# From the feasibility analysis to real-time test generation

Ismail Berrada, Richard Castanet, Patrick Felix

# FROM THE FEASIBILITY ANALYSIS TO REAL-TIME TEST GENERATION

ISMAIL BERRADA, RICHARD CASTANET AND PATRICK FÉLIX

**Abstract.** Testing real-time systems is an important and not obvious step in the validation process of critical systems. This paper proposes an efficient $o((n \times k)^2)$ algorithm for solving the feasibility problem corresponding to a path of length $n$ in a timed automaton, with $k$ clocks. The given solution, combined with synchronous product, allows in particular, generating both the fastest and the slowest timed test cases for a given test purpose.

## 1. Introduction

The behavior of a real-time system is highly dependent on the temporal performances of target hardware platforms used for the implementation. These temporal constraints should be taken into account at the earliest stage of development process. But, we should notice the increasing implementations complexity, due to their algorithms and time constraints. Since systems must be delivered faultless, one way to increase their security is through conformance testing.

### 1.1. Conformance Testing

A general definition of conformance is to compare the implementation behavior with the specification one, in order to find errors. This verification consists in the definition of some test sets, their submission to the system under test (IUT), and the observation of their executions with a verdict (oracle) based on the specification, whether they are successful or not. The definition of the test set can be based on the specification (black-box), on the structure of the implementation (glass-box), or both. Usually, the development of conformance testing procedures is mostly based on the designers experience. A potential improvement that is being examined is to make testing a *formal method* and to provide tools that automate test cases generation.

### 1.2. Test Based Approach

The most known approaches to test selection in the black-box conformance testing framework are:

Coverage Criteria. The most used coverage criteria are based on the specification model. In the case of timed automata, some authors proposed transition, state or variable coverage approaches.

Test Assumptions. Various test assumptions can be formulated and combined until a finite test set is obtained. In the case of input/output timed automata, some authors assume that the specification and the IUT are deterministic, output enabled, and deadlock and livelock free.

Test Purposes. The user chooses behaviors which have to be analyzed, hence the specification exploration reduction. Using the specification and test purposes, a finite number of test cases are generated.

### 1.3. Our Contribution

Generating a test case for a path assumes that we are already able to decide the reachability of some states for this path. In this paper, we consider the following feasibility problem: *if a given path of the automaton has executions, we wish to determinate which constraints, every execution of this path has to verify*. Our motivation for studying this problem is test cases generation: once the feasibility conditions over a path are calculated, they allow in particular, generating both the fastest and the slowest timed executions. While these executions are not unique, we will identify two versions for each fastest and slowest timed executions. The contributions of this paper are: a $o((n \times k)^2)$ time solution for the feasibility problem, generation of the slowest and the fastest timed executions for a path, introduction and formalization of the strong/weak feasibility concept, and an application to test cases generation.

### 1.4. Related Work

An extension of test theory for Mealy machines in the case of dense real-time systems was proposed by Springintveld et al. [18]. The authors suggested to preform a kind of discretization of the region graph model. The discretization step size takes into account the number of clocks as well as the timing constraints. From the generated model, they derive test cases. This extension yields to a finite and complete set of test but the method is highly exponential and is not usable in practice. Another result generating test sequences for a discretized deterministic timed automaton is given by En-Nouaary et al. in [9]. The authors propose to build a grid automaton from the region graph, and

use a Wp method for the generation assuring a good coverage of the initial specification, but the number of generated test cases can be large. In [14], the authors have chosen as model temporized automata with discrete time. The model is transformed into automaton without time, but with two special events on clocks: set and expire. The advantage of the method is combinatorial explosion limitation. A drawback of this approach is the possibility of generating some test cases that contain non-executable sequences with the events set and expire.

In [7], an implicit clock is used, the time is discrete and the proposed model is a temporized transition system. The test concerns only time domains(not events coming at a precise time). In [8], the system specification is based on a constraint graph. From a fault model, the authors define test criteria and generate test cases according to the test criteria. Since constraint graph is used as a model, only delays can be expressed between two successive events, and the coverage of faults cannot be complete. In [16], the generation of test cases is produced from logic formula (time is expressed by using two constructors: future and past). A unique clock is used and the temporal domain is discrete. This model is not sufficient to represent complex real-time systems and the generated test suite covers only integer values of time.

We also mention another interesting contribution [12] that proposes a generation method based on must/may traceability. The authors propose to test first, the correctness of the implementation of states and transitions. For that, they transform the specification into a FSM, and use the UIOv-method to derive test cases. For the correctness of the implementation of transition conditions, they assume that faults are restricted to some typical ones and use the must/may traceability method to generate test cases.

In [6, 10, 17], the authors propose first to enlarge the regions of the graph and to build dynamically the region graph according to test purposes. Our approach [13] uses timed automata model and a synchronous product between the specification and a test purpose as in [15]. A reachable path is dynamically calculated by resolving inequalities on time constraints. This method avoids combinatorial explosion of the region graph, but don't give a complete coverage of faults.

The remainder of this paper is structured as follows: After the introduction of timed automata with global clock (TAGC) model, and formalization of the feasibility problem, chapter 2 and 3, we will present, in chapter 4, a solution for reached values computation of the global clock. The identification of both, the fastest/slowest timed execution, and strong/weak feasibility is done in chapter 5. In chapter 6, the reached values computation algorithm for any clock is exposed. An application, of the results proposed in this paper, to test

cases generation is presented in chapter 7. The last chapter summarizes the contributions of the paper and presents future works.

## 2. Timed Automaton Modeling

Timed automata [2] were introduced as a formal notation to model the behavior of real-time systems. This section presents the timed automaton model and some basic definitions in order to specify notations.

