, (2) generation of an input, and (3) observation of output (including *quiescence*). It should be noted that the construction steps involve (negations of) predicates of the form  $a(d) \in out_M(S)$ , which on the general level of timed input-output transition systems are undecidable. The procedure given here, therefore, should be seen as a meta-algorithm that can be used to generate tests effectively for subclasses of *TIOTS* for which these predicates are decidable, such as timed automata [KT04,LMN03].

##### 1. termination

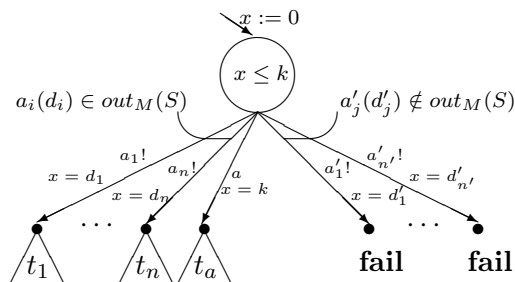


$t := \mathbf{pass}$

The single state test case **pass** is always a sound test case. It stops the recursion in the algorithm, and thus terminates the test case.

##### 2. inputs

choose  $k \in [0, M)$  and  $a \in L_I$

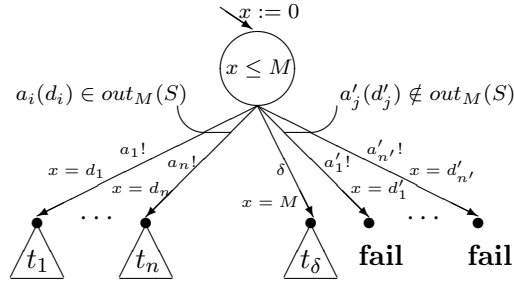


$$\begin{aligned}
t := & \Sigma \{a_i(d_i); t_i \mid a_i \in L_U \wedge d_i < k \wedge a_i(d_i) \in \text{out}_M(S)\} \\
& + a(k); t_a \\
& + \Sigma \{a'_j(d'_j); \mathbf{fail} \mid a'_j \in L_U \wedge d'_j < k \wedge a'_j(d'_j) \notin \text{out}_M(S)\}
\end{aligned}$$

where  $x$  is a clock,  $k$  is a timed variable and  $t_i$  and  $t_a$  are obtained by recursively applying the algorithm for  $(S \text{ after } a_i(d_i))$  and  $(S \text{ after } a(k))$ , respectively.

Test case  $t$  is waiting for  $k$  time-units and treating to make and input  $(a)$ . If an output arrives from the implementation it checks; if it is an invalid response, i.e.  $a'_j(d'_j) \notin \text{out}_M(S)$  then the test case terminates in **fail**; if it is a valid response after the timed pass then the test case continues recursively. If the time pass then the test makes the input  $(a)$  and continues recursively.

### 3. waiting for outputs



$$\begin{aligned}
t := & \Sigma \{a_i(d_i); t_i \mid a_i \in L_U \wedge d_i < M \wedge a_i(d_i) \in \text{out}_M(S)\} \\
& + \Sigma \{\delta(M); t_\delta \mid \delta \in \text{out}_M(S \text{ after } M)\} \\
& + \Sigma \{\delta(M); \mathbf{fail} \mid \delta \notin \text{out}_M(S \text{ after } M)\} \\
& + \Sigma \{a'_j(d'_j); \mathbf{fail} \mid a'_j \in L_U \wedge d'_j < M \wedge a'_j(d'_j) \notin \text{out}_M(S)\}
\end{aligned}$$

where  $x$  is a clock and  $t_i$  and  $t_\delta$  are obtained by recursively applying the algorithm for  $(S \text{ after } a_i(d_i))$  and  $(S \text{ after } M)$ , respectively.

