

Generating Analog-Clock Real-Time Testers Using Action Refinement Techniques

Saddek Bensalem*, Moez Krichen**, Stavros Tripakis***

*Verimag Laboratory, Centre Equation 2, avenue de Vignate, 38610, Gières, France.
saddek.bensalem@imag.fr

** Verimag and LAAS Laboratories, 7 avenue du Colonel Roche, 31077, Toulouse, France.
moez.krichen@laas.fr

*** Verimag Laboratory and Cadence Research Laboratories,
2150 Shattuck Avenue, 10th floor, Berkeley, CA 94704, USA.
tripakis@cadence.com.

Abstract

In a previous work we proposed a method for generating digital-clock tests for real-time systems using action refinement techniques. In this work, we extend this method for generating analog-clock testers. Analog-clock testers are testers which can observe real-time with precision. Our goal from testing is to check the conformance of a given implementation with respect to a given specification (the model). The main benefit of the method is to save memory space needed to build and to store tests. One important contribution of this work is a simplified way for both modelling and testing real-time systems. We first write a (high-level) simplified version of the model of the system, as an input-output transition system (IOTS) and then we refine it into a more detailed (low-level) model as a timed input-output transition system (TIOTS). This same mechanism applies to the test generation procedure.

1. Introduction

Action refinement techniques are well experimented techniques in the field of hierarchical design of wide classes of systems. They mainly consist in translating high-level actions into lower-level ones. That is to move from a high-level abstraction to a lower one until reaching the implementation level.

Applying action refinement techniques in the field of testing is quite promising. Current techniques typically suffer from state explosion problems: this includes test generation, storage and execution. The problem is more dramatic in the case of timed systems, where an extra level of complexity is introduced by handling time-measuring variables (clocks).

Our main goal in this paper is to reduce the size of generated tests. Our focus is on analog-clock tester generation for real-time systems [3] and our objective is to improve our previous method [4] to generate such tests. To achieve this, we use an approach based on action refinement. In our timed setting, untimed actions are refined into timed actions. This helps reduce redundancy and results in optimized storage of useful data.

1.1. Overview of our approach

As already mentioned, we are interested in testing real-time systems. For this goal, we adopt the following approach. We assume that the model of the system we want to test is given as an untimed graph. The latter describes the behavior of the considered system using high-level actions. More precisely this untimed graph is, in our case, an *input-output transition system* (IOTS). For instance, an example of an IOTS is the following

$$\rightarrow q \xrightarrow{\text{double?}} q' \xrightarrow{\text{bright!}} q''.$$

That is the (*high-level*) model of a lighting device: “ q ” is the *initial state*; “double” is a high-level *input-action*; and “bright” a high-level *output-action*. It says that after receiving input “double” the lighting device will emit output “bright”.

We then assume that each high-level action is refined into a *timed path*. For instance, the actions “double” and “bright” are refined, respectively, into

$$q \xrightarrow[0, \infty]{\text{touch}_r?} p \xrightarrow[0, 1]{\text{touch}_r?} q' \text{ and } q' \xrightarrow[2, 3]{\text{bright}_r!} q'',$$

where touch_r is a *low-level input-action* and bright_r a *low-level output-action*. That means that input double happens if two consecutive touch_r ’s occur within one time-unit interval. The interval $[0, \infty]$ means that there is no constraint on the timing of occurrence of the first touch_r . Similarly, the refinement of bright means that a time elapse between 2 and 3 time-units is needed before moving to the desired state.

Thus, the IOTS above is transformed into the following *timed input-output transition system* (TIOTS)

$$\rightarrow q \xrightarrow[0, \infty]{\text{touch}_r?} p \xrightarrow[0, 1]{\text{touch}_r?} q' \xrightarrow[2, 3]{\text{bright}_r!} q''.$$

We define *analog-clock semantics* of a given TIOTS. Analog-clock means that we observe time with an infinite-precision. Given an IOTS, the next step consists in using the algorithm of [7] to generate (untimed) tests. The generated tests are given as trees. Other refinement techniques are then used to transform these test trees into analog-clock testers.

The remaining part of the paper is structured as follows. Section 2 recalls the untimed testing framework of [7]. Section 3 defines the TIOTS model. Section 4 describes the timed testing framework. The action-refinement rules to transform an IOTS into a TIOTS are given in Section 5. The method for deriving refined analog-clock testers from untimed tests is described in Section 6. Finally, Section 7 concludes the paper and gives directions for future-work.

1.2. Related Work

A lot of literature dealing with action refinement techniques exists for the case of systems without time-constraints [9, 2]. Recently, several works [5, 6] have been devoted to the case of timed systems.