### 2.1. Preliminaries

Let $\mathbb{R}$ denote the set of reals, $\mathbb{R}^+$ the set of nonnegative reals, and $\overline{\mathbb{R}^+}$ the set of nonnegative reals with the single element $\infty$. We extend the standard partial ordering $\leq$, the addition operator $+$, and the subtraction operator $-$ over $\mathbb{R}^+$ to $\overline{\mathbb{R}^+}$ in the usual way: for every $t \in \overline{\mathbb{R}^+}$, $t \leq \infty$, $t + \infty = \infty + t = \infty$ and $\infty - t = \infty$. An interval $I$ of $\overline{\mathbb{R}^+}$ is a dense set over $\overline{\mathbb{R}^+}$ denoted by $[a, b]$ where $(a, b) \in \overline{\mathbb{R}^+}^2$. $a$ (resp. $b$) is the lower (resp. upper) bound of $I$: we write $inf(I) = a$ and $sup(I) = b$. $\overline{I}$ denotes the complementary of $I$ in $\overline{\mathbb{R}^+}$. For $(c, d) \in \overline{\mathbb{R}}^2$, $min(c, d)$ (resp. $max(c, d)$) is the smallest (resp. greatest) of $c$ and $d$. For $E = (E_{ij})_{i\in I,\, j\in J}$, $E^{'} = (E^{'}_{ij})_{i\in I,\, j\in J}$ and $\tau \in \mathbb{R}^+$: $\tau \times E = (\tau \times E_{ij})_{i\in I,\, j\in J}$, and $E + E^{'} = (E_{ij} + E^{'}_{ij})_{i\in I,\, j\in J}$. For a finite set $Q$, the $Card(Q)$ is the number of elements of $Q$.

#### Clocks, Constraints and Interpretations.

A clock is a variable that allows to record the passage of time. It can be set to a certain value and inspected at any moment to see how much time has passed. In the Alur-Dill model [2], clocks increase at the same rate, they are ranged over $\mathbb{R}^+$ and the only assignments that are allowed are clocks reset of the form $x := 0$. For a set $C$ of clocks, the set $\Phi(C)$ of clocks constraints $\phi$ is defined by the grammar:

$$\phi := x \leq c \mid c \leq x \mid \phi_1 \wedge \phi_2$$

where $x$ is a clock in $C$ and $c$ is a constant in $\mathbb{Q}$. A clock interpretation $\nu$ for a set $C$ of clocks is a mapping from $C$ to $\mathbb{R}^+$ that assigns a real value to each clock.

### 2.2. Timed Automaton

**Definition 1** (Timed Automaton)**.** *A timed automaton (TA) A is a tuple*

$(S, L, C, S_{init}, T)$ *where,*

- $S$ *is a finite set of locations,*
- $S_{init}$ *is an initial location,*
- $L$ *is a finite set of labels,*
- $C$ *is a finite set of clocks,*
- $T \subseteq S \times L \times 2^C \times \Phi(C) \times S$ *is a set of transitions.* $t_i = (s, a, \phi, \lambda, s^{'})$ *represents an edge from location s to location* $s^{'}$ *on symbol a.* $\phi$ *is a clock constraint over C that specifies when the transition is enabled, and the set* $\lambda \subseteq C$ *gives the clocks to be reset with this transition. For a transition* $t_i$ *and a clock x,* $C(t_i)$ *denotes the set of clocks that belong to* $t_i$*, and* $SC(t_i, x)$ *the specification constraint over x:* $\phi(x)$.

**Definition 2** (TA with Global Clock)**.** *A timed automaton with global clock (TAGC) A is:*

- *A timed automaton.*
- *There is a clock h of A without reset in any transition except for a transition which leads to the initial state.*

The notion of the clock *h* is implicit in the Alur-Dill model and explicit in the P-automaton [5]. The choice of TAGC model is justified by the context of black-box testing in which all events are seen according to a global clock, and by comprehension and clarity reasons.

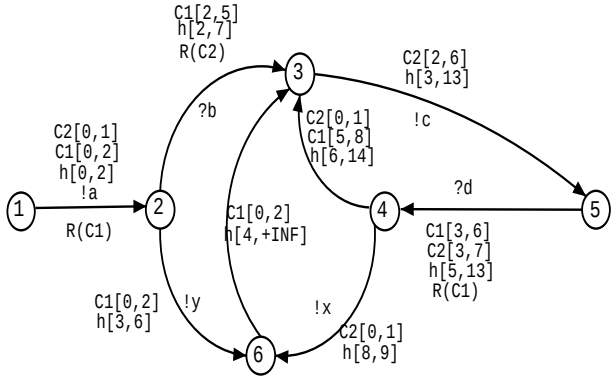


Figure 2.1: Timed automaton *A* with global clock *h*.

Let *A* be the automaton with global clock *h* of Fig. 2.1. This automaton has 6 states, 8 transitions and 3 clocks. *R(C1)* means that clock *C1* is reset in this transition. Each transition has some clock's constraints and events. For

this automaton, $C = \{C1, C2, h\}$, $C(t_{1->2}) = \{h, C1, C2\}$, $C(t_{3->5}) = \{h, C2\}$, $SC(t_{1->2}, C1) = [0, 2]$, and $SC(t_{3->5}, C2) = [2, 6]$.

**Definition 3** (Timed Execution)**.** *Let A be a TA. A path $\rho = t_1..t_n$ of A is a suite of transitions that form an arc in the graph associated with A. A timed execution (TE) $E$ of $\rho$ is a two-dimensional real array of size $n \times Card(C(\rho))$[1] that satisfies:*

- *Rule 1: $\forall x \in C, \forall i \in [1, n]$: $E(t_i, x)$ is the valuation of x in $t_i$.*
- *Rule 2: $\forall i \in [1, n]$, $E(t_i, x) \in SC(t_i, x)$.*
- *Rule 3: $\forall i \in [1, n-1]$, $E(t_i, x) \leq E(t_{i+1}, x)$ if x is not reset in $t_i$.*
- *Rule 4: For every clock y in $C$:*
  - *If x and y are not reset in the transition $t_i$, then $E(t_{i+1}, x) - E(t_i, x) = E(t_{i+1}, y) - E(t_i, y)$.*
  - *If y alone is reset in $t_i$, then $E(t_{i+1}, x) - E(t_i, x) = E(t_{i+1}, y)$.*
  - *If x alone is reset in $t_i$, then $E(t_{i+1}, x) = E(t_{i+1}, y) - E(t_i, y)$.*
  - *If both x and y are reset in $t_i$, then $E(t_{i+1}, x) = E(t_{i+1}, y)$.*

While all clocks increase at the same rate, these four rules ensure this hypothesis. From now on, we assimilate execution and timed execution.