Test case  $t$  is waiting for  $M$  time-units if an output arrives from the implementation it checks; if it is an invalid response, i.e.  $a'_j(d'_j) \notin \text{out}_M(S)$  then the test case terminates in **fail**; if it is a valid response after the timed pass then the test case continues recursively. The observation of *quiescence*  $\delta$  is treated separately, using the constant  $M$  given by the  $M$ -quiescent property.

**Soundness** The test generation procedure presented is sound with respect to the **tioco<sub>M</sub>** relation. This property is shown in Theorem 1.

**Definition 20.** Let  $p$  be a  $TLTS(L_I \cup L_U)$ , then

$$\sigma \in \Delta_M(p) \text{ is } \delta(M)\text{-saturated if and only if for all } \sigma' \text{ with } \sigma \delta_M \sigma' \text{ we have } \sigma = \sigma'$$

**Proposition 1.** If  $\sigma$  is  $\delta(M)$ -saturated, then for all  $a(d) \neq \delta(M)$  occurring in  $\sigma$  we have  $d < M$ .

*Proof.* Directly from the definition of the  $\delta_M$ -relation.  $\square$

**Theorem 1.** *Let  $p$  be a  $TLTS(L_I \cup L_U)$ , then for all  $M$ -quiescent  $q$  in  $TIOTS(L_I, L_U)$  and all test cases  $t$  obtained from  $spec$  by the above procedure:*

$$q \mathbf{tioco}_M p \Rightarrow q \mathbf{passes} t$$

*Proof.* Let  $q$  be  $M$ -quiescent with  $q \mathbf{tioco}_M spec$ , then we will show that for all  $\sigma \in \Delta_M(spec)$  and all test cases  $t$  generated by the procedure from  $spec$ : if  $(t||q \xrightarrow{\sigma} t'||q')$  then  $(t' \neq \mathbf{fail})$ . Without loss of generality we can assume that  $\sigma$  is  $\delta(M)$ -saturated.

By induction on the length of  $\sigma$ :

- if  $\sigma = \epsilon$  and  $t||q \xrightarrow{\epsilon} t'||q'$ 
  - if  $t$  was constructed using case 1 in the first step, then  $t||q \xrightarrow{\epsilon} \mathbf{pass}||q'$
  - if  $t$  was constructed using cases 2 or 3 in the first step, then  $t = t' \neq \mathbf{fail}$  and all derivations of  $\xrightarrow{\epsilon}$  have the form:  $t||q \xrightarrow{\epsilon} t||q'$
- if  $\sigma = \sigma' \cdot a$  and  $t||q \xrightarrow{\sigma'} t''||q'' \xrightarrow{a} t'||q' \wedge a = a(d)$ , because  $t$  can do  $a$  there are only two possibilities to construct  $t'$ :
  - from case 2:  $a = a(d) \wedge a \in L_I \wedge d < M$ , then  $t||q \xrightarrow{\sigma} t'||q' \wedge t' \neq \mathbf{fail}$
  - from case 3:  $a = a(d) \wedge a \in out_M(\Delta(q) \mathbf{after} \sigma)$ , then because  $(q \mathbf{tioco}_M spec)$ :  $a(d) \in out_M(\Delta(spec) \mathbf{after} \sigma)$ , and thus  $t||q \xrightarrow{\sigma} t'||q' \wedge t' \neq \mathbf{fail}$ .

$\square$

**Exhaustiveness** The test generation procedure is also exhaustive in the sense that for each non-conforming implementation a test case can be generated that detects the non-conformance.

**Lemma 6.** *Let  $p$  be a  $TLTS(L_I \cup L_U)$ ,  $\sigma \in \Delta_M(p)$   $\delta(M)$ -saturated, and  $t'$  a test case generated by the procedure for  $(\Delta(p) \mathbf{after} \sigma)$  then there exists a test case  $t$  generated from  $p$  with  $t \xrightarrow{\sigma} t'$ .*

*Proof.* By induction on the length of  $\sigma$ :

- $|\sigma| = 0$  then take  $t = t'$
- suppose  $t$  exists for all  $\sigma$  with length  $n$  and let  $\sigma = \sigma'x$  and  $x = a(d)$ 
  - if  $a \in L_I$ , using case 2 for the input  $a$ :  $t \xrightarrow{\sigma'} t'' \xrightarrow{x} t'$
  - if  $a \in out_M(\Delta(spec) \mathbf{after} \sigma)$ , using case 3:  $t \xrightarrow{\sigma'} t'' \xrightarrow{x} t'$ .

