TTCN

Sébastien Salva Hacène Fouchal Eric Petitjean

LERI-*RESYCOM*
Fax : (33) 3 26 91 33 97
Université de Reims Champagne-Ardenne,
Moulin de la Housse, BP 1039,
51687 Reims Cedex 2, France
e-mail:{Sébastien.Salva, Hacene.Fouchal, Eric.Petitjean}@univ-reims.fr

Abstract

Testing real-time systems or protocols containing time constraints seems difficult, without formal methods, on account of the notion of time which cannot be observed and directly controlled. But ensuring a functional behavior , respecting time constraints becomes very important, especially for critical systems. So, we propose a novel testing methodology which aims to check if timed systems, modeled with timed automata, can respect temporal and behavior properties of test purposes. A synchronous product is generated between them for obtaining timed test cases. Then, we describe, in details, a method to execute the test of an timed implementation. Finally, we express test execution in TTCN.

Key-words : Timed automata, Clock Region, test cases, TTCN.

1 Introduction

Nowadays, creating real-time and critical systems becomes more and more difficult and expensive, on account of their complexity which grows year after year and on account of the addiction of multimedia and time. Moreover, consumers want final working products, especially when a system is a multimedia protocol, an air traffic system or a real-time system. Beta testing could be a solution to check the correctness of a system, although it is cost (for management) and it low covers infrequently used parts of the system. Despite these problems, beta testing is still widely used in practice. For critical systems, formal methods and better coverage is needed. So, in this field, we usually deal with two main validation techniques:

- the verification technique, which handles a specification and tries to prove its correctness (in this case the system can be seen as a white box),
- the conformance testing technique, which uses an implementation of the system and tries to find incorrectness on it without having any information about its structure (in this case the implementation can be seen as a black box). More clearly, to check if the implementation's behavior is the same as the specification's behavior, a set of sequences of actions (called *'test cases'*) is generated and applied to the implementation under test (I.U.T). For timed systems, we also need to check whether the implementation actions can be executed in the same time intervals than the specification's ones.

By testing an action of a system, we can obtain the following results: **conclusive** which implies your implementation is conformed, **fail** which implies the action cannot be performed in the implementation and **inconclusive** which implies at least one action cannot be tested.

*** parler des autres meth ***

In this paper, we propose a new testing methodology for timed systems, modeled with timed automata ([AD94]. This one is test purpose based approach. Test purpose are expressed with sequences of actions and time constraints, which represents temporal and behavior properties that must be verified on the implementation.

We present in section 3, the timed test cases generation which is summarized as follow: a part of the specification containing the test purpose and this last one are translated into region graphs to explicit relations between clocks and the elapse of time. Region graph are equivalent representation of timed automata, often used ([]) and are the lonely representation which able to compare different clocks in timing constraints, but they contains unusual information, which increase costs of the generation, so they are transformed into TIORA. Then, a synchronous product is computed to obtain a graph, respecting the specification and containing the test purpose. Test cases are the paths of this graph.

We describe in section 4, in details, test cases execution with a timed implementation, and we show how actions are tested in time intervals. We also show how timed test cases can be translated into TTCN.

The remainder of this paper is structured as follow: Related works and notations are expressed in Section 2. Section **??** is devoted to a timed conformance relation based on trace. We give some remarks about the methodology and its complexity in section **??**. Section 5 illustrates an example of faulty implementation. Finally, in section 6, we conclude.

2 Related works

2.1 Timed system model

2.1.1 Timed Input Output automata (TIOA)

Timed automata [AD94] are graphs representing timed systems during their executions. The first state is called the initial state. The choice of the next state depends on the action to execute (represented by a transition, labeled with a symbol) and on the moment where this action is executed. To represent time in timed systems, a set of clocks is associated to the automaton. Each clock is represented by a real value (dense time representation) and grows strictly monotonically. All clocks are set to 0 in the initial state. Clocks can be reset on any transition. At each time, the clock values give us the time past for its previous reset. Finally, most of the transitions contain clock constraints. To execute a transition, all the clocks of the system must satisfy these ones.

Definition 2.1 (Clock constraint [AD94]) *Let C_A be a finite set of clocks, and $x_i \in C_A$. A clock constraint δ over x_i is a boolean expression of the form $\delta = x_i \leq c \mid c \geq x_i \mid \neg\delta \mid \delta_1 \wedge \delta_2$ where $c \in \mathbb{Q}$.*

The set of clock constraints over C_A is denoted $\Phi(C_A)$.