**Notation 2.1.** *For a path $\rho = t_1.t_2...t_n$ of a TAGC A, $\rho_i = t_1.t_2...t_i$ is a subpath of $\rho$. $ES(\rho)$ is the set of all executions of $\rho$. If for every $i$ in $[1, n]$, $a_i$ is the event associated to $t_i$, then: $(d_1, a_1).(d_2, a_2)...(d_n, a_n)$, such that $d_i \in \overline{\mathbb{R}}$, is the execution $E$ defined by:*

- *For every $i$ in $[1, n]$, $E(t_i, h) = d_i$.*
- *For every $i$ in $[1, n]$, for every clock $x$ in $C(t_i)$ last reset in $t_k$, $E(t_i, x) = d_i - d_k$.*

In the reset of the paper, we assume that $\rho = t_1..t_n$ is a path of a given TAGC *A* from the initial state, and $n$ is the last state of this path.

## 3. Feasibility of a Path

In the feasibility problem, we wish to determine the reachability of transition $t_n$. That is, we are interested in the computation of suitable constraints over $\rho$ for firing $t_n$. Section 3.1 formalizes this problem by introducing the reached values concept and Section 3.2 introduces well-formed (WF) suites, out reach intervals and some properties of these intervals.

[1] $C(\rho)$ is the set of clocks that appear in $\rho$.

### 3.1. Reached Values of a Constraint in a Transition

**Definition 4.** *Let $RV(t_i,x)_\rho$ denote a subset of $SC(t_i,x)$. $RV(t_i,x)_\rho$ is the reached values set of the clock $x$ in transition $t_i$ if:*

1. *For every $\tau \in RV(t_i,x)_\rho$, there is an execution $E \in ES(\rho)$ such that: $E(t_i,x) = \tau$.*
2. *For every execution $E \in ES(\rho)$, $E(t_i,x)$ is in $RV(t_i,x)_\rho$.*

The reached values of a constraint in a given transition depend on the chosen path. The feasibility problem consists then, in the reached values computation for every clock in every transition of this path. When there is no confusion on the path $\rho$, $RV(t_i,x)_\rho$ will be noted $RV(t_i,x)$.

**Corollary 1.** *The state $n$ is a reachable state of $\rho$ iff for every $i$ in $[1,n]$, $RV(t_i,h)$ is a non empty set. In this case, we say that $\rho$ is a feasible path.*

**Lemma 2.** *For every $i$ in $[1,n]$, for every $x$ in $C(t_i)$, $RV(t_i,x)$ is an interval over $\overline{\mathbb{R}^+}$.*

*Proof.* From the definition of an execution and the grammar constraints (a constraint over a clock is a dense set), it is easy to see that from two executions of $\rho$, $E_1$ and $E_2$, and a given $\delta \in [0,1]$, the execution $E$ defined by:

$$\forall i \in [1,n],\ \forall x \in C(t_i),\ E(t_i,x) = \delta \times E_1(t_i,x) + (1-\delta) \times E_2(t_i,x)$$

is an execution of $\rho$. Then $\forall i \in [1,n]$, $\forall x \in C(t_i)$, $RV(t_i,x)$ is a dense set over $\overline{\mathbb{R}^+}$. □

**Remark 3.1.** *If $\rho'$ is a path of a TA where the clock h is implicit, and $E$ is an execution of $\rho'$, then the clock $h$ values of $E$ in every transition can be computed. In fact, $\forall i \in [1,n]$, $\forall x \in C(t_i)$: $E(t_i,h) = \sum^{i}_{j/Reset(t_j,x)} E(t_j,x)$, where $Reset(t_j,x)$ means that the clock x is last reset in transition $t_j$ with $j < i$. Now, if for two executions of $\rho'$, $E_1$ and $E_2$, there is $\delta \in [0,1]$ such that $\forall i \in [1,n]$, $\forall x \neq h \in C(t_i)$, $E(t_i,x) = \delta \times E_1(t_i,x) + (1-\delta) \times E_2(t_i,x)$, then: $E(t_i,h) = \sum^{i}_{j/Reset(t_j,x)} (\delta \times E_1(t_j,x) + (1-\delta) \times E_2(t_j,x)) \Rightarrow E(t_i,h) = \delta \times E_1(t_i,h) + (1-\delta) \times E_2(t_i,h)$. Conclusion, $RV(t_i,h)$ is also a dense set over $\overline{\mathbb{R}^+}$ if h is implicit.*

### 3.2. Out Reach Intervals

**Definition 5.** *Let $I = [a,b]$ and $J = [c,d]$ be two intervals of $\overline{\mathbb{R}^+}$. The sum $I \oplus J$ of $I$ and $J$ is the interval defined by:$I \oplus J = [a+c, b+d]$.*

**Definition 6** (Out Reach Intervals)**.** *The out reach interval of a transition is the time interval outside of which the transition must happen*[2]*, in order to be in agreement with $\rho$ time constraints. If a transition happens in this interval, we know that at least one constraint of $\rho$ is violated. Formally we define:*

$h(t_0) = \overline{[0,0]} = ]0, \infty[$.

$h(t_i) = \bigcap_{x \in C(t_i)} \overline{h}(t_{k_x}) \oplus SC(t_i, x)$

*where x is last reset in $t_{k_x}$ ($k_x < i$), and $\overline{h}(t_i)$ the complementary of $h(t_i)$.*

**Example 3.1.** *Let $\rho$ (Fig. 3.2) be the path extracted from the automaton of Fig. 2.1.*

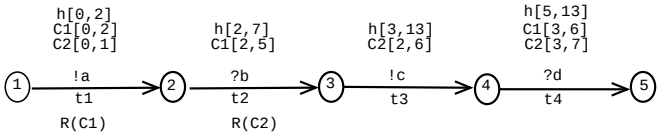


Figure 3.2: Out reach intervals.