$\square$

**Theorem 2.** *Let  $p$  be a  $TLTS(L_I \cup L_U)$ , then for all  $M$ -quiescent  $q$  in  $TIOTS(L_I, L_U)$  with  $q \mathbf{tioco}_M p$ , there exists a test case  $t$  generated from  $p$  by the procedure such that:*

$$q \mathbf{passes} t$$

*Proof.* If  $q$  **tioco**<sub>M</sub>  $p$  then there exists  $\sigma \in \Delta_M(p)$  :  
 $out_M(\Delta(q) \text{ after } \sigma) \not\subseteq out_M(\Delta(p) \text{ after } \sigma)$ .

Without loss of generality we can assume that  $\sigma$  is  $\delta(M)$ -saturated.

Then let  $a \in out_M(\Delta(q) \text{ after } \sigma) \setminus out_M(\Delta(p) \text{ after } \sigma)$  and  $q \xrightarrow{\sigma} q' \xrightarrow{a} q''$ .

Let  $t'$  be the result of applying case 3 of the procedure to  $(\Delta(p) \text{ after } \sigma)$ , and let  $t$  be the test case constructed out of  $t'$  and  $\sigma$  by the above lemma.

Because  $a \notin out_M(\Delta(p) \text{ after } \sigma)$ :  $t \parallel q \xrightarrow{\sigma \cdot a} \text{fail} \parallel q''$ ,  $q$  **passes**  $t$ .

□

The exhaustiveness of our test generation procedure is less useful than the corresponding result in the untimed case. There, it implies that the test generation algorithm, if repeatedly executed in a fair non-terminating manner, will generate all test cases in the limit, and therefore, in the limit, achieve full coverage with respect to **ioco** and the given specification *spec*.

Here, the number of potential test cases is uncountable because of the underlying continuous model of time, and no countable repetition of test generations suffices. It is possible, however, to obtain a version of the stronger form of exhaustiveness for real-time test generation as well by considering equivalence classes of (minimal) error traces. It can be shown that reasonable assumptions of our test generation procedure will hit each such equivalence class in the limit. This result will be reported in detail in a forthcoming publication.

## 5 Example

In the setting of timed automata, deciding the predicate  $o_i(d_i) \in out_M(S)$  amounts to reachability analysis. For the simpler version of **tioco** based on timed trace inclusion (i.e. excluding *quiescence*) this has already been implemented in the tool environment IF [KT04], the UPPAAL-based testing tool TUPPAAL, and a real-time extension of TORX. We present an example of our test case generation based on a timed automaton model of a coffee machine, similar to the previous one, but with infinite behaviour due to cycles.

*Example 3.* Figure 3 shows two quirky coffee machines with time. The first one is a specification and the second one is a wrong implementation. To the right, there is a test case derived by the algorithm that can detect the error in the implementation. We suppose both machines are saturated with all input actions in each state. In the specification we show the  $\delta$ -transitions, while in the implementation we detect them using  $M = k$ . We assume that  $k > 1$ .