To our knowledge, only few works have attempted to investigate the link between testing and action refinement techniques. In [8], the authors present an approach to automatically obtain (untimed) test cases at some required level of detail by a particular type of action refinement, so-called *atomic linear input-inputs refinement*. In our previous work [1], we proposed a method for generating *digital-clock testers* using action refinement techniques. These are testers which can measure time only with a finite precision. The refinement techniques we used consist in refining low-level untimed actions into sequences of high-level (digital-clock) timed-sequences for both input and output actions.

2. Model-Based Untimed Conformance Testing

Definition 1 An input-output transition system is a 5-tuple $(Q, \text{In}, \text{Out}, \text{Tr}, q_0)$ where: Q is a nonempty countable set of states. In is a countable set of input labels. Out is a countable set of output labels (such that $\text{In} \cap \text{Out} = \emptyset$). $\text{Tr} \subseteq Q \times (\text{In} \cup \text{Out}) \times Q$ the transition relation. q_0 the initial state.

A triple $(q, \mu, q') \in \text{Tr}$ is denoted $q \xrightarrow{\mu} q'$. We write $q \xrightarrow{\mu}$ if $\exists q' \in Q : q \xrightarrow{\mu} q'$. An IOTS is said to be *input-complete* if $\forall q \in Q, \forall \lambda \in \text{In} : q \xrightarrow{\lambda}$. A state is said to be *quiescent* if no output-action is possible from it. A new output label δ is considered. For each quiescent state q , a self-loop $q \xrightarrow{\delta} q$ is added to the considered IOTS. Next, we consider only IOTS for which this operation is already achieved. Let

$$\text{Out}_\delta = \text{Out} \cup \{\delta\} \text{ and } L_\delta = \text{In} \cup \text{Out}_\delta.$$

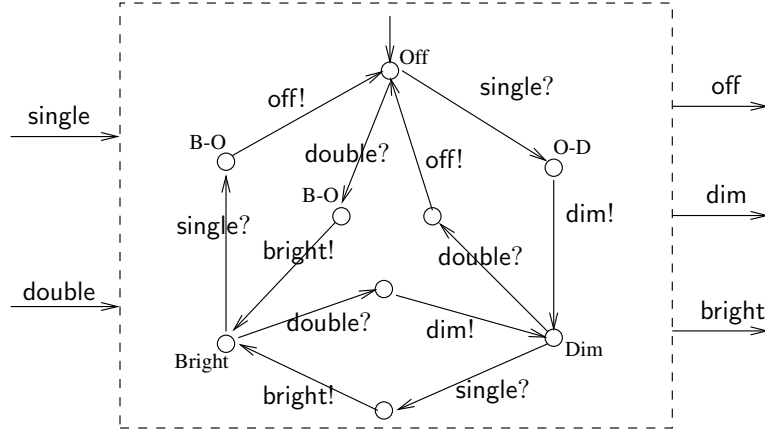


FIG. 1 – An example of an IOTS: a lighting device.

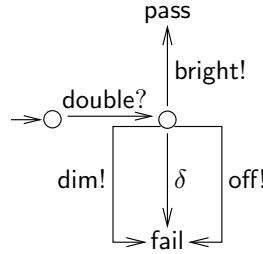


FIG. 2 – An untimed test represented as an IOTS.

For $\sigma = \mu_1 \cdots \mu_n \in L_\delta^+$, we write $q \xrightarrow{\sigma} q'$ if $\exists q_1, q_2, \dots, q_{n-1} \in Q : q \xrightarrow{\mu_1} q_1 \xrightarrow{\mu_2} q_2 \cdots q_{n-1} \xrightarrow{\mu_n} q'$ and write $q \xrightarrow{\sigma}$ if $\exists q' \in Q : q \xrightarrow{\sigma} q'$.

The sequence of transitions $q \xrightarrow{\mu_1} q_1 \xrightarrow{\mu_2} q_2 \cdots q_{n-1} \xrightarrow{\mu_n} q'$ is called a *path* of the IOTS. An IOTS is said to be *deterministic* if $\forall q, q', q'' \in Q, \forall \mu \in L_\delta :$

$$q \xrightarrow{\mu} q' \wedge q \xrightarrow{\mu} q'' \Rightarrow q' = q''.$$

An example of an IOTS is given in Figure 1. It is a modification of the case study presented in [3].