*First, we add a new transition $t_0$ in which all clocks are reset: $\overline{h}(t_0) = [0,0]$.*
*For $t_1$, all clocks are reset in $t_0$, then:*
$\overline{h}(t_1) = (\overline{h}(t_0) \oplus [0,2]) \cap (\overline{h}(t_0) \oplus [0,2]) \cap (\overline{h}(t_0) \oplus [0,1]) = [0,1]$.
*For $t_2$, C1 is reset in $t_1$, then:*
$\overline{h}(t_2) = (\overline{h}(t_1) \oplus [2,5]) \cap (\overline{h}(t_0) \oplus [2,7]) = [2,6]$.
*For $t_3$, C2 is reset in $t_2$, then:*
$\overline{h}(t_3) = (\overline{h}(t_2) \oplus [2,6]) \cap (\overline{h}(t_0) \oplus [3,13]) = [4,12]$.
*At the end, in $t_4$, C2 is reset in $t_2$ and C1 in $t_1$ then:*
$\overline{h}(t_4) = (\overline{h}(t_1) \oplus [3,6]) \cap (\overline{h}(t_2) \oplus [3,7]) \cap (\overline{h}(t_0) \oplus [5,13]) = [5,7]$.

**Definition 7** (Well-formed Suite)**.** *Let $(I_i)_{i \in [1,n]}$ be a suite of non empty intervals over $\overline{\mathbb{R}}$, $I_i = [inf(I_i), sup(I_i)]$. $(I_i)_{i \in [1,n]}$ is a well-formed (WF) suite if for every $i$ in $[1, n-1]$, $inf(I_i) \leq inf(I_{i+1})$, $sup(I_i) \leq sup(I_{i+1})$.*

**Definition 8.** *$(I_i)_{i \in [1,n]}$ accepts a WF sub-suite $(J_i)_{i \in [1,n]}$ if for every $i$ in $[1,n]$, $J_i \subset I_i$, and $(J_i)_{i \in [1,n]}$ is WF.*

The next lemma gives a relation between $(RV(t_i, h))_{i \in [1,n]}$ intervals, consequence of path clocks reset.

**Lemma 3.** *$\rho$ is feasible iff there is a suite of intervals $(I_i)_{i \in [1,n]}$ such that:*

1. *$(I_i)_{i \in [1,n]}$ is a WF suite.*
2. *For every $i$ in $[1,n]$, $I_i \subset \overline{h}(t_i)$.*

[2] According to the clock *h*.

3. *For every $i$ in $[1,n]$, for every $x$ in $C(t_i)$ being last reset in $t_j$, $j<i$:*
   - *For every $\tau_j$ in $I_j$, there is $\tau_i$ in $I_i$ such that $\tau_i - \tau_j \in SC(t_i,x)$.*
   - *For every $\tau_i$ in $I_i$, there is $\tau_j$ in $I_j$ such that $\tau_i - \tau_j \in SC(t_i,x)$.*

*Moreover, if for every $i$ in $[1,n]$, $I_i \subset \overline{h}(t_i)$ is the largest interval verifying the third condition, then $(I_i)_{i\in[1,n]}$ is the wanted suite $(RV(t_i,h))_{i\in[1,n]}$.*

*Proof.* $\Rightarrow$: From the definition of an execution, reached values and the definition 7, the suite $(RV(t_i,h))_{i\in[1,n]}$ verifies the theorem.

$\Leftarrow$: If there is a suite $(I_i)_{i\in[1,n]}$ that verifies the theorem, then the execution $E$ defined by: $\forall i \in [1,n], \forall x \in C(t_i)$ being last reset in $t_j$, $j<i$: $E(t_i,h) = sup(I_i),\ E(t_i,x) = sup(I_i) - sup(I_j)$, is an execution of $\rho$. Therefore $\rho$ is feasible.

For two suites $(I_i)_{i\in[1,n]}$ and $(J_i)_{i\in[1,n]}$ that verify the theorem, the suite $(K_i)_{i\in[1,n]}$ defined by: for every $i$ in $[1,n]$, $inf(K_i) = inf(inf(I_i), inf(J_i))$, $sup(K_i) = sup(sup(I_i), sup(J_i))$, verifies also the theorem. Then the largest suite that verifies 3) is the suite $(RV(t_i,h))_{i\in[1,n]}$ □

Corollary 4 gives a necessary condition for the feasibility of the path, consequence of lemma 3.

**Corollary 4.** *If $\rho$ is feasible then the suite of the complementary of out reach intervals of $\rho$ has a WF sub-suite.*

## 4. Computation of the Reached Values of Clock h

In the preceding section, we have introduced the reached values of a constraint. We showed that path feasibility depends on the reached values of the global clock. Section 4.1 presents the constraints graph associated to a two variables inequalities system, Section 4.2 develops an algorithm for the computation of the reached values of the clock $h$, and Section 4.3 discusses the complexity of this algorithm.

### 4.1. Constraints Graph.

For a two variables inequalities system, we associated a constraints matrix. For example, $M$ is the constraints matrix associated to system $S$.

$$S = \begin{cases} 3 \le x \le 6 \\ x - 3 \le y \\ y \ge 5 \end{cases} \qquad M = \begin{pmatrix} 0 & -3 & -5 \\ 6 & 0 & 3 \\ \infty & \infty & 0 \end{pmatrix}$$

A system of inequalities can be represented as a graph where every node is a variable and every edge is a constraint over these variables.

**Definition 9** (Constraints Graph.)**.** *Let $X$ be a set of variables and $\phi_x$ a conjunction of atomic terms of the form $x - x^{'} \leq c$, with $(x, x^{'}) \in X \cup \{0\}$ and $c \in \mathbb{Q}$. The constraints graph associated to $\phi_x$ is a valuated directed graph $G(\phi_x) = \{X \cup \{0\}, A\}$ defined by:*

- *$X \cup \{0\}$ is the graph vertexes set.*
- *$A \subset (\{X \cup \{0\} \times R \times (X \cup \{0\})\})$ is the graph edges set. We denoted by $x \xrightarrow{c} x^{'}$ every edge of $A$.*

$$x \xrightarrow{c} x^{'} \in A \Longleftrightarrow (x - x^{'} \leq c)\ is\ a\ \phi_x\ term$$

A two variables inequalities system admits a solution if its constraints graph does not contain a negative cycle.