The problem appears because:

$$out(\text{spec after } m?(1)c?(1)\delta(k)b?(1)c?(1)) = \{c![0, \infty)\}$$

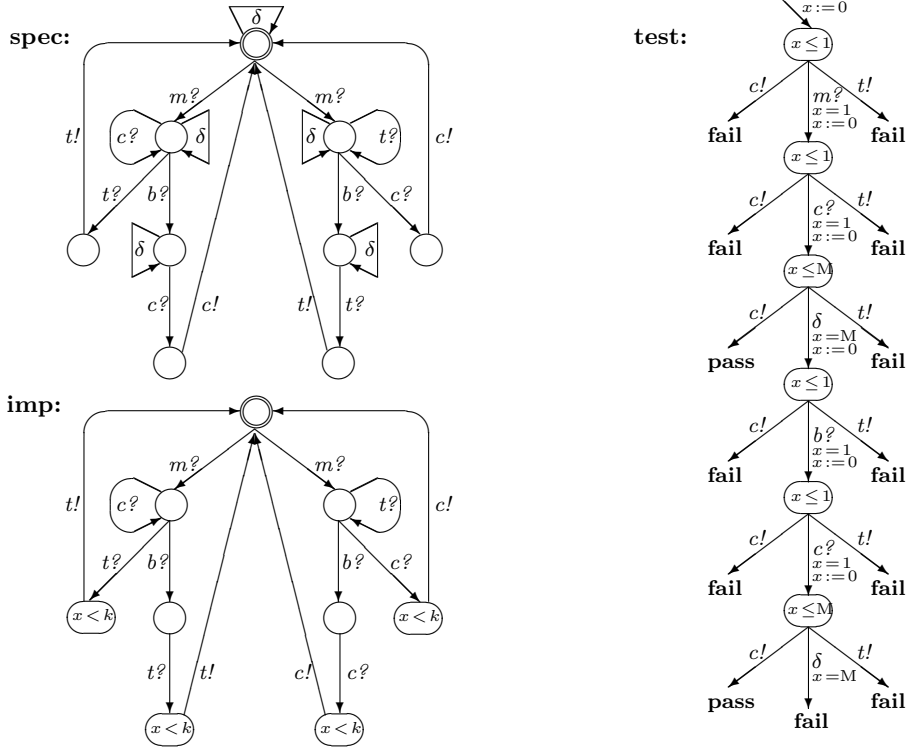
and

$$out(\text{imp after } m?(1)c?(1)\delta(k)b?(1)c?(1)) = \{\delta(k)\}$$

where we use the notation  $c![0, \infty)$  to denote that the output  $c!$  can be at any time between 0 and  $\infty$ .

## 6 Related work

As already indicated before this work is closely related to work carried out by Krichen et al. in [KT04], and closely related work by Larsen et al. [LMN03], who deal with a *quiescence*-free



**Fig. 3.** A specification of a quirky coffee machine with time, an implementation with  $M = k$ , and a test case derived from the specification.

interpretation of timed **io**co based on timed trace inclusion for timed automata. Our work shows how such results may be extended to deal with *quiescence*, and provides a general framework at the level of timed transition systems.

Previous attempts of extending testing with time include older work by Nielsen et al. in [NS01], for testing a subclass of timed automata called event-recording automata (ERA). The technique is based on the symbolic analysis of timed automata inspired by the UPPAAL model-checker, but lacks a suitable notion of implementation relation. Springintveld et al. in [SVD01] present an exhaustive testing method for deterministic timed automata with dense time, using the notion of a grid automaton that represents each clock region with a finite set of clock valuations. Although being exact, the grid method is impractical because it generates “an astronomically large number of test sequences” [SVD01]. Cardell-Oliver presents a method for networks of deterministic timed automata extended with integer data variables [CO02], where only a part of the system is visibly using test views, so that a test is never exhaustive.

Several authors have tried to obtain good specification coverage for their test methods by adapting transition-tour methods from classical FSM-based testing [ENDK98,HNTC99].

Clarke and Lee [CL97] use the algebra of communicating shared resources (ACSR) on a discrete time base. ACSR allows non-deterministic specifications, the use of internal events and priorities. For testing, only boundary points of the time domain are selected. Cleaveland et al. propose a testing method for probabilistic processes on a discrete model of time [CLLS96] that bears a close resemblance to the classical testing theory of Hennessy and De Nicola [NH83]. Mandrioli et al. use temporal logic with arithmetic on a discrete time base [MMM95].