For $q \in Q, P \subseteq Q$ and $\sigma \in L_\delta^+$:

- $q \text{ after } \sigma =_{df} \{q' \mid q \xrightarrow{\sigma} q'\}.$
- $\text{out}(q) =_{df} \{\mu \in \text{Out}_\delta \mid q \xrightarrow{\mu}\}.$
- $\text{STraces}(q) =_{df} \{\sigma \in L_\delta^+ \mid q \xrightarrow{\sigma}\}.$
- $\text{out}(P) =_{df} \bigcup_{q \in P} \text{out}(q).$

Let $\text{Spec} = (Q, \text{In}, \text{Out}, \text{Tr}, q_0)$ (specification) and $\text{Imp} = (Q', \text{In}, \text{Out}, \text{Tr}', q'_0)$ (implementation) be two IOTS: $\text{Imp} \text{ ioco } \text{Spec} =_{df} \forall \sigma \in \text{STraces}(q_0) : \text{out}(q'_0 \text{ after } \sigma) \subseteq \text{out}(q_0 \text{ after } \sigma).$

Untimed tests can be represented as either total functions or IOTS. A possible test for the lighting-device example is given in Figure 2. The *execution of the test* T on the implementation Imp can be defined as the

parallel composition of the IOTS defined by T and Imp , with the usual *synchronization* rules for transitions carrying the same label.

We say that Imp passes the test, denoted Imp passes T , if state fail is not reachable in the product $Imp\|T$. We say that an implementation passes (resp. fails) a set of tests \mathcal{T} if it passes all tests (resp. fails at least one test) in \mathcal{T} . We say that \mathcal{T} is *sound* with respect to $Spec$ if

$$\forall Imp : Imp \text{ ioco } Spec \Rightarrow Imp \text{ passes } \mathcal{T}.$$

Untimed test generation is performed by Algorithm 1 [7]. Given a X , $\text{pick}(X)$ chooses randomly an element in X . Given a set of states P and an action μ , $\text{succ}(P, \mu)$ is defined as the set of states which can be reached by some state in P after executing action μ .

Algorithm 1 Untimed test generation.

```

1   $S \leftarrow \{q_0\}; T \leftarrow$  the one-node tree with root  $S$ ;
2  while(true)
3    foreach(leaf  $S$  of  $T$  distinct from pass and fail)
4       $i \leftarrow \text{pick}(\{0, 1, 2\})$ ;
5      case( $i = 0$ ):
6         $\mu \leftarrow \text{pick}(\text{In}); S' \leftarrow \text{succ}(S, \mu)$ ;
7        append edge  $S \xrightarrow{\mu} S'$  to  $T$ ;
8      case( $i = 1$ ):
9        foreach( $\nu \in \text{Out}_\delta$ )
10        $S' \leftarrow \text{succ}(S, \nu)$ ;
11       if( $S' \neq \emptyset$ ) append edge  $S \xrightarrow{\nu} S'$  to  $T$ ;
12       else append edge  $S \xrightarrow{\nu} \text{fail}$  to  $T$ ;
13     case( $i = 2$ ): replace  $S$  with pass in  $T$ ;
```

3. Timed Input-Output Transition Systems

Let \mathbb{R} be the set of non-negative reals and \mathbb{N} the set of non-negative integers. An *analog-timed sequence* over the set of actions $L = \text{In} \cup \text{Out}$ is a sequence $\rho = (\mu_0, t_0)(\mu_1, t_1) \cdots (\mu_n, t_n)$ where $\mu_i \in L$ and $t_i \in \mathbb{R}$ and such that $\forall i : t_i \leq t_{i+1}$. The sequence ρ is observed if each μ_i occurs at time t_i .

Definition 2 A timed input-output transition system is a 7-tuple $(Q, \text{In}, \text{Out}, \text{Tr}, q_0, l, u)$ where:

- $(Q, \text{In}, \text{Out}, \text{Tr}, q_0)$ is an IOTS.
- $l \in \mathbb{N}^{\text{Tr}}$ is the minimal-delay function.
- $u \in (\mathbb{N} \cup \{\infty\})^{\text{Tr}}$ is the maximal-delay function such that $\forall \tau \in \text{Tr} : l(\tau) \leq u(\tau)$.

By convention we assume that $\forall n \in \mathbb{N} : n \leq \infty$.

Each TIOTS defines a set of accepted timed sequences. The analog-timed trace $(\mu_1, t_1) \cdots (\mu_n, t_n)$ is *accepted* by $S = (Q, \text{In}, \text{Out}, \text{Tr}, q_0, l, u)$ if there exists a path $q_{i_0} \xrightarrow{\mu_1} q_{i_1} \cdots q_{i_{n-1}} \xrightarrow{\mu_n} q_{i_n}$ in S such that $q_{i_0} = q_0$ and $\forall j = 1, \dots, n : l(\tau) \leq t_j - t_{j-1} \leq u(\tau)$ where $\tau = q_{i_{j-1}} \xrightarrow{\mu_j} q_{i_j}$ and $t_0 = 0$. For each state q of S , the duration of authorized stay at q must not exceed the maximum duration $\text{max}_d(q) = \text{Max}\{u(\tau) \mid \tau = (q, \mu, q') \in \text{Tr}\}$ where “Max” is the maximum operator.

