A Test Generation Framework for *quiescent* Real-Time Systems

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 behaviour. 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. [1, 13, 9, 10]. 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 [12] 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,

J. Grabowski and B. Nielsen (Eds.): FATES 2004, LNCS 3395, pp. 64–78, 2005.

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 introduces the model of timed input-output transition systems and our conformance relation. Section 3 presents the real-time test generation algorithm. Section 4 illustrates the theory with an example in the setting of timed automata. Section 5 compares our achievements to related work. Finally, section 6 presents the conclusions and future work.

2 Implementation Relations for Real-Time *quiescence*

2.1 Timed Input-Output Transition Systems

In this section we introduce the concept of Timed Labelled Transition Systems, their properties and notation, and then specialize them to obtain the model of Timed Input-Output Transition Systems. After that, we proceed to obtain a conformance relation between a specification and an implementation, defined as timed input-output transition systems, analogous to the **ioco** relation for the untimed case.

For details of the underlying theory (the implementation relation **ioco**) we refer to [12]. To save space we omitted the proof of lemmas and theorems in this paper, but they can be found in the full version [15].

We distinguish three types of actions: *time-passage actions*, *visible labelled actions* and the special *internal action* τ. 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, viz. the nonnegative reals ($\mathbb{R}^+$); no lower *a priori* bounds are imposed on the delays between events.

Definition 1. *A timed labelled transition system (TLTS) is a 4-tuple* $\langle S, s_0, Act_{\tau\varepsilon}, \rightarrow \rangle$*, where*

- S *is a non-empty set of states*
- $s_0 \in S$ *is the initial state*
- $Act_{\tau\varepsilon} \stackrel{def}{=} Act \cup \{\tau\} \cup \mathcal{D}$ *are the actions* Act *including the internal action* τ *and time-passage actions; where* $\mathcal{D}$ *is* $\{\varepsilon(d) \mid d \in \mathbb{R}^+\}$
- $\rightarrow \subseteq (S \times Act_{\tau\varepsilon} \times S)$ *is the transition relation with the following consistency constraints:*
  - **Time Determinism** *whenever* $s \xrightarrow{\varepsilon(d)} s'$ *and* $s \xrightarrow{\varepsilon(d)} s''$ *then* $s' = s''$
  - **Time Additivity** $\forall\ s, s'' \in S \wedge \forall\ d_1, d_2 \geq 0 : (\exists\ s' \in S : s \xrightarrow{\varepsilon(d_1)} s' \xrightarrow{\varepsilon(d_2)} s'')$ *iff* $s \xrightarrow{\varepsilon(d_1+d_2)} s''$
  - **Null Delay** $\forall\ s, s' \in S : s \xrightarrow{\varepsilon(0)} s'$ *iff* $s = s'$.

The labels in Act_ε ($Act_\varepsilon \stackrel{def}{=} Act \cup \mathcal{D}$) represent the observable actions of a system, i.e. labelled actions and passage of time; the special label τ represents an unobservable internal action. A transition $(s, \mu, s') \in \rightarrow$ is denoted as $s \xrightarrow{\mu} s'$. A *computation* is a finite or infinite sequence of transitions:

$$s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} s_2 \xrightarrow{\mu_3} \cdots \xrightarrow{\mu_{n-1}} s_{n-1} \xrightarrow{\mu_n} s_n (\rightarrow \cdots)$$

A *timed trace* captures the observable aspects of a computation; it is the sequence of observable actions. The set of all *finite* sequences of actions over Act_ε is denoted by Act_ε^*, while ϵ denotes the empty sequence. If $\sigma_1, \sigma_2 \in Act_\varepsilon^*$ then $\sigma_1 \cdot \sigma_2$ is the concatenation of σ_1 and σ_2.

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

Definition 2. *Let* $p = \langle S, s_0, Act_{\tau\varepsilon}, \rightarrow \rangle$ *be a TLTS(Act) with* $s, s', s_i \in S; d, d', e \in \mathbb{R}^+; \mu_i \in Act_{\tau\varepsilon}; \beta \in Act; \alpha_i \in Act_\varepsilon; \alpha \in Act_\varepsilon^*$*, then*