## 4.2. Algorithm

Given a TAGC *A* and a path $\rho = t_1..t_n$ of *A* from an initial state, the computation algorithm of the suite $(RV(t_i, h))_{i\in[1,n]}$ is outlined in Fig. 4.3. The input of the algorithm is a path $\rho = t_1..t_n$, an empty system $S_\rho$ and a vector $\tau$ of $n$ reals. The algorithm consists in three steps: First, a checking step in which we compute $(\overline{h}(t_i))_{i\in[1,n]}$. If $(\overline{h}(t_i))_{i\in[1,n]}$ does not have a WF sub-suite then the path is not feasible (Corollary 4). Second, a system construction step in which the relations between clocks are formulated (Lemma 3). Finally, a solving step of the system constructed $S_\rho$.

## 4.3. Complexity Analysis: Modified Timestamps Generation Algorithm

The algorithm Fig. 4.3 introduces a linear system of inequalities. In [3], the authors give an efficient algorithm, of complexity $o(n \times k^2)$, for generating a possible sequence of time values at which the individual edges are traversed. The algorithm computes the shortest cost path $shortest_G(0, i)$ from node 0 to node $i$ in the digraph $G$ associated to the system $S_\rho$. This solution can be modified to solve the feasibility problem. In fact, the shortest cost path $shortest_G(0, i)$ in our case corresponds to $inf(RV(t_i, h))$: $inf(RV(t_i, h)) = -shortest_G(0, i)$. If now we consider the digraph $G^{'}$ obtained from $G$ by reversing edges, by applying the same algorithm to $G^{'}$, we have $shortest_{G'}(0, i) = shortest_G(i, 0) = sup(RV(t_i, h))$.

**Theorem 5.** *Given a path $\rho$ from the initial state of a TAGC $A$, with $n$ transitions and $k$ clocks, the computation of the reached values of the global clock $h$ can be solved in time $o(n \times k^2)$.*

**Input:** A timed path $\rho$.
**Output:** The reached values of clock $h$.
**Data Structure:** A system $S_\rho$, and a vector $\tau$ of $n$ variables $\tau_i$.
**Begin**
Phase One: Checking the feasibility of $\rho$.
/* $S_\rho$ is empty at the beginning*/
/* Checking if$(\overline{h}(t_i))_{i\in[1,n]}$ has a WF suite * /
Compute the suite $(\overline{h}(t_i))_{i\in[1,n]}$;
If $(\overline{h}(t_i))_{i\in[1,n]}$ does not have a WF sub suite Then **exit**;
For $i := 1$ To $n$ Do
$SC(t_i, h) := \overline{h}(t_i)$;
End For
Phase Two: Filling $S_\rho$.
/* Adding relation inequalities (Lemma 3) */
For $i := n$ To 1 Do
If $i \neq 1$ Add $\tau_{i-1} \leq \tau_i$ To $S_\rho$;
Add $inf(SC(t_i, h)) \leq \tau_i \leq sup(SC(t_i, h))$ To $S_\rho$;
For $x \in C(t_i)$ Add To $S_\rho$
/* $x$ was last reset in $t_j$ */
$inf(SC(t_i, x)) \leq \tau_i - \tau_j \leq sup(SC(t_i, x))$;
End For
End For
/* Solving step */
Phase Three: Solve $S_\rho$ .
Solve $S_\rho$;
**End;**

Figure 4.3: $(RV(t_i, h))_{i\in[1,n]}$ computation algorithm.

## 5. Fastest and Slowest Accumulate Delay Executions

Generating the fastest and the slowest accumulate delay executions are interesting for several reasons. Firstly, they allow testing the implementation behavior in critical situations. Secondly, it is generally useful that regression testing can be executed as quickly as possible in order to improve the turn around time between changes [11]. Since these executions are not unique, in this section, we will identify two versions for each fastest and slowest timed execution: the fastest and the slowest timed executions according to every transition crossed (Section 5.1), and the fastest and the slowest timed executions according to every clock of a transition (Section 5.3). Section 5.2 introduces and formalizes the strong/weak feasibility concept.

Recall. For a path $\rho = t_1.t_2...t_n$ of a TAGC $A$, $\rho_i = t_1...t_i$ is a sub-path of $\rho$. $ES(\rho)$ is the set of all executions of $\rho$.

### 5.1. Fastest and Slowest Timed Executions According to every Transition Crossed

One way to reach a state of a path in an early (resp. late) time is to reach every state between this one and the initial state as soon (resp. late) as possible. This observation is formulated below.

**Lemma 6.** *Let us assume that $\rho$ is a feasible path.*

1. *If for every $i$ in $[1,n]$, $RV(t_i,h)$ is bounded, then there is a single execution $E_S \in ES(\rho)$ such that: for every $i$ in $[1,n]$, $E_S(t_i,h) = sup(RV(t_i,h))$.*
2. *There is a single execution $E_F$ such that: for every $i$ in $[1,n]$, $E_F(t_i,h) = inf(RV(t_i,h))$.*

*Proof.* See Appendix A. □

### 5.2. Weak and Strong Feasibility

**Problem 5.1.** *The problem treated here is: under what constraints over a feasible path, if a transition is fired, the next transition can be also fired until the last transition of this path.*

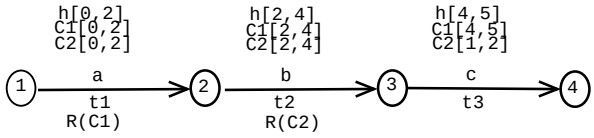


Figure 5.4: A weakly feasible path.

Let us take the example of Fig. 5.4. If the transition $t1$ is fired at moment 2, and $t2$ at 4, we notice that $t3$ can not be then fired according to its path constraints. Now, let us take the example of Fig. 5.5. For this path, each time a transition is fired respecting its constraints, the next transition can be fired.

**Definition 10.** *The feasible path $\rho = t_1.t_2...t_n$ is said to be strongly feasible if for every natural $i \in [1,n]$, and for every execution $E' \in ES(\rho_i)$, there is an execution $E \in ES(\rho)$ that meets $E'$. On the opposite (i.e. there is $E' \in ES(\rho_i)$, such that for every $E \in ES(\rho)$, $E$ does not meet $E'$), $\rho$ is said to be weakly feasible.*

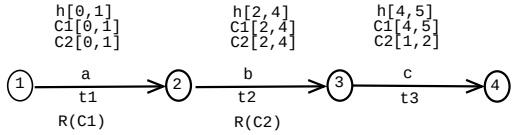


Figure 5.5: A strongly feasible path.