Timed Input Output automata are extended timed automata where symbols are divided into input symbols and output symbols.

Definition 2.2 (Timed Input Output automata) *A TIOA A is defined as a tuple $< \Sigma_A, S_A, s_A^0, C_A, E_A >$, where:*

- Σ_A *is a finite alphabet,*
- S_A *is a finite set of states,*
- s_A^0 *is the initial state,*
- C_A *is a finite set of clocks,*
- $E_A \subseteq L_A \times (\{?, !\} \times \Sigma_A) \times 2^{C_A} \times \Phi(C_A)$ *is the finite set of transitions.*
  An input symbol begins with "?" and an output one begins with "!".
  A tuple $< l, l', a, \lambda, G >$ represents a transition from state l to state l', labeled with the symbol a. The subset $\lambda \subset C_A$ gives the clocks to be reset within this transition, and G is a clock constraint over C_A.

A transition $S_i \xrightarrow{?x} S_j$, labeled by the input symbol "?x", models an input action provoked by the external environment of the implementation. "?x" must be given to it in order that the action could be executed. A transition $S_i \xrightarrow{!x} S_j$, labeled by the output symbol "!x", represents an output action provoked only by the implementation. Its execution is indicated by the sending of "!x" by the implementation.

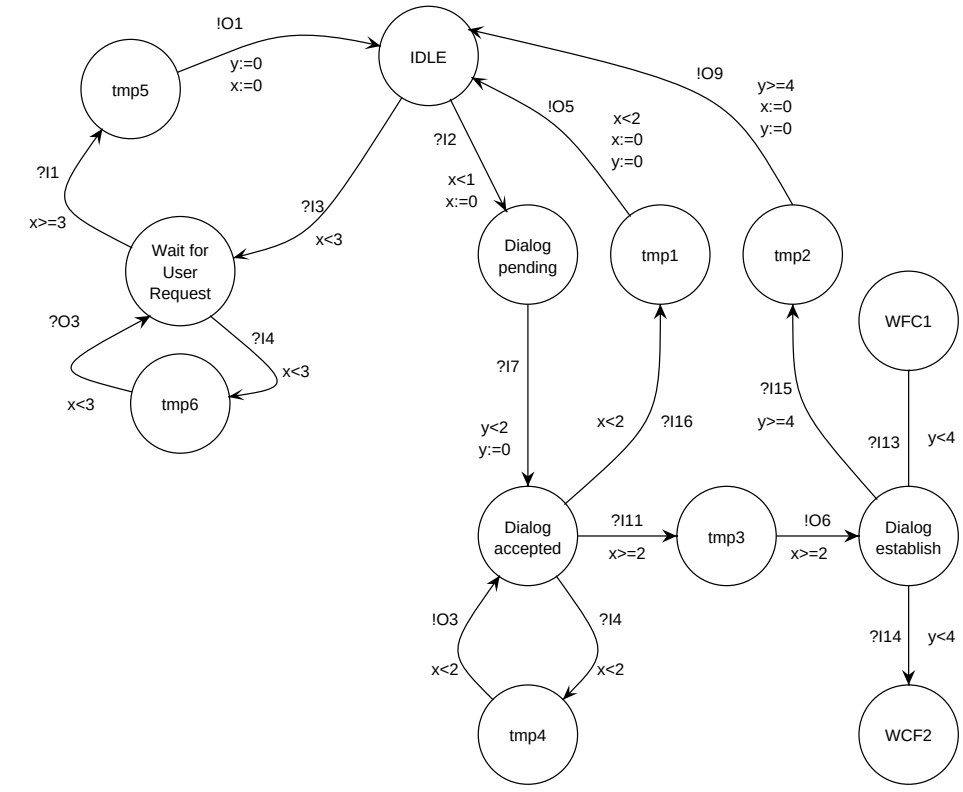

Figure 1: A part of the MAP-GSM protocol, modeled with TIOA.

An example of TIOA, modeling a part of the MAP-GSM protocol, is illustrated in figure 1. If we consider the transition $Tmp2 \xrightarrow{!O9} IDLE$, the clock constraint $y >= 4$ should be satisfied when the system sends !O9 (after this execution , x and y will be reset to 0).