Let cl be a variable ranging over \mathbb{R} which measures the time elapsed since the system S has reached its current state. Variable cl is reset as soon as a new state is visited. We say that S is occupying the *global state* $\langle q, \text{cl} \rangle$. Clearly for each global state $\langle q, \text{cl} \rangle$, we have $\text{cl} \leq \text{max}_d(q)$.

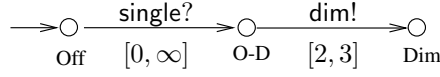


FIG. 3 – An example of a TIOTS.

We define two types of *valid transitions* between global states: (1) *Timed transitions*: for $d \in \mathbb{R}$, if $\text{cl} + d \leq \max_d(q)$ then $\langle q, \text{cl} \rangle \xrightarrow{d} \langle q, \text{cl} + d \rangle$. (2) *Discrete transitions*: for $\mu \in L$ and $\tau = (q, \mu, q') \in \text{Tr}$, if $l(\tau) \leq \text{cl} \leq u(\tau)$ then $\langle q, \text{cl} \rangle \xrightarrow{\mu} \langle q', 0 \rangle$. An example of a TIOTS is given in Figure 3. That is the timed version of one possible path of the IOTS example shown in Figure 1.

4. Analog-Clock Real-Time Testing

An *analog-timed trace* is an element of $(L \cup \mathbb{R})^*$. Clearly a unique analog-timed trace corresponds to each analog-timed sequence. For instance for $\rho = (\mu, 0.3)(\nu, 5.6)(\nu, 8.4)$, the corresponding analog trace is $\sigma = 0.3 \mu \ 5.3 \nu \ 2.8 \nu$.

The analog-timed trace $\sigma = t_0 \mu_1 t_1 \cdots \mu_n t_n$ is said to be a *valid analog-timed trace* of the TIOTS $S = (Q, \text{In}, \text{Out}, \text{Tr}, q_0, l, u)$ if there exist $q_{i_0}, \dots, q_{i_n} \in Q$ such that $q_{i_0} = q_0$ and

$$\langle q_{i_0}, 0 \rangle \xrightarrow{t_0} \langle q_{i_0}, t_0 \rangle \xrightarrow{\mu_1} \langle q_{i_1}, 0 \rangle \xrightarrow{t_1} \langle q_{i_1}, t_1 \rangle \cdots \xrightarrow{t_n} \langle q_{i_n}, t_n \rangle$$

is a sequence of valid transitions of S . In this case, we will use the following notation $\langle q_{i_0}, 0 \rangle \xrightarrow{\sigma} \langle q_{i_n}, t_n \rangle$ and $\langle q_{i_0}, 0 \rangle \xrightarrow{\sigma}$.

For $q \in Q, \text{cl} \in [0, \max_d(q)]$, \mathcal{P} a set of global states and $\sigma \in (\mathbb{R} \cdot L)^* \cdot \mathbb{R}$:

- $\langle q, \text{cl} \rangle \text{ after } \sigma =_{df} \{ \langle q', \text{cl}' \rangle \mid \langle q, \text{cl} \rangle \xrightarrow{\sigma} \langle q', \text{cl}' \rangle \}$.
- $\text{out}(\langle q, \text{cl} \rangle) =_{df} \{ \mu \in \text{Out} \mid \langle q, \text{cl} \rangle \xrightarrow{\mu} \} \cup \{ d \in \mathbb{R} \mid \langle q, \text{cl} \rangle \xrightarrow{d} \}$.
- $\text{TTraces}(\langle q, \text{cl} \rangle) =_{df} \{ \sigma \in (\mathbb{R} \cdot L)^* \cdot \mathbb{R} \mid \langle q, \text{cl} \rangle \xrightarrow{\sigma} \}$.
- $\text{DTraces}(\langle q, \text{cl} \rangle) =_{df} \{ [\sigma] \mid \sigma \in \text{TTraces}(\langle q, \text{cl} \rangle) \}$.
- $\text{out}(\mathcal{P}) =_{df} \bigcup_{\langle q, \text{cl} \rangle \in \mathcal{P}} \text{out}(\langle q, \text{cl} \rangle)$.

Let $\text{Spec} = (Q, \text{In}, \text{Out}, \text{Tr}, q_0, l, u)$ (specification) and $\text{Imp} = (Q', \text{In}, \text{Out}, \text{Tr}', q'_0, l', u')$ (implementation) be two TIOTS. $\text{Imp tioco Spec} =_{df} \forall \sigma \in \text{TTraces}(\langle q_0, 0 \rangle) : \text{out}(\langle q'_0, 0 \rangle \text{ after } \sigma) \subseteq \text{out}(\langle q_0, 0 \rangle \text{ after } \sigma)$.