$s \xrightarrow{\mu_1 \cdots \mu_n} s' \stackrel{def}{=} \exists\ s_0, \ldots, s_n : s = s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_n} s_n = s'$	
$s \xrightarrow{\mu_1 \cdots \mu_n} \stackrel{def}{=} \exists\ s' : s \xrightarrow{\mu_1 \cdots \mu_n} s'$	$s \not\xrightarrow{\mu_1 \cdots \mu_n} \stackrel{def}{=} \nexists\ s' : s \xrightarrow{\mu_1 \cdots \mu_n} s'$
$s \stackrel{\epsilon}{\Rightarrow} s' \stackrel{def}{=} s = s'$ *or* $s \xrightarrow{\tau \cdots \tau} s'$	$s \stackrel{\beta}{\Rightarrow} s' \stackrel{def}{=} \exists\ s_1, s_2 : s \stackrel{\epsilon}{\Rightarrow} s_1 \xrightarrow{\beta} s_2 \stackrel{\epsilon}{\Rightarrow} s'$
$s \stackrel{\varepsilon(d)}{\Rightarrow} s' \stackrel{def}{=} (\exists\ s_1, s_2 : s \stackrel{\epsilon}{\Rightarrow} s_1 \xrightarrow{\varepsilon(d)} s_2 \stackrel{\epsilon}{\Rightarrow} s')$ *or* $(\exists\ s_1, d', e : d' + e = d : s \stackrel{\varepsilon(d')}{\Rightarrow} s_1 \stackrel{\varepsilon(e)}{\Rightarrow} s')$	$s \xrightarrow{\alpha_1 \cdots \alpha_n} s' \stackrel{def}{=} \exists\ s_0 \ldots s_n : s = s_0 \stackrel{\alpha_1}{\Rightarrow} s_1 \stackrel{\alpha_2}{\Rightarrow} \cdots \stackrel{\alpha_n}{\Rightarrow} s_n = s'$
$s \stackrel{\alpha}{\Rightarrow} \stackrel{def}{=} \exists\ s' : s \stackrel{\alpha}{\Rightarrow} s'$	$s \stackrel{\alpha}{\not\Rightarrow} \stackrel{def}{=} \nexists\ s' : s \stackrel{\alpha}{\Rightarrow} s'.$

We do not always distinguish between a timed labelled transition system and its initial state: if $p = \langle S, s_0, Act_{\tau\varepsilon}, \rightarrow \rangle$ we will often identify the process p with its initial state s_0, e.g. we write $p \stackrel{\alpha}{\Rightarrow}$ instead of $s_0 \stackrel{\alpha}{\Rightarrow}$.

Definition 3.

- $ttraces(p) \stackrel{def}{=} \{\sigma \in Act_\varepsilon^* \mid p \stackrel{\sigma}{\Rightarrow}\}$
- $init(p) \stackrel{def}{=} \{\mu \in Act_{\tau\varepsilon} \mid p \xrightarrow{\mu}\}$

- $der(p) \stackrel{def}{=} \{p' \mid \exists\ \sigma \in Act_\varepsilon^* : p \stackrel{\sigma}{\Rightarrow} p'\}$
- $p\ \textbf{\textit{after}}\ \sigma \stackrel{def}{=} \{p' \mid p \stackrel{\sigma}{\Rightarrow} p'\}$
- $P\ \textbf{\textit{after}}\ \sigma \stackrel{def}{=} \bigcup_{p \in P} (p\ \textbf{after}\ \sigma)$, *where* P *is a set of states*
- p *is* deterministic *if* $\forall\ \sigma \in Act_\varepsilon^* : (p\ \textbf{\textit{after}}\ \sigma)$ *has at most one element. If* $\sigma \in ttraces(p)$, *then* $(p\ \textbf{\textit{after}}\ \sigma)$ *is overloaded to denote this element.*

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

Definition 4. *Let* $p = \langle S, s_0, Act_{\tau\varepsilon}, \rightarrow \rangle$ *be a TLTS*(Act), *then*
p *is* **time divergent***: if for all* $s \in S$ *there exists an infinite computation* σ *from* s *with infinite cumulative delay:*

$$\forall\ s \in S : \exists\ \sigma \in Act_{\tau\varepsilon}^{\omega} : \sigma = \mu_1 \cdot \mu_2 \cdot \mu_3 \cdots : s \stackrel{\sigma}{\rightarrow}\ \wedge\ \Sigma\{d_i \mid \mu_i = \varepsilon(d_i)\} = \infty$$

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

$$\exists\ s \in S : \exists\ \sigma \in Act_{\tau\varepsilon}^{\omega} : \sigma = \mu_1 \cdot \mu_2 \cdot \mu_3 \cdots : s \stackrel{\sigma}{\rightarrow}\ \wedge\ \mid \{i \mid \mu_i \neq \varepsilon(d_i)\} \mid = \infty \\ \wedge\ \Sigma\{d_i \mid \mu_i = \varepsilon(d_i)\} < \infty.$$

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

We now introduce timed input-output transition systems (*TIOTS*) to model timed systems for which the set of actions can be partitioned into *output actions* and *input actions*. To do this properly, we formalize the notion of *input enabling*: 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. Also, we want to exclude the possibility that the flow of time in a system can be blocked because the environment does not provide certain input actions, i.e. there must be no *forced inputs*.