## 7 Conclusion and future work

In this paper we have presented an extension of Tretmans’ **ioco** theory and algorithm for test generation for input-output transition systems to real-time systems. Our treatment is based on an operational interpretation of the notion of *quiescence* that gives rise to a family of implementation relations parameterized by observation durations  $M$  for *quiescence*. These relations detect differences in behaviour after the execution of suspension traces provided that the observations of *quiescence* all take longer than the stipulated duration  $M$ , but may not detect differences in refusal behaviour that require shorter observations of *quiescence*.

It is shown how this theory may be used to test real-time implementations under the assumption that the absence of system interaction with its environment for  $M$  time units implies *quiescence*. We have defined a nondeterministic ( $M$ -parameterized) test generation framework that generates test cases that are sound with respect to the corresponding implementation relation **tioco** $_M$ . The test generation is also exhaustive in the sense that for each non-conforming implementation a test case can be generated that can detect the non-conformance.

The framework can be effectively instantiated for subclasses of timed input-output transition systems for which  $out_M(\Delta(p) \text{ after } \sigma)$  is computable, as is the case for timed automata. Using standard symbolic state space representation in the form of difference bounded matrices [Dil90], a real-time version of TORX for timed automata models is being implemented.

The work presented here can be extended in a number of ways. As already indicated, it is possible to show a stronger exhaustiveness result for the test generation procedure based on an appropriate notion of equivalence of error traces. The generation procedure will hit each such class in the limit, provided that the error class is not negligible, i.e. it must have positive measure in some appropriate sense.

Another extension is to relax the requirement that there must be a uniform observation deadline  $M$  for *quiescence*. Obvious alternatives that we are studying are:

- the observation parameter  $M(\sigma)$  is a function of the behaviour (trace)  $\sigma$  observed so far. This would allow us to model sequential phases of *quiescence*, i.e. slow vs. quick response times;
- the observation parameter  $M(C_i)$  is a function of the communication channel  $C_i$  on which output is being observed. This would allow us to model different kinds of response times for different communication channels with the system under test, and would correspond to a real-time extension of the **mioco** implementation relation of [BHT98].

Our real-time theory inherits its focus on control aspects of system behaviour from the existing **ioco** theory. Ultimately, it will be important to combine this testing theory with methods for testing the static data aspects of systems. It will be interesting to see to what extent the symbolic representation of data types can be combined with symbolic representations of time.

In a more general vein, one can say that the development of a real-time testing theory forces us to confront modelling issues with respect to physical aspects of time and implementation. From a physical point of view, for example, it is questionable whether negligible behaviour can be implemented. This has also implications for specification formalisms that can be used to specify such behaviour, e.g. timed automata can define negligible behaviour by using guards that force behaviour to go through specific points in time, such as  $x = 3$ . It would seem that realistic specifications and/or implementation relations allow for tolerances in the evaluation

of clock conditions. This would then introduce a third source of non-determinism in the testing theory of real-time systems. At any rate, a more systematic study of the formal aspects of tolerance and robustness is definitely needed.