2.1.2 Region Graph

Definition 2.3 (Clock valuation [AD94]) *A clock valuation over a set of clocks C_A is a mapping v that assigns to each clock $x \in C_A$ a value in $\mathbb{R}^+$, called clock value. We denote the set of clock valuation by $V(C_A)$.*
A clock valuation v satisfies a clock constraint G, denoted $v \models G$, if and only if G is evaluated to true under v.
For $d \in \mathbb{R}^+$, $v+d$ denotes the clock valuation which assigns a value $v(x)+d$ to each clock x. For $X \subseteq C_A$, $[X \rightarrow d]v$ denotes the clock valuation for C_A which assigns d to each $x \in X$, and agrees with v over the rest of the clocks.

So, if n is the number of clocks of A, a clock valuation is a n-tuple of clock values. With this previous definition, we can say the future behavior of a timed system is determined by its states and by the clock valuations over C_A. This motivates a new representation of timed systems, by combining these states and these clock valuations. However, the set of clock values, is infinite, because we deal with dense time representation. An equivalence relation is defined in [AD94] in order to gather clock values which have the same integral parts.

For any $t \in \mathbb{R}^+$, $fract(t)$ denotes the fractional part of t and $\lfloor t \rfloor$ denotes the integral part of t.

Definition 2.4 (Clock Region [AD94]) *Let $A =< \Sigma_A, L_A, l_A^0, C_A, E_A >$ be a timed automaton. For each $x \in C_A$, let c_x be the largest integer c such that $(x \leq c)$ or $(c \leq x)$ is a subformula of some clock constraints appearing in E_A.*

The equivalence relation $\sim$ is defined over the set of all clock interpretations for C_A; $v \sim v'$ iff all the following conditions hold:

- *For all $x \in c_A$, either $\lfloor v(x) \rfloor$ and $\lfloor v'(x) \rfloor$ are the same, or both $v(x)$ and $v'(x)$ are greater than c_x.*
- *For all $x, y \in C_A$ with $v(x) \leq c_x$ and $v(y) \leq c_y$, $fract(v(x)) \leq fract(v(y))$ iff $fract(v'(x)) \leq fract(v'(y))$.*
- *For all $x \in C_A$ with $v(x) \leq c_x$, $fract(v(x)) = 0$ iff $fract(v'(x)) = 0$.*

A clock region for A is an equivalence class of clock valuations induced by $\sim$.

We will use $[v]$ the clock region to which v belongs.
Clock regions can also be seen as time intervals, depending on several clocks, and they can be illustrated by polyhedrons with several vertices. A polyhedron is defined with inequations. For a clock region R, we also say a clock valuation $c \in R$ iff c satisfies all the inequations of R.

Definition 2.5 (Time Successor) *A clock region R' is a time-successor of a clock region R iff for each $v \in R$, there exists a positive $t \in \mathbb{R}$ such that $v + t \in R'$*

Region graphs are equivalent representations of timed automata where timed constraints are represented by one or several clock regions. A region graph state is a tuple containing one state of the TIOA and one clock region. This new model allows to distinguish each time interval during which an action may be executed. An algorithm to transform TIOA into region graphs is given in [AD94].

Definition 2.6 (Region Graph) *Let $A = (\Sigma_A, S_A, s_A^0, C_A, E_A)$ be a timed input output automaton. A region graph of A is an automaton $RA = (\Sigma_{RA}, S_{RA}, s_{RA}^0, E_{RA})$ where:*

- $\Sigma_{RA} = \Sigma_A \cup \delta$, *where δ represents the elapse of time,*
- $S_{RA} \subseteq \{\langle s, [v]\rangle \mid s \in S_A \wedge v \in V(C_A)\}$
- $s_{RA}^0 = \langle s_A^0, [v_0]\rangle$ *where $v_0(x) = 0$ for all $x \in C_A$*
- R_A *has a transition, $q \xrightarrow{a}_{RA} q'$, from state $q(\langle s, [v]\rangle)$ to state $q'(\langle s', [v']\rangle)$ with the symbol a, iff either*
  - *$a \neq \delta$ and there is a transition $(s, s', a, \lambda, G) \in E_A$ and $d \in \mathbf{R}^+$ such that $(v + d) \models G$ and $v' = [\lambda \mapsto 0](v + d)$,*
  - *$a = \delta$, $s \neq s'$ and there exists $d \in \mathbf{R}^+$ such that $v' = v + d$.*

Our definition is slightly different from the definition in [AD94], because we take into account the delay to reach a clock region from another one, which is modeled with delay transitions. Region graphs are often used in verification and testing techniques for timed systems [AD94, DY96, SSb00]. Moreover, some algorithms, allowing to minimize region graphs have been written [YL93, ACH+92], which generates the portion of the minimized system that is reachable in polynomial time. Consequently, all clock regions, in which the same actions can be executed, are gathered into one clock region. So, the number of states of region graphs is strongly reduced, what reduces test costs.