Definition 5. *A timed input-output transition system (TIOTS) is a timed labelled transition system* $\langle S, s_0, Act_{\tau\varepsilon}, \rightarrow \rangle$ *with Act partitioned into* input actions, Act_I, *and* output actions, Act_U, $(Act_I \cup Act_U = Act, Act_I \cap Act_U = \emptyset)$, *that has the properties of*
weak input enabling*:* $\forall\ s \in S : \forall\ \mu \in Act_I : s \stackrel{\mu}{\Rightarrow}$
no forced inputs*: iff for all* $s \in S$ *there exists an infinite computation* σ *from* s *containing no input actions and with infinite cumulative delay:* $\forall\ s \in S : \exists\ \sigma \in (Act_U \cup \{\tau\} \cup \mathcal{D})^{\omega} : \sigma = \mu_1 \cdot \mu_2 \cdots : s \stackrel{\sigma}{\rightarrow}\ \wedge\ \Sigma\{d_i \mid \mu_i = \varepsilon(d_i)\} = \infty$

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

We follow the convention that input actions are identified by names followed by a *?*-symbol, and output actions by names followed by a *!*-symbol.

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

Definition 6. *Let* $\sigma \in Act_\varepsilon^*$*, then*

- σ *is a* normalized timed trace *iff* $\sigma \in (\mathcal{D}\cdot Act)^*$
- $nttraces(p) = \{\sigma \in (\mathcal{D}\cdot Act)^* \mid p \stackrel{\sigma}{\Rightarrow}\}$
- *for normalized timed traces* $\sigma = \varepsilon(d_0)\cdot a_0\cdot\varepsilon(d_1)\cdot a_1 \cdots \varepsilon(d_n)\cdot a_n$ *we also write* $\widehat{\sigma} = a_0(d_0)\cdot a_1(d_1)\cdots a_n(d_n)$.

If a timed trace begins with an action it can always be converted to a *normalized timed trace* by combining delays, or adding zero delays $\varepsilon(0)$ in the appropriate places. But if a timed trace ends with a delay, such as $\sigma = \varepsilon(d_0)a?\varepsilon(d_1)b!\varepsilon(d_2)$ then is not possible to interpret it as a *normalized timed trace*. The next lemma shows, however, that in the presence of input enabledness *normalized timed traces* preserve the information of timed traces.

Lemma 7. *Let* $p_1,\ p_2 \in TIOTS(Act_I, Act_U)$*, then*
$ttraces(p_1) \subseteq ttraces(p_2)$ *iff* $nttraces(p_1) \subseteq nttraces(p_2)$.

From now on we will not distinguish between a timed trace σ and its normalization $\widehat{\sigma}$ if it exist.

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 8. *Let* $p \in TIOTS(Act_I, Act_U)$*. A state* s *of* p *is* quiescent*, denoted by* $\delta(s)$*, iff* $\forall\ \mu \in Act_U : \forall\ d \in \mathbb{R}^+ : s \stackrel{\mu(d)}{\not\Rightarrow}$.

As before in the untimed case, we can start out by representing *quiescence* as a special action δ ($\delta \notin Act \cup \{\tau\}$)[1], and extending the timed transition relation of a *TIOTS* p to include self-loop transitions $s \stackrel{\delta}{\rightarrow} s$ iff s is a *quiescent* state. Moreover, let $\Delta(p)$ denote the extended timed transition system of p that is obtained in this way.

2.2 Timed Implementation Relations

The extension of the timed transition relation allows us to define the following relation over *TIOTS*.

Definition 9. *Let* p *and* $q \in TIOTS(Act_I, Act_U)$*, then*
$q \sqsubseteq_{tiorf} p$ *iff* $nttraces(\Delta(q)) \subseteq nttraces(\Delta(p))$.

For specifications $p \in TIOTS$ the *quiescent* states can, in principle, be identified by analyzing the timed transition system, i.e. we can assume that $\Delta(p)$ is

[1] In [12] the action symbol θ is used for the observation of *quiescence*. We prefer to use δ for both *quiescence* and its observation, in line with the philosophy that identical actions synchronize.

at our disposal. For implementations q, however, we only can detect *quiescence* by waiting for outputs. But we cannot wait forever, and therefore need to choose a maximal duration M. This motivates the following parameterized version of $\sqsubseteq_{tiorf}$, where σ can only appear after M time-units.