## References

- [BB04] L. Brandán Briones and E. Brinksma. A test generation framework for quiescent real-time systems. <http://fint.cs.utwente.nl/research/testing/files/BBB04.ps.gz>, 2004.
- [BFV<sup>+</sup>99] A. Belinfante, J. Feenstra, R.de Vries, J. Tretmans, N. Goga, L. Feijs, S. Mauw, and L. Heerink. Formal test automation: A simple experiment. In G.Csopaki, S.Dibuz, and K.Tarnay, editors, *Int. Workshop on Testing of Communicating Systems 12*, pages 179–196. Kluwer, 1999.
- [BHT98] E. Brinksma, L. Heerink, and J. Tretmans. Factorized test generation for multi input/output transition systems. In A.Petrenko and N. Yevtushenko, editors, *Int. Workshop on Testing of Communicating Systems 11*, pages 67–82. Kluwer, 1998.
- [CL97] D. Clarke and I. Lee. Automatic test generation for the analysis of a real-time system: Case study. In *IEEE Real Time Technology and Applications Symp.*, pages 112–124, 1997.
- [CLLS96] R. Cleaveland, I. Lee, P. Lewis, and S. Smolka. A theory of testing for soft real-time processes, 1996.
- [CO02] R. Cardell-Oliver. Conformance test experiments for distributed real-time systems. In *Proceedings of the int. symp. on Software testing and analysis*, pages 159–163. ACM Press, 2002.
- [Dil90] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proceedings of the int. workshop on Automatic verification methods for finite state systems*, pages 197–212. Springer-Verlag NY, Inc., 1990.
- [ENDK98] A. En-Nouaary, R. Dssouli, and F. Khendek. Timed test cases generation based on state characterization technique. In *19th IEEE Real-Time Systems Symp.*, pages 220–229, 1998.
- [FJJV96] J-C. Fernandez, C. Jard, T. Jeron, and C. Viho. Using on-the-fly verification techniques for the generation of test suites. In *Copmputer Aided Verification CAV'96. LNCS 1102, Springer-Verlan*. R.Alur and T.A.Hezinger, 1996.
- [FJJV97] J-C. Fernandez, C. Jard, T. Jeron, and C. Viho. An experiment in automatic generation of test suites for protocols with verification technology. In *Science of Computer Programming - Special Issue on COST247, Verification and Validation Methods for Formal Descriptions, 29(1-2)*, pages 123–146, 1997.
- [Gla93] R.J.van Glabbeek. The linear time-branching time spectrum ii (the semantics of sequential systems with silent moves). In *CONCUR'93. LNCS 715*, pages 66–81. E.Best, 1993.
- [HNTC99] T. Higashino, A. Nakata, K. Taniguchi, and R. Cavalli. Generating test cases for a timed i/o automaton model. In *IWTCS 1999*, pages 197–214, 1999.
- [ISO89] ISO8807. *Information processing systems, Open Systems Interconnection, LOTOS, A formal description technique based on the temporal ordering of observational behaviour*. Int. Organization for Standardization, 1989.
- [KT04] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *SPIN 2004*, pages 109–126. Springer-Verlag Heidelberg, 2004.
- [Lan90] R. Langerak. A testing theory for lotos using deadlock detection. In *Proceedings of the IFIP WG 6.1 Ninth int. Symp. on Protocol Spec., Testing, and Verification*, pages 87–98. IFIP, 1990.
- [LMN03] K. Larsen, M. Mikucionis, and B. Nielsen. Real-time system testing on-the-fly. In K.Sere, M.Walden, and A.Karlsson, editors, *The 15th Nordic Workshop on Programming Theory (NWPT)*, Åbo Akademi University, Turku, Finland, oct 2003. Extended abstract.
- [MMM95] D. Mandrioli, S. Morasca, and A. Morzenti. Generating test cases for real-time systems from logic specifications. *TOCS*, 13(4):365–398, Nov. 1995.
- [NH83] R.de Nicola and M. Hennessy. Testing equivalences for processes. In *ICALP83*, volume 154, 1983.
- [NS01] B. Nielsen and A. Skou. *Automated Test Generation from Timed Automata*. TACAS 2001: 343-357, 2001.
- [SVD01] J. Springintveld, F. Vaandrager, and P. D'Argenio. Testing timed automata. *Theoretical Computer Science*, 254(1-2):225–257, 2001.
- [TB03] J. Tretmans and E. Brinksma. Torx: Automated model-based testing. In *First European Conference on Model-Driven Software Engineering, Nuremberg*. A.Hartmann and K.Dussa-Ziegler, 2003.
- [Tre96] J. Tretmans. Test generation with inputs, outputs and repetitive quiescence. In *Software-Concepts and Tools*, 17(3), pages 103–120. Also: Technical Report N0. 96-26, Center for Telematics and Information Technology, University of Twente, The Netherlands, 1996.