An *analog-clock tester* for a specification Spec over L is a total function

$$T : (L \cup \mathbb{R})^* \rightarrow \text{In} \cup \{\text{wait}, \text{pass}, \text{fail}\}.$$

In this case, the tester knows exactly at which time each action took place. As for the untimed case, the tester can decide either to send an input to the implementation or to let time elapse or to emit verdict *pass* or *fail*.

There are cases where an analog-clock tester can be represented as a particular type of TIOTS, so-called *TIOTS tester*. A TIOTS tester has two distinct types of states, namely *output-states* and *input-states*. The outgoing edges from an output-state are all labeled with output-actions. When the tester is occupying such a state, it just waits for outputs coming from the implementation and reacts accordingly. Moreover a maximal delay is associated with each output-state.

For the second type of states, each state must have exactly one outgoing edge labeled with an input-action and all the other edges must be labeled with output-actions. For each input-state, the tester has an input-action that it must send to the implementation at a precise timing.

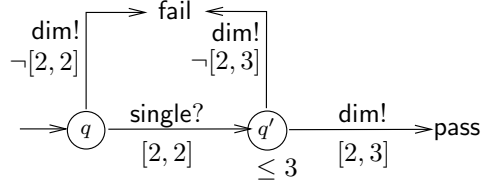


FIG. 4 – An example of a TIOTS tester.

When the tester occupies some input-state, it waits till the time for sending the corresponding input happens. If an output is received before that time the tester reacts accordingly. Otherwise, it will send the specific input-action to the implementation at the precise chosen time and continues the execution of the test strategy accordingly.

Each TIOTS tester has two particular states called *verdict-states*: **pass** and **fail**. That is a **pass** or a **fail** verdict is to be emitted by the tester as soon as one of these states is reached. Moreover when occupying an output-state the tester has to emit a **fail** verdict as soon as the maximal delay associated with this output-state is exceeded.

A TIOTS Tester may have edges of the form

$$q \xrightarrow[\neg[l, u]]{\mu} \text{fail}$$

where q is a state of the tester, μ is an output-action and $l, u \in \mathbf{N}$. Such an edge means that a **fail** verdict has to be emitted if the tester is occupying state q and action μ happens at a time outside the interval $[l, u]$.

An example of TIOTS tester is shown in Figure 4. That is a possible tester for the TIOTS of Figure 3. The state q of this TIOTS tester is an input-state while the state q' is an output-state. According to this TIOTS, the input-action **single?** must be executed exactly two time-units after the beginning of the experiment. The duration of the stay at state q' must not exceed three time-units.

5. Action Refinement

The high-level actions to be refined are in $L_\delta = \text{In} \cup \text{Out}_\delta$ and the low-level actions are in $L_r = \text{In}_r \cup \text{Out}_{r, \delta_r}$, where $\text{Out}_{r, \delta_r} = \text{Out}_r \cup \{\delta_r\}$. Each high-level action μ is refined into a timed-path of the form

$$\text{analog}_{\text{ref}}(\mu) = q_1 \xrightarrow[\text{[}l_1, u_1\text{]}]{\mu_1^r} q_2 \xrightarrow[\text{[}l_2, u_2\text{]}]{\mu_2^r} \cdots q_n \xrightarrow[\text{[}l_n, u_n\text{]}]{\mu_n^r} q_{n+1}$$

where $\mu_i^r \in L_r$ and $l_i, u_i \in \mathbf{N}$ are, respectively, the minimal and maximal delays of the corresponding transition. The sequence $\text{analog}_{\text{ref}}(\mu)$ is called the *analog-clock action refinement* of the high level action μ .

Each high-level input-action in In is refined into a timed-path over low-level input-actions in In_r . Similarly, each high-level output-action in Out is refined into a timed-path over low-level output-actions in Out_r . Moreover we assume that a same given low-level output-action cannot appear in the analog-clock refinements of two distinct high-level output-actions (i.e., a set of low-level output-actions is associated with each high-level output-action). The length of the analog-clock refinement of a given low-level action is the number of edges that compose it. For instance the length of the analog-timed refinement of the action μ above is n .

The refinement of the particular action δ is made as follows:

$$\text{analog}_{\text{ref}}(\delta) = q \xrightarrow[\text{[}\Delta, \Delta\text{]}]{\delta_r} q'$$

where $\Delta \in \mathbf{N}$ is the maximal waiting time constant (e.g., set either by the system designer or the tester). That is Δ time-units must elapse before announcing that a timeout occurs.