Definition 10. *Let p and $q \in$ TIOTS, then $q \sqsubseteq^{M}_{tiorf} p$ iff $\Delta_M(q) \subseteq \Delta_M(p)$ where $\Delta_M(r) \stackrel{def}{=} nttraces(\Delta(r)) \cap (\mathcal{D}\cdot Act \cup \varepsilon(M)\cdot\delta)^*$.*

The above definition takes only into account observations of *quiescence* that are made after a minimal delay of M time units. Naturally this definition implies a pre-order.

Lemma 11. *If $M_1 < M_2$, then if $q \sqsubseteq^{M_1}_{tiorf} p$ then $q \sqsubseteq^{M_2}_{tiorf} p$.*

This is not without consequences: in contrast to the untimed case, time delays can change the system state, which has interesting consequences, as shown in the quirky coffee machine of Figure 1, inspired by [21].

Example 12. Figure 1 shows two quirky coffee machines with time. Suppose both graphs are saturated with input action transitions in each state by adding self-loops for all input transitions that are not explicitly given. For simplicity, in the figure, we use *m?* for money, *b?* for bang, *c?*, *c!* for coffee, and *t?*, *t!* for tea. We suppose that each action resets the clock x and that $k < M$ (we used the representation of timed automata). 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.

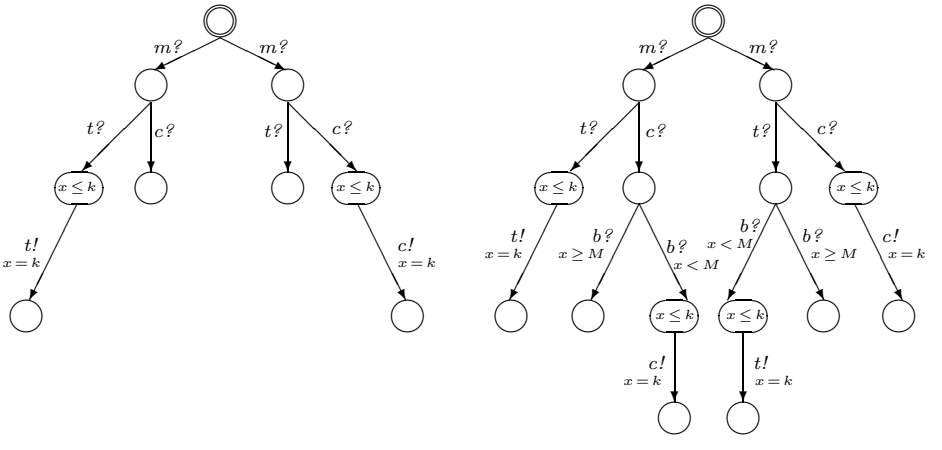


Fig. 1. The quirky coffee machine with time, a modified version of [21]

The output set of a given state of a system in $TIOTS(Act_I, Act_U)$ consists of the time stamped output actions that are allowed from that state (abstracting from τ-actions), including δ-actions after a delay of M time-units.

Definition 13. *Let p be a state of an (extended) timed transition system in $TIOTS(Act_I, Act_U)$,*
then $out_M(p) = \{\mu(d) \mid \mu \in Act_U \wedge p \stackrel{\mu(d)}{\Rightarrow}\} \cup \{\delta(M) \mid p \stackrel{\delta(M)}{\Rightarrow}\}$
and for P a set of states, then $out_M(P) = \bigcup_{p \in P} out_M(p)$.

Lemma 14. *Let p and $q \in TIOTS(Act_I, Act_U)$, then*

$$q \sqsubseteq^M_{tiorf} p \text{ iff}$$
$$\forall\, \sigma \in (\mathcal{D}{\cdot}Act \cup \varepsilon(M){\cdot}\delta)^* : out_M(\Delta(q) \textbf{ after } \sigma) \subseteq out_M(\Delta(p) \textbf{ after } \sigma).$$

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

Definition 15. *Let p and $q \in TIOTS(Act_I, Act_U)$, then*

$$q \sqsubseteq^M_{tioco} p \text{ iff } \forall\, \sigma \in \Delta_M(p) : out_M(\Delta(q) \textbf{ after } \sigma) \subseteq out_M(\Delta(p) \textbf{ after } \sigma)$$

we also write $\sqsubseteq^M_{tioco}$ as $\mathbf{tioco}_M$.

2.3 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 TIOTS and $M \in \mathbb{R}^+$, then*

- *a state s of q is* M-quiescent *iff* $\forall\, s' \in (s \textbf{ after } \varepsilon(M)) : s'$ *is* quiescent
- *q is* M-quiescent *iff all states s of q are* M-quiescent.