2.2 Test Architecture

In order to apply the test cases on the implementation, the authors of [EnFDE97] propose a first test architecture, inspired by the well-known one defined in [ISO91]. But in [PF99], we suggested another architecture which is more realistic. In fact, the main idea is to remember that what we want to test is a temporal behaviour, however it may have been implemented. This behaviour is represented in the specification by clock constraints and clock resets, and we do not know *a priori* what their equivalent is in the specification, and not even if they have any equivalent at all. The aim is to test whether the implementation accept the inputs at every moment it should, according to the specification, and if the outputs it produces occur at the expected moments. Consequently, and since the only temporal information we have at our disposal lie in the specification, we must treat the temporal behavior in our tests as a part of what is obtained from the specification, i.e. the tester, and only as a part of the tester. This architecture is described by the figure exposed in figure 2.

In this architecture, we make no assumption about how time is modeled inside the IUT, which now becomes an actual black box. The new tester is composed of two parts, communicating with one another : the clock part, which contains the clocks appearing in the specification, and the behaviour part, whose role is to communicate with the implementation through the single PCO, i.e. to send inputs to the IUT and receive outputs from it.

Since the clocks are now a part of the tester and belong no more to the implementation, all the actions involving the clocks must now be performed by the tester.

The behaviour part of tester may, like the behaviour part of the IUT in the former architecture, ask the clock part at any time for the value of one or more clocks and receive instantaneously the answer, but this communication is now internal to the tester, and no more between the tester and the implementation.

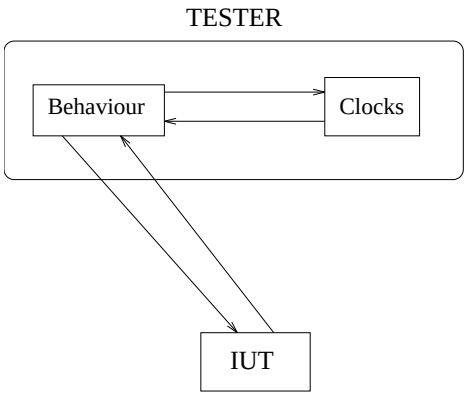

Figure 2: A specific test architecture

On the other hand, the actions to be performed on the clocks, i.e the resets, do not involve the implementation anymore. The tester, about which we do know it contains clocks, performs the resets independently from the IUT. This situation implies that the tester has to know which clocks are to be reset to zero in each transition. Consequently, we must keep, in the whole process of test generation, the sets of clocks to be reset (denoted as λ in the definition 2.2), in the transition labels. It will be interpreted during the test execution as an action to be performed by the behaviour part of the tester upon its clock part of the tester.

3 Test Cases Generation

This part deals with the test cases generated from the specification and used for testing. But previously, we need to make some hypothesis on TIOA.

4 Test Cases Execution

During the test, temporal and behavior properties of the implementation are checked, by means of test cases. These ones contains actions which must be checked in the PASS interval, expressing timing constraints of the test purpose and of the specification. Unfortunately, clock regions are dense representation of time: that is they contains infinite tuple of values, which cannot be all checked. Therefore, we consider the fault coverage of the I.U.T is sufficient by applying these three cases:

- Let "?A" be an input action to test, in a PASS interval R, and let $v_{init} \in R$ be the first clock valuation reached by the clocks, during the test. The tester can send "A" to the I.U.T for any clock values reached by the clocks.

  We propose that the tester sends "A" to the I.U.T *as soon as possible and the latest possible*, that is for v_{init} and the last clock valuation $v_{final} \in R$, reached by the clocks. For reaching v_{final}, an elapse of time is necessary and it must be computed on the fly during the test, on account of the dynamic behavior of the I.U.T (see section 4.1.2 for computation).

- Let "!A" be an output action to test, R a PASS interval and R' an INCONCLUSIVE interval. The tester can only wait the receipt of the symbol "A" from the I.U.T. So if "A" is received whereas the tuple of clocks equals one clock valuation of the PASS interval, then the I.U.T respects the specification

and the test purpose. For a clock valuation of the INCONCLUSIVE interval, only the specification is respected. Otherwise, the I.U.T does not respect both (FAIL interval).