μ	$\text{analog}_{\text{ref}}(\mu)$
single?	$q \xrightarrow[\text{[0,}\infty\text{]}]{\text{touch}_r?} q'$
double?	$p \xrightarrow[\text{[0,}\infty\text{]}]{\text{touch}_r?} p' \xrightarrow[\text{[0,1]}]{\text{touch}_r?} p''$
dim!	$r \xrightarrow[\text{[2,3]}]{\text{dim}_r!} r'$
bright!	$s \xrightarrow[\text{[2,3]}]{\text{bright}_r!} s'$
off!	$t \xrightarrow[\text{[2,3]}]{\text{off}_r!} t'$
$\delta!$	$u \xrightarrow[\text{[4,4]}]{\delta_r!} u'$

FIG. 5 – Analog-time refinement rules for the lighting device example.

The refinement technique above carries over in a natural way to the refinement of an IOTS S into a TIOTS S_r . Given a transition $\tau = q \xrightarrow{\mu} q'$ of S , if

$$\text{analog}_{\text{ref}}(\mu) = q_0 \xrightarrow[\text{[}l_0, u_0\text{]}]{\mu_0^r} q_1 \xrightarrow[\text{[}l_1, u_1\text{]}]{\mu_1^r} \cdots q_n \xrightarrow[\text{[}l_n, u_n\text{]}]{\mu_n^r} q_{n+1}$$

then τ is replaced within S_r with the sequence

$$\text{analog}_{\text{ref}}(\tau) = q \xrightarrow[\text{[}l_0, u_0\text{]}]{\mu_0^r} q_{\tau,1} \xrightarrow[\text{[}l_1, u_1\text{]}]{\mu_1^r} \cdots q_{\tau,n} \xrightarrow[\text{[}l_n, u_n\text{]}]{\mu_n^r} q'$$

where q and q' are the same states as in τ and $q_{\tau,1}, \dots, q_{\tau,n}$ are the names of the new added states (they must be distinct from the already added ones). Notice that q and q' may be the same. If $S = (Q, \text{In}, \text{Out}, \text{Tr}, q_0)$ then $S_r = \text{analog}_{\text{ref}}(S) = (Q_r, \text{In}_r, \text{Out}_{r,\delta_r}, \text{Tr}_r, q_0^r, l_r, u_r)$ such that: $Q_r = Q \cup \{q_{\tau,i} \mid \tau \in \text{Tr}\}$. $\text{Tr}_r = \bigcup_{\tau \in \text{Tr}} \text{analog}_{\text{ref}}(\tau)$; $q_0^r = q_0$. l_r and u_r are the functions encoding, respectively, the minimal and maximal delays appearing in $\text{analog}_{\text{ref}}(\tau)$ for all $\tau \in \text{Tr}$.

The *digital-time action refinement* of μ is defined as follows:

$$\text{digital}_{\text{ref}}(\mu) = \{t_0 \mu_0 \cdots t_n \mu_n \mid \forall i : t_i \in \mathbf{N} \wedge l_i \leq t_i \leq u_i\}.$$

We consider again the lighting-device example. The model given in Figure 1 is indeed a simplified version of the real implementation. In fact, the lighting-device has only one input-action *touch*. The input high-level actions are introduced for ease of modeling. The lighting device disposes of a touch-sensitive pad. The user interface logic is as follows: a simple unique touch corresponds to the high-level input-action *single* and two “quick” consecutive touches correspond to the high-level input-action *double*. The maximum delay between two consecutive touches considered as a double touch is fixed to 1 time-unit.

As already shown in Figure 3, minimum and maximum delays for the lamp to change intensity need to be introduced as well. We assume that moving from one intensity level to another takes between 2 and 3 time units. Furthermore, we fix the maximal waiting time constant Δ to 4 time-units. The (analog-time) refinement rules for this example are summed up within the table shown in Figure 5.

6. Analog-clock Tester Generation

The method we propose for generating analog-clock testers consists in transforming an initially untimed test “ T ” into a timed tester “ T_r ” using the refinement techniques introduced so far. We assume that T is given as a tree. An internal node of T is called an *input-node* if it has a unique outgoing edge labeled with an input-action. Similarly, an internal node of T is called an *output-node* if all its outgoing edges are labeled with output-actions.