In line with the above development we now want to formalize how *normalized timed traces* of *TIOTS*s may be enriched directly with δ-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 $M = 4$ and $\sigma = a?(2) \cdot b?(5) \cdot c!(3)$ is an observed timed trace then it is also possible to observe $\sigma' = a?(2){\cdot}\delta(4){\cdot}b?(1){\cdot}c!(3)$. We formalize the addition of δ-observations to *normalized timed traces* as a formal relation δ_M between (extended) *normalized timed traces*.

Definition 17. *Let σ, σ' be normal form of $\sigma, \sigma' \in (\mathcal{D}{\cdot}(Act \cup \delta))^*$, then*

- $\sigma\ \delta_M\ \sigma'$ *iff* $\exists\, \sigma_1, \sigma_2 : \exists\, \mu : \exists\, d \geq M : \sigma = \sigma_1{\cdot}\mu(d){\cdot}\sigma_2 \wedge \sigma' = \sigma_1{\cdot}\delta(M){\cdot}\mu(d{-}M){\cdot}\sigma_2$

- *let Σ be a set of normalized timed traces, then* $\delta_M(\Sigma) = pref\,(\bigcup_{\sigma\in\Sigma}\{\sigma' \mid \sigma\ \delta_M^*\ \sigma'\})$ *where pref(S) is interpreted as the prefix-closure of a set of traces S and δ_M^* is the reflexive transitive closure of the relation δ_M.*

If δ-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 property expressed in the following lemma.

Lemma 18. *Let $q \in TIOTS(Act_I, Act_U)$ be* M-quiescent*, then*
$\nexists\ \sigma \in \delta_M(nttraces(q)) : \exists\ \mu \in Act_U : \sigma = \sigma' \cdot \delta(M) \cdot \mu(d)$.

Corollary 19. *Let $q \in TIOTS$ be* M-quiescent*, then $\delta_M(nttraces(q)) = \Delta_M(q)$.*

This corollary means that if an implementation q can be assumed to be M-*quiescent* we may use the set of enriched observations $\delta_M(nttraces(q))$ 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.

3 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 20.
- *A* test case *t is a TLTS $\langle S, s_0, Act_\varepsilon \cup \{\delta\}, \rightarrow\rangle$ such that*
  - *t is deterministic and has* bounded behaviour*, i.e. $\exists\ N > 0 : \forall\ \sigma :$ $\sigma = \mu_1.\mu_2.\mu_3 \ldots : \ |\{i \mid \mu_i \neq \varepsilon(d_i)\}| < \infty$ and $\Sigma\{d_i \mid \mu_i = \varepsilon(d_i)\} < N$*
  - *S contains the terminal states* **pass** *and* **fail***, with* init(**pass**) = init(**fail**) $= \emptyset$
  - *for any state $t' \in S$ of the test case with $t' \neq$* **pass**, **fail**, $\exists\ d > 0$ *with* init(t' **after** $\varepsilon(d')$) $= Act_U \cup \{\varepsilon(e) \mid e = d - d'\}$ *for all $d' < d$,* init(t' **after** $\varepsilon(d)$) $= \mu$ *with $\mu \in Act_I$ or $\mu = \delta$*
  - *t does not have τ-transitions*

  The class of test cases over Act_I and Act_U is denoted as $\mathcal{TTEST}(Act_I, Act_U)$ but we represent it similarly as a timed automata, only for simplifying the notation
- *A* test suite **T** *is a set of test cases:* **T** $\subseteq \mathcal{TTEST}(Act_I, Act_U)$.

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 21. *Let $t \in \mathcal{TTEST}(Act_I, Act_U)$ and $imp \in TIOTS(Act_I, Act_U)$* M-quiescent*, then*

- *Running a test case t with an implementation imp is modelled by the parallel operator $||: \mathcal{TTEST}(Act_I, Act_U) \times TIOTS(Act_I, Act_U) \rightarrow TIOTS(Act_I, Act_U)$ which is defined by the following inference rules:*

$$\begin{array}{ll} imp \xrightarrow{\tau} imp' & \vdash \; t||imp \xrightarrow{\tau} t||imp' \\ t \xrightarrow{\delta} t' & \vdash \; t||imp \xrightarrow{\delta} t'||imp \\ t \xrightarrow{\mu} t', imp \xrightarrow{\mu} imp', \mu \in Act & \vdash \; t||imp \xrightarrow{\mu} t'||imp' \\ t \xrightarrow{\varepsilon(d)} t', imp \xrightarrow{\varepsilon(d)} imp' & \vdash \; t||imp \xrightarrow{\varepsilon(d)} t'||imp' \end{array}$$

- *A test run of t with imp, is a $\sigma \in \Delta_M$ of $t||imp$ leading to a terminal state of $t : \sigma$ is a test run of t and*

$$imp \stackrel{def}{=} \exists\, imp' : (t||imp \stackrel{\sigma}{\Rightarrow} \mathbf{pass}||imp') \text{ or } (t||imp \stackrel{\sigma}{\Rightarrow} \mathbf{fail}||imp')$$