- For an elapse of time "δ", allowing to reach the next clock region in which the next action is tested, none test is needed. However, the tester must compute on the fly the duration of this elapse of time and must simply wait. (see section 4.1.1 for computation)

Consequently, we consider input actions are checked for two clock valuations of the PASS interval. So, a test case must be applied to the I.U.T 2^n times with n the number of input action. To express all **of** these cases, we develop a test case into a tree, called *Execution Tree*.

4.1 Execution Tree

We suppose that we have a test sequence obtained from the synchronous product of two TIORA. The following algorithm transforms such a sequence into a tree, called *Execution Tree*, by respecting the three previous assumptions. This tree is composed of TIORA states and of edges labeled by:

- send(A) representing the sending of the input symbol ?A by the tester
- recv(A) representing the sending of the output symbol !A by the I.U.T to the tester
- *wait* representing the time needed to reach the last clock valuation reached by the I.U.T clocks in a clock region. This time is computed during the test.
- δ representing the elapse of time needed to reach the next clock region. This time is also computed during the test.

Algorithm

Const-tree(SEQ,$s \xrightarrow[Rt]{A,PASS(R),INCONCLUSIVE(R')} s'$)

IF A=δ

THEN $\begin{cases} \text{Add } \xrightarrow{\delta} s' \\ \text{Cons-tree(SEQ,next transition of SEQ)} \end{cases}$

%Input action
IF A=?I

IF two clock valuations can be reached by the tuple of clocks

THEN $\begin{cases} \text{Add a branch } \xrightarrow{wait} \xrightarrow[Rt]{send(I)} s' \\ \text{Cons-tree(SEQ,next transition of SEQ)} \\ \text{Add a second branch } \xrightarrow[Rt]{send(I)} s' \\ \text{Cons-tree(SEQ,next transition of SEQ)} \end{cases}$

ELSE $\begin{cases} \text{Add } \xrightarrow[Rt]{send(I)} s' \\ \text{Cons-tree(SEQ,next transition of SEQ)} \end{cases}$

%Output action
IF A=!O

THEN $\begin{cases} \text{Add a branch } \xrightarrow[Rt]{recv(O),PASS(R),INCONCLUSIVE(R')} s' \\ \text{Cons-tree(SEQ,next transition of SEQ)} \end{cases}$

Finally, each branch of this tree can be given to the tester. This one can execute them directly because each elapse of time, each sending or receipt of action are indicated. The elapse of time *wait* and δ are computed as follow:

4.1.1 Elapse of time needed to reach the next clock region

Consider the sequence $s_1 \xrightarrow[Rt]{A,R_1} s_2 \xrightarrow{\delta} s_3 \xrightarrow[Rt]{B,R_2} s_4$. We need to calculate the elapse of time needed to reach R_2 from any clock valuation, reached after executing the action "A", that is the minimal value d such as $v_{init} + d \in R_2$.

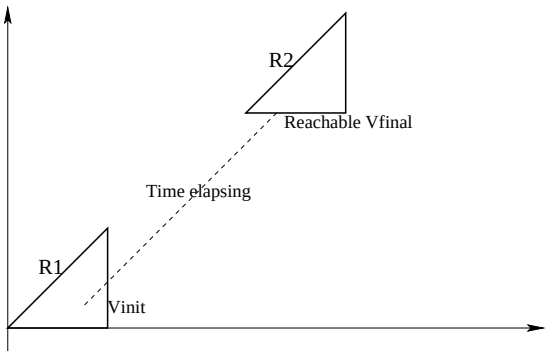

Figure 3: Reaching another clock region in v_{final} from v_{init}

Let $v_{init} = (v_1, ..., v_n)$ be the first clock valuation of R_1 reached by the clocks after the execution of "A". As, these ones grow with the same manner, strictly monotonically, it is easy to prove they take the values of the equation $\begin{cases} x_1 - x_2 = v_1 - v_2 \\ ... \\ x_{n-1} - x_n = v_{n-1} - v_n \end{cases}$

So, the first clock valuation v_{final} reached by the clocks in R_2 is unique and is obtained by resolving the system of inequations

$\Delta = \begin{cases} \text{Inequations of } R_2 \\ x_1 - x_2 = v_1 - v_2 \\ ... \\ x_{n-1} - x_n = v_{n-1} - v_n \end{cases}$

Finally, the elapse of time δ equals to the difference between v_{init} and the minimal solution v_{final} of Δ. An example, illustrating this elapse of time is given in 3.