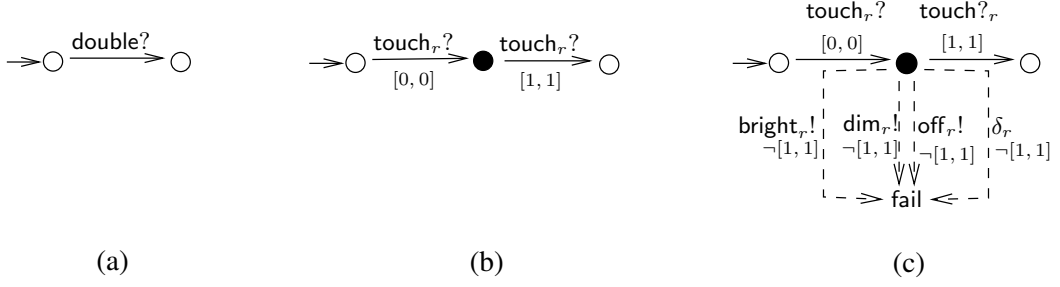


FIG. 6 – The different steps for refining an edge of a test case labelled with the low-level input-action *double*: (a) the original high-level edge, (b) the corresponding low-level timed path, (c) adding the edges leading to fail.

To obtain T_r we proceed as follows:

- S 1. We make a copy of T (we already call it T_r);
- S 2. We omit all edges of T_r leading to fail;
- S 3. We refine all edges of T_r and add progressively the new edges leading to fail.

Step “S 3” is achieved as follows:

- (I) For an input-node q , let $e = q \xrightarrow{\mu} q'$ be the outgoing edge from q :
 1. We choose a possible digital-time action-refinement $\mu_r = t_0\mu_0t_1\mu_1 \cdots t_n\mu_n$ of μ (i.e., $\mu_r \in \text{digital}_{\text{ref}}(\mu)$).
 2. We replace e in T_r with the path $q_0 \xrightarrow[t_0, t_0]{\mu_0} q_1 \xrightarrow[t_1, t_1]{\mu_1} q_2 \cdots q_n \xrightarrow[t_n, t_n]{\mu_n} q_{n+1}$, where $q_{i_0} = q$, $q_{n+1} = q'$ and the other q_j 's are new names not used in the past.
 3. For each $t_j \neq 0$ and $\nu \in \text{Out}_{r, \delta_r}$, we add a new edge $q_j \xrightarrow[\neg[t_j, t_j]]{\nu} \text{fail}$ to T_r .¹

We consider the test case of Figure 2. We refine the edge emanating from the initial node of the test tree. This edge is labelled with the high-level input-action *double*. The low-level digital-timed input-trace we choose to refine this input-action with is “0 *touch_r*? 1 *touch_r*?”. The different steps of the refinement of this edge are given in Figure 6. To make it clearer the new added nodes are filled in with black and the new added edges leading to fail are drawn as dashed arrows.

- (II) For an output-node q , let

$$\beta = \{\mu \in \text{Out}_{\delta} \mid \exists q' : q \xrightarrow{\mu} q' \text{ is an edge of } T_r\}$$

be the set of labels of the outgoing edges from q . Moreover, let

$$\beta_r = \{\mu_0 \in \text{Out}_{r, \delta_r} \mid \exists \mu \in \beta : \text{analog}_{\text{ref}}(\mu) = q \xrightarrow[l_0, u_0]{\mu_0} q_1 \cdots q_n \xrightarrow[l_n, u_n]{\mu_n} q'\}$$

be the set of first low-level output-action appearing in the refinement of each element of β . Similarly, we define

$$U_r = \{u_0 \in \mathbf{N} \mid \exists \mu \in \beta : \text{analog}_{\text{ref}}(\mu) = q \xrightarrow[l_0, u_0]{\mu_0} q_1 \cdots q_n \xrightarrow[l_n, u_n]{\mu_n} q'\}.$$

¹This step allows to reject all the low-level output-actions that may be observed during the execution of refined timed trace μ_r .

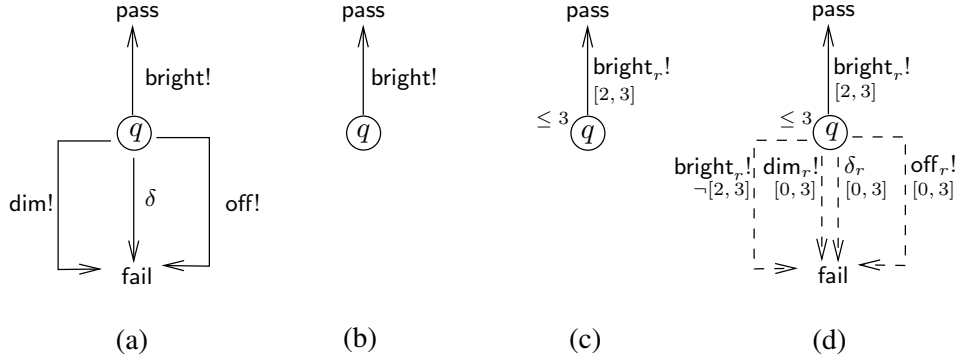


FIG. 7 – The different steps for refining the edges emanating from an output-node: (a) the original edges emanating from the output-node q , (b) removing all the high-level edges leading to fail, (c) replacing the edge labelled with **bright** with its digital-timed refinement and calculating the maximal delay, (d) adding the low-level edges leading to fail.