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

$$imp \textbf{ passes } t \stackrel{def}{=} \forall\, \sigma \in \Delta_M : \forall\, imp' : t||imp \stackrel{\sigma}{\not\Rightarrow} \mathbf{fail}||imp'$$

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

$$imp \textbf{ passes } \mathbf{T} \stackrel{def}{=} \forall\, t \in \mathbf{T} : imp \textbf{ passes } t$$

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

$$imp \textbf{ fails } \mathbf{T} \stackrel{def}{=} \exists\, t \in \mathbf{T} : imp \textbf{ passes } t.$$

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 an only if all possible test runs lead to the verdict **pass**.

3.1 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 [8]:

$$B \stackrel{def}{=} a; B \mid B + B \mid \Sigma\, \mathcal{B}$$

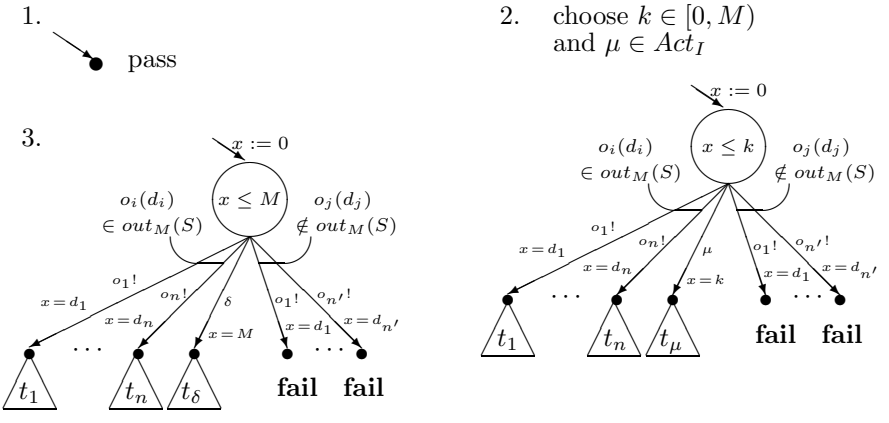
where $a \in Act_\varepsilon$, $\mathcal{B}$ is a countable set of behaviour expressions, and the axioms and the inference rules are:

$$\begin{array}{ll} a \in Act & \vdash \; a; B \xrightarrow{a} B' \\ a = \varepsilon(d), d' < d & \vdash \; a; B \xrightarrow{\varepsilon(d')} \varepsilon(d - d'); B' \\ a = \varepsilon(d) & \vdash \; a; B \xrightarrow{\varepsilon(d)} B' \\ B_1 \xrightarrow{\mu} B_1',\ \mu \in Act_\varepsilon & \vdash \; B_1 + B_2 \xrightarrow{\mu} B_1' \\ B_2 \xrightarrow{\mu} B_2',\ \mu \in Act_\varepsilon & \vdash \; B_1 + B_2 \xrightarrow{\mu} B_2' \\ B \xrightarrow{\mu} B', B \in \mathcal{B}, \mu \in Act_\varepsilon & \vdash \; \Sigma\, \mathcal{B} \xrightarrow{\mu} B' \end{array}$$

Moreover, we use $\mu(d)$ as syntactic sugar for $\varepsilon(d); \mu$.

Test case generation procedure. We define a procedure to generate test cases from a given specification timed transition system. Similar to [12] test cases result from the nondeterministic, 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 $o(d) \in out_M(S)$, 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 [16, 14].



1. *termination*
   $t :=$ **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*
   $$\begin{aligned} t := \quad & \Sigma\{o_i(d_i); t_i \mid o_i \in Act_U \wedge o_i(d_i) \in out_M(S)\} \\ & + \mu(k); t_\mu \\ & + \Sigma\{o_j(d_j); \mathbf{fail} \mid o_j \in Act_U \wedge o_j(d_j) \notin out_M(S)\} \end{aligned}$$
   where x is a clock, k is a timed variable and t_i and t_μ are obtained by recursively applying the algorithm for (S **after** $o_i(d_i)$) and (S **after** $\mu(k)$), respectively.
   Test case t is waiting for k time-units an treating to make and input (μ). If an output arrives from the implementation it checks; if it is an invalid response, i.e. $o_j(d_j) \notin out_M(S)$ 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 (μ) and continues recursively.