4.1.2 Elapse of Time needed to reach a clock valuation of a clock region

For input actions, which can be executed in a clock region R, the tester tests them as soon as it can, and for the latest clock valuation v_{final}, reached by the clocks in R, knowing that the clocks gives the values $v_{init} = (v_1, ..., v_n)$. An elapse of time *wait* is needed to reach v_{final}. The figure 4 illustrates this other elapse of time.

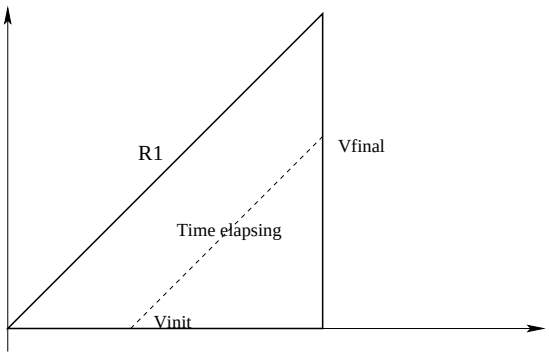

Figure 4: Reaching the last clock valuation v_{final} of a clock region from v_{init}

The time, needed to reach v_{final} can be computed as the previous way. The clocks also take the values of the equation $\begin{cases} x_1 - x_2 = v_1 - v_2 \\ ... \\ x_{n-1} - x_n = v_{n-1} - v_n \end{cases}$

And v_{final} is the maximal solution of the system of inequations

$$\Delta = \begin{cases} \text{Inequations of } R \\ x_1 - x_2 = v_1 - v_2 \\ ... \\ x_{n-1} - x_n = v_{n-1} - v_n \end{cases}$$

Finally, *wait* equals to the difference $v_{final} - v_{init}$.

From the synchronous product, illustrated in figure **??**, we obtain the execution tree of the figure 5

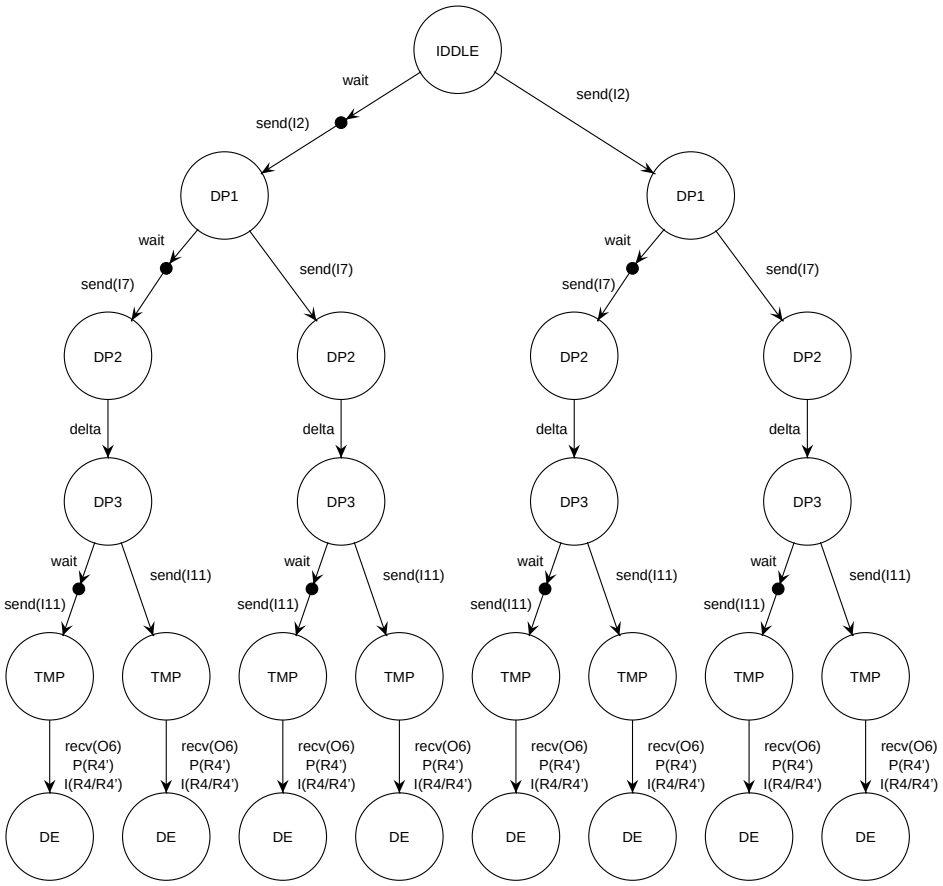

Figure 5: Execution Tree

4.2 Transformation of execution tree in TTCN

TTCN (Tree and Tabular Combined Notation) is a language, normalized [ISO91], expressing test cases, and recognized from most of the present testers. Thus, transforming execution trees into TTCN allows to manage them directly, without constructing a new tester.