Then:

1. We associate with q the maximal delay $d_q = \text{Max} \{u \mid u \in U_r\}$.
2. For each $\mu_r \in \text{Out}_{r,\delta_r} \setminus \beta_r$, we add a new edge $q \xrightarrow[\text{fail}]{\mu_r, [0, d_q]}$.
3. For each $\mu \in \beta$, we replace $q \xrightarrow{\mu} q'$ with its refinement $\text{analog}_{\text{ref}}(\mu) = q \xrightarrow[\text{fail}]{\mu_0, [l_0, u_0]} q_1 \cdots q_n \xrightarrow[\text{fail}]{\mu_n, [l_n, u_n]} q'$ and we add $q \xrightarrow[\text{fail}]{\mu_0, \neg[l_0, u_0]}$.
4. For each added state q_i , we associate a maximal delay $d_{q_i} = u_i$.
5. For each added state q_i , we add $q_i \xrightarrow[\text{fail}]{\mu_i, \neg[l_i, u_i]}$.
6. For each added state q_i and each $\mu_r \in \text{Out}_{r,\delta_r} \setminus \{\mu_i\}$, we add $q_i \xrightarrow[\text{fail}]{\mu_r, [0, u_i]}$.

Again, we consider the test case of Figure 2. Now, we refine the edge labelled with the high-level output-action **dim**. We call q the node from which this edge emanates. Clearly, q is an output-node. The set β associated with q is the singleton set $\{\text{bright}\}$. The different steps of the refinement of the edge labelled with **bright** are given in Figure 7. As for Figure 6, the new added nodes are the black ones and the edges leading to fail are drawn as dashed arrows.²

7. Conclusion

In this work, we succeeded to make the link between: (I) The framework for untimed testing based on the model of IOTS and the (untimed) conformance relation ioco [7] and; (II) Our framework for real-time testing based on the model of TIOTS and the timed conformance relation tioco [3]. We have proposed a new method for generating analog-clock testers based on the so-called action refinement techniques.

The main practical benefit to derive from action-refinement, in this case, is to save memory space needed to build and to store tests. The proposed method can be shown to be sound. On the practical side, the other important contribution of this work is the simplified way for both modelling and testing real-time systems.

Possible future directions of this work are: (1) To extend the method from TIOTS to general timed automata specifications. (2) To consider more general action refinement rules (e.g., an action is refined into a tree rather than a "linear" trace).

²In order not to overload the figure we draw only one dashed arrow for several edges leading to fail.

References

- [1] S. Bensalem, M. Krichen, L. Majdoub, R. Robbana, and S. Tripakis. A simplified approach for testing real-time systems based on action refinement. In *ISoLA Workshop On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA'07)*, Poitiers, France, 2007. [2](#)
- [2] R. Gorrieri and A. Rensink. Action refinement, 2000. [2](#)
- [3] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *11th International SPIN Workshop on Model Checking of Software (SPIN'04)*, volume 2989 of LNCS. Springer, 2004. [1](#), [3](#), [9](#)
- [4] M. Krichen and S. Tripakis. Real-time testing with timed automata testers and coverage criteria. In *Formal Techniques, Modelling and Analysis of Timed and Fault Tolerant Systems (FORMATS-FTRTFT'04)*, volume 3253 of LNCS. Springer, 2004. [1](#)
- [5] Mila E. Majster-Cederbaum and Jinzhao Wu. Action refinement for true concurrent real time. In *ICECCS*, pages 58–68. IEEE Computer Society, 2001. [2](#)
- [6] Mila E. Majster-Cederbaum, Jinzhao Wu, and Houguang Yue. Refinement of actions for real-time concurrent systems with causal ambiguity. *Acta Inf.*, 42(6-7):389–418, 2006. [2](#)
- [7] Jan Tretmans. Testing concurrent systems: A formal approach. In *CONCUR'99*, volume 1664 of LNCS. Springer, 1999. [2](#), [4](#), [9](#)
- [8] Machiel van der Bijl, Arend Rensink, and Jan Tretmans. Action refinement in conformance testing. In Ferhat Khendek and Rachida Dssouli, editors, *TestCom*, volume 3502 of Lecture Notes in Computer Science, pages 81–96. Springer, 2005. [2](#)
- [9] Rob van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Inf.*, 37(4-5):229–327, 2000. [2](#)