3. *waiting for outputs*
   $$\begin{aligned} t := \quad & \Sigma\{o_i(d_i); t_i \mid o_i \in Act_U \wedge o_i(d_i) \in out_M(S)\} \\ & + \Sigma\{\delta(M); t_\delta \mid \delta \in out_M(S \textbf{ after } \varepsilon(M))\} \\ & + \Sigma\{\delta(M); \mathbf{fail} \mid \delta \notin out_M(S \textbf{ after } \varepsilon(M))\} \\ & + \Sigma\{o_j(d_j); \mathbf{fail} \mid o_j \in Act_U \wedge o_j(d_j) \notin out_M(S)\} \end{aligned}$$
   where x is a clock and t_i and t_δ are obtained by recursively applying the algorithm for (S **after** $o_i(d_i)$) and (S **after** $\varepsilon(M)$), respectively.

Test case t is waiting for M time-units if an output arrive from the implementation it checks; if it is an invalid response, i.e. $o_j(d_j) \notin out_M(S)$ 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* δ 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 $\mathbf{tioco}_M$ relation. This property is shown in the following theorem.

Theorem 22. *Let spec* $\in$ *TIOTS, then for all* M-quiescent *imp* $\in$ *TIOTS and all test cases* t *obtained from spec by the above procedure:*

$$imp\ \mathbf{tioco}_M\ spec \Rightarrow imp\ \mathbf{passes}\ t.$$

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.

Definition 23. *Let* $p \in TIOTS$, *then*
$\sigma \in \Delta_M(p)$ *is* $\delta(M)$*-saturated iff for all* σ' *with* $\sigma\ \delta_M\ \sigma'$ *we have* $\sigma = \sigma'$.

Theorem 24. *Let spec* $\in$ *TIOTS, then for all* M-quiescent *imp* $\in$ *TIOTS with imp* $\mathbf{tioco}_M$ *spec, there exists a test case* t *generated from spec by the procedure such that: imp* $\mathbf{passes}$ t.

The exhaustiveness of our test generation procedure as proven in [15] 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.

4 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 [16], 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 25. Figure 2 shows two quirky coffee machines with time. The first one is a specification and the second one is a wrong implementation. To the

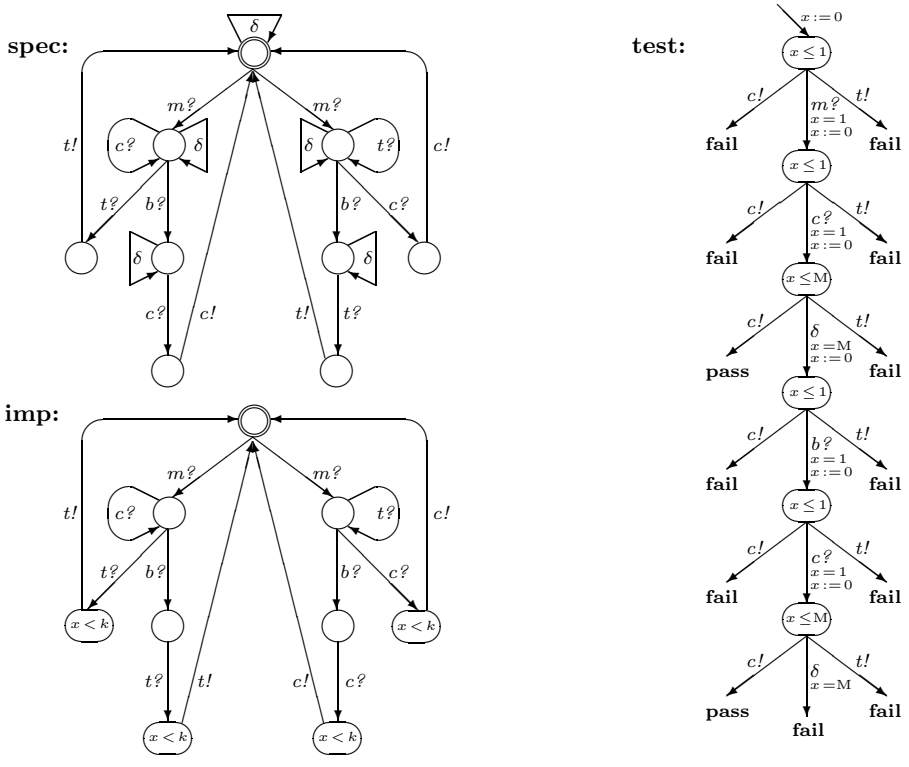


Fig. 2. A specification of a quirky coffee machine with time, an implementation with $M = k$, and a test case derived from the specification

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 δ-transitions, while in the implementation we detect them using $M = k$. We assume that $k > 1$.
The problem appears because:

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

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

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

5 Related Work

As already indicated before this work is closely related to work carried out by Krichen et al. in [16], and closely related work by Larsen et al. [14], who deal with a *quiescence*-free interpretation of timed **ioco** 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 [2], 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 [11] 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" [11]. Cardell-Oliver presents a method for networks of deterministic timed automata extended with integer data variables [17], 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 [7, 22].