Before executing an execution tree branch, the tester clocks $X_1, ..., X_N$ must be started with: $STARTX_1...STARTX_N$.

We also need two macro, the first one, called $C_1(v, R)$ returns the elapse of time $WAIT$, needed to reach the next clock region R from a clock valuation v (compu-

tation in 4.1.1). The second one, called $C_2(v, R)$ returns the two clock valuations $V_{min} = (V_1, ..., V_N)$ and $V_{max} = (V'_1, ..., V'_N)$, reached by the clocks in R, knowing that the tuple of clocks currently equals to v (computation in 4.1.2).

The three following expressions allows to translate a branch of execution tree into TTCN.

4.2.1 TTCN Expression of the elapse of time δ

A branch $s \xrightarrow{\delta} s' \xrightarrow[Rt]{A,R} s''$, expressing the elapse of time needed to reach the next clock region R, is expressed with:

Description	Verdict	Comment
READTIMER X_1 T_1 $\vdots$ READTIMER X_N T_N $C_1(< T_1, ...T_N >, R_2) \rightarrow Wait$ $[WAIT > 0]$?ELAPSE $WAIT$		computation of the elapse of time reaching R_2

4.2.2 TTCN Expression of input action test

Consider the transition $s \xrightarrow[Rt]{?A,R} s'$. The execution tree has two branches $s \xrightarrow[Rt]{send(A)} s'$ and $s \xrightarrow{wait} \xrightarrow[Rt]{send(A)} s'$. So, we obtain in TTCN for the first one:

Description	Verdict	Comment
!A $\vdots$ CANCEL X_i START X_i $\vdots$	Pass	send "A" reset x_i if necessary

For the second branch, it is necessary to calculate the time needed to reach the last clock valuation and to wait. We obtain:

Description	Verdict	Comment
READTIMER X_1 T_1 $\vdots$ READTIMER X_N T_N $C_2(< T_1, ..., T_N >, R) \rightarrow (V1_{min}, ..., VN_{min})(V1_{max}, ..., VN_{max})$?ELAPSE $V1_{max} - T_1$!A $\vdots$ CANCEL X_i START X_i $\vdots$	Pass	reaching v_{final} send "A" reset x_i if necessary

4.2.3 TTCN Expression of output action test

Consider the transition $s \xrightarrow[Rt]{!A,R} s'$ with an INCONCLUSIVE interval R'. The execution tree has one branch $s \xrightarrow[Rt]{recv(A),PASS(R),INCONCLUSIVE(R')} s'$. So, we obtain in TTCN:

Description	Verdict	Comment
?A		receive "A"
READTIMER X_1 T_1		
$\vdots$		
READTIMER X_N T_N		
$C_2(< T_1, ..., T_N >, R) \rightarrow$		
$(V1_{min}, ..., VN_{min})(V1_{max}, ..., VN_{max})$		
$C_2(< T_1, ..., T_N >, R') \rightarrow$		
$(V1'_{min}, ..., VN'_{min})(V1'_{max}, ..., VN'_{max})$		
$[V1_{min} \leq T_1$ AND $T_1 \leq V1_{max}]$	Pass	
$[V1'_{min} \leq T_1$ AND $T_1 \leq V1'_{max}]$	Inconclusive	
$[V1'_{min} > T_i$ AND $T_1 > V1'_{max}]$	Fail	"A" not received in R
$\vdots$		
$[VN_{min} \leq T_1$ AND $T_1 \leq VN_{max}]$	Pass	
$[VN'_{min} \leq T_1$ AND $T_1 \leq VN'_{max}]$	Inconclusive	
$[VN'_{min} > T_i$ AND $T_1 > VN'_{max}]$	Fail	"A" not received in R
?otherwise	Fail	bad symbol
$\vdots$		
CANCEL X_i		reset x_i if necessary
START X_i		
$\vdots$		

5 A faulty implementation

In order to illustrate the detection of potential faults in an faulty implementation, let us consider the implementation in figure 6. This one contains two faults: the first fault concerns the outgoing transition of the state TMP3 which is is labeled with "!O9" whereas it should be "!O6" in the PASS interval R'_4 . The second one, is a violation of timing constraints on input action "?I2". The time during which "?I2" could be executed is reduced because the constraint should be $x < 1$. During the test execution, the tester sends inputs symbols and receives output ones of the test case, illustrated in figure **??**.