Typically, $\rho$ is strongly feasible if: for every $i$ in $[1, n]$, for every execution $E^{'}$ in $ES(\rho_i)$, there is $E$ in $ES(\rho)$ such that: for every $j$ in $[1, i]$, for every $x$ in $C(t_j)$, $E(t_j, x) = E^{'}(t_j, x)$.

The strong feasibility allows deciding at advance if the running behavior of the system will reach the needed target behavior. The latter is capital during test applications on real implementations. The tester will be then able to decide stopping, at each time, the implementation execution, and not to await the end of the test case. Lemma 7 gives a necessary and a sufficient condition for the strong feasibility.

**Lemma 7.** *Let $\rho = t_1.t_2...t_n$ be a feasible path. $\rho$ is strongly feasible iff for every $i$ in $[1, n]$, for every clock $x$ in $C(t_i)$, $t_i$ is fired in $RV(t_i, x)_\rho$.*

The idea of the proof is to replace the specification constraints by the reached values and to proceed by induction.

*Proof.* See Appendix B. □

## 5.3. Fastest and Slowest Executions According to every Clock of a Transition

As the fastest and slowest executions are not unique, in this subsection, we will present other executions that also allow computation of the fastest and slowest delay for reaching a path target state.

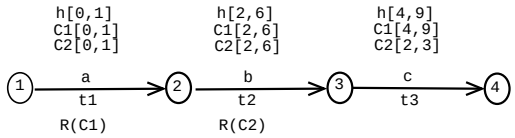


Figure 5.6: Temporal path example.

Let be the path of Fig. 5.6. We can verify that, $\forall i \in [1,3]$, $\forall x \in C(t_i)$, $RV(t_i,x) = SC(t_i,x)$. The slowest execution according to every transition crossed $E_S$[3] corresponding to this path is $(1,a).(6,b).(9,c)$. For this execution, $E_S(t_3,C_1) = 8$, $E_S(t_3,C_2) = 3$, and $E_S(t_3,h) = 9$. Let us now consider $E$, the execution given by $(0,a).(6,b).(9,c)$. This execution is also a slowest execution, and we have $E(t_3,C_1) = 9$, $E(t_3,C_2) = 3$, and $E(t_3,h) = 9$. As we can see, $E$ takes simultaneously the $sup(RV(t_3,C_1))$, $sup(RV(t_3,C_2))$ and $sup(RV(t_3,h))$. Next, we will prove that this result holds for every path in every transition.

**Lemma 8.** *Let us assume that $\rho$ is feasible. If $Card(C(t_n)) = m$:* [4]

1. *If for every $i$ in $[1,n]$, $RV(t_i,h)_\rho$ is bounded. Then there is an execution that takes simultaneously the $sup(RV(t_n,x)_\rho)$, for every $x$ in $C(t_n)$.*
2. *There is an execution that takes simultaneously the $inf(RV(t_n,x)_\rho)$, for every $x$ in $C(t_n)$.*

The proof of the lemma is done by induction over the number of transition clocks, using the constraints graph associated to a path.

*Proof.* See Appendix C. □

We have showed that there is an execution that takes simultaneously the $(inf(RV(t_n,x)))_{x\in C(t_n)}$ (resp. $(sup(RV(t_n,x)))_{x\in C(t_n)}$), for transition $t_n$. In fact, this result holds for every transition of $\rho$.

**Lemma 9.** *Let us assume that $\rho$ is feasible. For all $j$ in $[1,n]$ we have :*

1. *If for every $i$ in $[1,j]$, $RV(t_i,h)_\rho$ is bounded, then there is an execution of $\rho$ that takes simultaneously the $sup(RV(t_j,x)_\rho)$, for every $x$ in $C(t_j)$.*
2. *There is an execution of $\rho$ that takes simultaneously the $inf(RV(t_j,x)_\rho)$, for every $x$ in $C(t_j)$.*

*Proof.* Let us consider the path $\rho_j$, such that: for every $k$ in $[1,j]$, and for every $x$ in $C(t_k)$, $SC(t_k,x) := RV(t_k,x)_\rho$. According to lemma 8, for $\rho_j$, there is an execution $E \in ES(\rho_j)$ that takes simultaneously the $sup(RV(t_j,x)_\rho)$, for every clock $x$ in $C(t_j)$. Now, according to lemma 7, this execution can be extended to an execution of $\rho$ due to the strong feasibility of $\rho$ (every transition of $\rho_j$ is fired according to the reached values sets of $\rho$). □

We have proved the existence of an execution $E$ (resp. $E'$) of $\rho$ that takes simultaneously the maximal (resp. minimal) values of $RV(t_j,x)$, for every $j$ in $[1,n]$. Let $E_1$ (resp. $E_2$) be an execution that meets $E$ (resp. $E'$), for

[3] See lemma 6.

[4] m less than $Card(C(\rho))$.

every $i \leq j$, and fires as soon (resp. late) as possible transitions after $t_j$ (the existence of $E_1$ and $E_2$ are ensured by the strong feasibility of $\rho$). For these executions, the delay between the instant of firing $t_j$ and the instant of firing every transition $t_i$, $j \leq i$, is less (resp. more) than the delay between $t_j$ and $t_i$ for any other execution of this path. This result is proved in the next lemma.

**Lemma 10.** *Let us assume that $\rho$ is feasible. Let $E_1 \in ES(\rho)$ (resp. $E_2 \in ES(\rho)$) be the execution that takes simultaneously the maximal (resp. the minimal) values of $RV(t_j, x)$, for a $j$ in $[1, n]$, and for every $x$ in $C(t_j)$, while making the possible minimum (resp. maximum) delay between transitions after $t_j$. Then, for every execution $E \in ES(\rho)$:*

1. *For every $i$, $j \leq i$, $E_1(t_i, h) - E_1(t_j, h) \leq E(t_i, h) - E(t_j, h)$.*
2. *For every $i$, $j \leq i$, $E(t_i, h) - E(t_j, h) \leq E_2(t_i, h) - E_2(t_j, h)$.*

*Proof.* See Appendix D. □

## 6. Computation of the Reached Values of any Clock

In this section we will introduce a computation method of the reached values of any clock by using the two versions of each fastest and slowest timed executions.