Clarke and Lee [3] 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 [18] that bears a close resemblance to the classical testing theory of Hennessy and De Nicola [19]. Mandrioli et al. use temporal logic with arithmetic on a discrete time base [5].

6 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 $\mathbf{tioco}_M$. 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 $out_M(\Delta(spec)\ \mathbf{after}\ \sigma)$ is computable, as is the case for timed automata. Using standard symbolic state space representation in the form of difference bounded matrices [4], 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 in 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) σ observed so far. This would allows 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 allows 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 [6].

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

1. A.Belinfante, J.Feenstra, R.deVries, J.Tretmans, N.Goga, L.Feijs, S.Mauw, and L.Heerink. Formal test automation: A simple experiment. In *Int. Workshop on Testing of Communicating Systems 12* (1999), Kluwer, pp. 179–196.
2. B.Nielsen, and A.Skou. *Automated Test Generation from Timed Automata.* TACAS 2001: 343-357, 2001.
3. 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.* (1997), pp. 112–124.
4. 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* (1990), Springer-Verlag NY, Inc., pp. 197–212.

5. D.Mandrioli, S.Morasca, and A.Morzenti. Generating test cases for real-time systems from logic specifications. *TOCS 13*, 4 (1995), 365–398.
6. E.Brinksma, L.Heerink, and J.Tretmans. Factorized test generation for multi input/output transition systems. In *Int. Workshop on Testing of Communicating Systems 11* (1998), Kluwer, pp. 67–82.
7. En-NouaaryA, R.Dssouli, and F.Khendek. Timed test cases generation based on state characterization technique. In *19th IEEE Real-Time Systems Symp.* (1998), pp. 220–229.
8. 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.
9. J-C.Fernandez, C.Jard, T.Jeron, and C.Viho. Using on-the-fly verification techniques for the generation of test suites. In *Copmuter Aided Verification CAV'96. LNCS 1102, Springer-Verlan* (1996), R.Alur and T.A.Hezinger.
10. J-C.Fernandez, C.Jard, T.Jeron, and C.Viho. An experiment in automatic generation of test suites for protocols with verification technology. In *Sience of Computer Programming - Special Issue on COST247, Verification and Validation Methods for Formal Descriptions, 29(1-2)* (1997), pp. 123–146.
11. J.Springintveld, F.Vaandrager, and P.D'Argenio. Testing timed automata. *Theoretical Computer Science 254*, 1–2 (2001), 225–257.
12. J.Tretmans. Test generation with inputs, outputs and repetitive quiescence. In *Software-Concepts and Tools, 17(3)* (1996), Also: Technical Report N0. 96-26, Center for Telematics and Information Technology, University of Twente, The Netherlands, pp. 103–120.
13. J.Tretmans, and E.Brinksma. Torx: Automated model-based testing. In *First European Conference on Model-Driven Software Engineering, Nuremberg* (2003), A.Hartmann and K.Dussa-Ziegler.
14. K.Larsen, M.Mikucionis, and B.Nielsen. Real-time system testing on-the-fly. In *The 15th Nordic Workshop on Programming Theory (NWPT)* (2003), K.Sere, M.Walden, and A.Karlsson, Eds. Extended abstract.
15. L.Brandán-Briones, and E.Brinksma. A test generation framework for quiescent real-time systems. http://fmt.cs.utwente.nl/research/testing/files/BBB04.ps.gz.
16. M.Krichen, and S.Tripakis. Black-box conformance testing for real-time systems. In *SPIN 2004* (2004), Springer-Verlag Heidelberg, pp. 109–126.
17. R.Cardell-Oliver. Conformance test experiments for distributed real-time systems. In *Proceedings of the int. symp. on Software testing and analysis* (2002), ACM Press, pp. 159–163.
18. R.Cleaveland, I.Lee, P.Lewis, and S.Smolka. A theory of testing for soft real-time processes, 1996.
19. R.deNicola, and M.Hennessy. Testing equivalences for processes. In *ICALP83* (1983), vol. 154.
20. R.J.vanGlabbeek. The linear time-branching time spectrum ii (the semantics of sequential systems with silent moves). In *CONCUR'93. LNCS 715* (1993), E.Best, pp. 66–81.
21. 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* (1990), IFIP, pp. 87–98.
22. T.Higashino, A.Nakata, K.Taniguchi, and R.Cavalli. Generating test cases for a timed i/o automaton model. In *IWTCS 1999* (1999), pp. 197–214.