First, according to the execution tree (figure 4), it sends "?I2" for a clock valuation equals to $(0, 0)$ (see execution tree), and it sends "?I7?I11" for others clock valuations. The only reaction we can observe from the I.U.T is "!O9", and this is an unexpected output. So, the implementation is faulty.

Then the tester sends "?I2" for a clock valuation equals to $(1 - \epsilon, 1 - \epsilon)$. The I.U.T stays in the state IDLE because its timing constraints are not satisfied. As there is none output reaction, the tester continues to send "?I7?I11" for others clock valuations. None output reaction can be received because the I.U.T is still in IDLE, whereas "!O6" is excepted in the PASS interval R'_4. So, the implementation is faulty.

Consequently, the two faults are detected after executing the whole test case with

different clock valuations. However, it is impossible to detect where the implementation is faulty. A diagnostic is necessary.

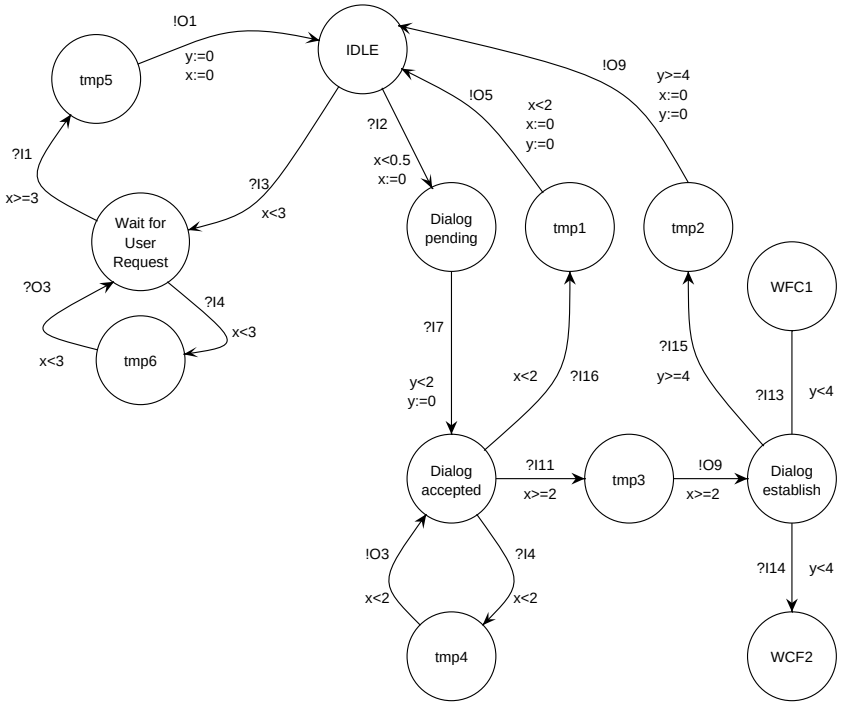

Figure 6: A faulty implementation

6 Conclusion

References

[ACH+92] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transition systems. In R. Cleaveland, editor, *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science*, pages 340–354. Springer-Verlag, 1992.

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[DY96] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proceedings of the 1996 IEEE Real-Time Systems Symposium, RTSS'96,* Washington DC, USA. IEEE Computer Society Press, 1996.

[EnFDE97] A. En-nouaary, H. Fouchal, R. Dssouli, and A. Elqortobi. Test derivation for timed systems. Report LERI-97-09-01, LERI-RS (Université de Reims), 1997.

[ISO91] ISO. Conformance Testing Methodology and Framework. International Standard 9646, International Organization for Standardization — Information Technology — Open Systems Interconnection, Genève, 1991.

[PF99] Eric Petitjean and Hacène Fouchal. A fault model for timed system testing. In *Research report, submitted to TCS 2000*, 1999.

[SSb00] Hacene Fouchal Sebastien Salva and Simon bloch. Metrics for Timed Systems Testing. In *?th OPODIS International Conference on Distribued Systems, Paris*, 2000.

[YL93] M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems (Extended abstract). In C. Courcoubetis, editor, *Proceedings of the 5th International Conference on Computer Aided Verification,* Elounda, Greece, volume 697 of *Lecture Notes in Computer Science*, pages 210–224. Springer-Verlag, 1993.