### Algorithm:

We assume that used clocks are $h$, $x_1$,$x_2$,..., $x_{Card(C(\rho))}$. The algorithm for computing the reached values for every clock is outlined in Fig. 6.7. At the beginning of the algorithm, $H_m[k][j]$ contains the lower bound of the specification constraint over $x$ (i.e. $inf(SC(t_k, x_j))$), and $H_M[k][j]$ the upper bound of the specification constraint over $x$ (i.e. $sup(SC(t_k, x_j))$). In the first phase, the algorithm computes the reached values of the global clock. For every clock $x$ in a transition $t_i$, last reset in the initial state, its reached values are the one of the clock $h$ (i.e. $RV(t_i, x) := RV(t_i, h)$). $H_m$ and $H_M$ are then updated.

The second phase consists in a loop on transitions starting from $t_1$. For every transition, we check the emptiness of the set of associated clocks reset. If it is empty then we increase the current transition index $t_i$. In the opposite, we start computing the maximal and the minimal delay from the currant transition to any transition of the path. This is done by computing the reached values of $h$ for paths $\rho_1$ and $\rho_2$. We notice that $\rho_1$ and $\rho_2$ are defined such that $ES(\rho_1)$ and $ES(\rho_2)$ contain all executions that take simultaneously, respectively the maximal and minimal reached values over clocks of the current transition $t_i$ (Sets $ES(\rho_1)$ and $ES(\rho_2)$ are non empty according to lemma 9).

**Input**: A feasible timed path $\rho = t_1 \ldots t_n$ with $n$ states and $k$ clocks.
**Output**: Computation of reached values for every clock.
**Data Structure**: $H_m$ and $H_M$ a two dimensional arrays of size $n \times k$.
**Temporary Variables**: Two paths $\rho_1 = t_1 \ldots t_n$ and $\rho_2 = t_1 \ldots t_n$
**Begin**:
Phase One: Filling $H_m$ and $H_M$ with reached values of clocks last reset in the initial state.
Compute $(RV(t_i, h)_\rho)_{i \in [1,n]}$;
For $i := 1$ To $n$ Do
For every $x_k \in C(\rho)$, if $x_k$ is last reset in the initial state Then
$H_m[i][k] := inf(RV(t_i, h))$; $H_M[i][k] := sup(RV(t_i, h))$;
Phase Two: Filling $H_m$ and $H_M$ with the other reached values.
For $i := 1$ To $n$ Do
If there is clocks reset in $t_i$ Then
For $\rho_1$: $\forall x_k \in C(\rho)$, $SC(t_i, x_k) := [H_m[i][k], H_m[i][k]]$;
For $\rho_2$: $\forall x_k \in C(\rho)$, $SC(t_i, x_k) := [H_M[i][k], H_M[i][k]]$;
/* The feasibility of $\rho_1$ and $\rho_2$ are ensured by lemma 9.*/
Compute $(RV(t_i, h)_{\rho_1})_{i \in [1,n]}$;
Compute $(RV(t_i, h)_{\rho_2})_{i \in [1,n]}$;
For $j := i$ To $n$ Do
For every $x_k \in C(\rho)$ Do
/* Updating $H_m$ and $H_M$*/
If $x_k \in t_j$ is last reset in $t_i$ Then:
$H_m[j][k] := inf(RV(t_j, h)_{\rho_2}) - inf(RV(t_i, h)_{\rho_2})$;
$H_M[j][k] := sup(RV(t_j, h)_{\rho_1}) - sup(RV(t_i, h)_{\rho_1})$;
**End;**

Figure 6.7: Reached values computation algorithm.

Once the reached values of $h$ for $\rho_1$ and $\rho_2$ are computed, for every clock $x_k$ in $C(t_j)$, such that $j$ is greater than $i$, and $x_k$ last reset in the current transition $t_i$, then $inf(RV(t_j, x_k)_\rho) := inf(RV(t_j, h)_{\rho_2}) - inf(RV(t_i, h)_{\rho_2})$, and $sup(RV(t_j, x_k)_\rho) := sup(RV(t_j, h)_{\rho_2}) - sup(RV(t_i, h)_{\rho_2})$. This result is a consequence of lemma 10. At the end, $H_m$ and $H_M$ contain the minimum and the maximum reached values of any clock.

In every transition having clocks reset, the complexity of computing the reached values of this transition clocks is $o(n \times k^2)$, then the complexity of the algorithm is $o((n \times k)^2)$.

**Theorem 11.** *Given a path $\rho$ from the initial state of a TAGC $A$, with $n$ transitions and $k$ clocks, the computation of the reached values for every clock can be solved in time $o((n \times k)^2)$.*

## 7. Test Generation: Test Purpose Based Approach

Protocol testing consists in checking that some implementations conform to a given specification. We assume the determinism of the specification and the IUT. The latter are given as timed I/O automata. The architecture of the test is given in Fig. 7.8. The tester takes the place of the environment and controls the IUT by injecting test cases via control and observation points (COP).

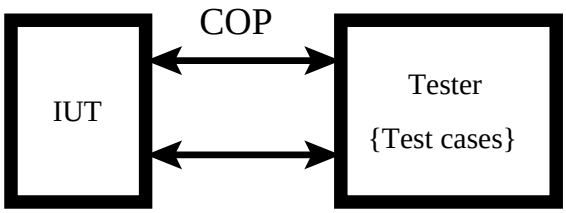


Figure 7.8: Test specification.

### 7.1. Test Purpose

A test purpose is a temporal property that one would like to observe in implementation behaviors. In the case of real-time systems, such property may include real-time features where time constraints appear explicitly. We also model test purpose with I/O automaton.

**Definition 11.** *A Test purpose $T_p$ is a deterministic, acyclic automaton with a distinguished non empty set of accepting states.*

### 7.2. Synchronous Product

**Definition 12.** *Let $Spec$ be a specification and $T_p$ a test purpose. The synchronous product of $Spec$ and $T_p$ is the I/O automaton $SP$ defined as follows.*

- $S_{init}(SP)) = (S_{init}(Spec), S_{init}(T_p))$.
- $S(SP) \subset S(Spec) \times S(T_p)$.
- $C(SP) = C(Spec) \cup C(T_p)$.
- $L(SP) = L(Spec) \cup L(T_p)$.

$S(SP)$ and $T(SP)$ are the smallest relations defined by the following rules:
(1)

$$\frac{(s_1, s_2) \in S(SP) \wedge (s_1, a, \phi_1, \lambda_1, s_1^{'}) \in T(Spec) \wedge (s_2, a, \phi_2, \lambda_2, s_2^{'}) \notin T(T_p)}{(s_1^{'}, s_2) \in S(SP) \wedge ((s_1, s_2), a, \phi_1, \lambda_1, (s_1^{'}, s_2)) \in T(SP)}$$

(2)
$$\frac{(s_1, s_2) \in S(SP) \wedge (s_1, a, \phi_1, \lambda_1, s_1^{'}) \in T(Spec) \wedge (s_2, a, \phi_2, \lambda_2, s_2^{'}) \in T(T_p)}{(s_1^{'}, s_2) \in S(SP) \wedge ((s_1, s_2), a, \phi_1 \wedge \phi_2, \lambda_1 \cup \lambda_2, (s_1^{'}, s_2)) \in T(SP)}$$

Using this synchronous product for a given test purpose $T_p$, we compute a set of paths from the specification that coincide in events with the test purpose. While not all of these paths respect the temporal constraints of the test purpose, we have to select the good ones.

### 7.3. Example

Let the automaton of Fig. 2.1 be an example of a specification, and let us consider the test purpose of Fig. 7.9.

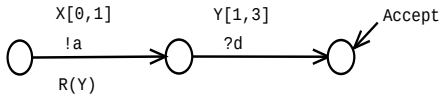


Figure 7.9: Test purpose 1.

In this test purpose, we want to test if after receiving an event "a" in a delay between 0 and 1 ($X$[0,1]), the IUT can send an event "d" in a delay between 1 and 3 ($Y$[1,3])[5]. By using the synchronous product, the path of Fig. 3.2 synchronizes with this purpose. By adding the constraints of test purpose to this path we have Fig. 7.10:

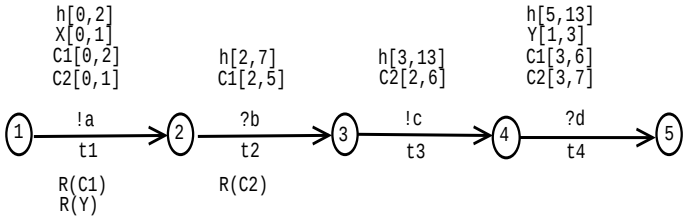


Figure 7.10: Synchronous Product 1.

- $\overline{h}(t_0) = [0, 0]$.
- $\overline{h}(t_1) = (\overline{h}(t_0) \oplus [0, 2]) \cap (\overline{h}(t_0) \oplus [0, 2]) \cap (\overline{h}(t_0) \oplus [0, 1]) \cap (\overline{h}(t_0) \oplus [0, 1]) = [0, 1]$.

[5] After reseting the clock $Y$ when receiving "a".

- $\overline{h}(t_2) = (\overline{h}(t_1) \oplus [2,5]) \cap (\overline{h}(t_0) \oplus [2,7]) = [2,6]$.
- $\overline{h}(t_3) = (\overline{h}(t_2) \oplus [2,6]) \cap (\overline{h}(t_0) \oplus [3,13]) = [4,12]$.
- $\overline{h}(t_4) = (\overline{h}(t_1) \oplus [3,6]) \cap (\overline{h}(t_2) \oplus [3,7]) \cap (\overline{h}(t_0) \oplus [5,13]) \cap (\overline{h}(t_1) \oplus [1,3]) = [5,3] = [,]$.

We notice that the out reach interval of transition $t_4$ is empty. We know immediately that, this path does not respect the test purpose.

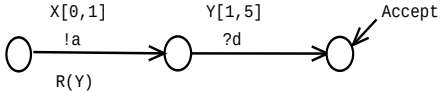


Figure 7.11: Test purpose 2.

Let us now consider the test purpose of Fig. 7.11. The suite of out reach intervals of the synchronous product of this path are: $\overline{h}(t_0) = [0,0]$, $\overline{h}(t_1) = [0,1]$, $\overline{h}(t_2) = [2,6]$, $\overline{h}(t_3) = [4,12]$ and $\overline{h}(t_4) = [5,6]$. The constraints system associated to this path after removing redundant constraints is:

$$S := \begin{cases} 0 \leq \tau_1 \leq 1 \quad 2 \leq \tau_2 \leq 6 \\ 4 \leq \tau_3 \leq 12 \quad 5 \leq \tau_4 \leq 6 \\ 2 \leq \tau_2 - \tau_1 \leq 5 \quad 2 \leq \tau_3 - \tau_2 \leq 6 \\ 3 \leq \tau_4 - \tau_1 \leq 5 \quad 3 \leq \tau_4 - \tau_2 \leq 7 \end{cases}$$

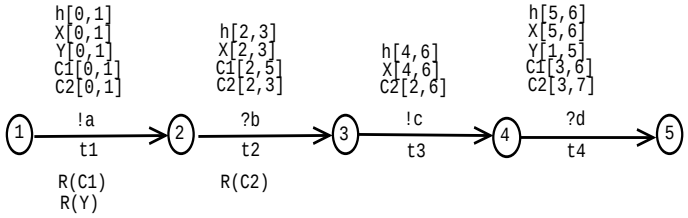


Figure 7.12: Phase one algorithm application.

The solution of this system is: $RV(t_1,h) = [0,1]$, $RV(t_2,h) = [2,3]$, $RV(t_3,h) = [4,6]$, and $RV(t_4,h) = [5,6]$. The reached values of clocks last reset in the initial state (Phase One) are given in Fig. 7.12.

For the computation of the maximum reached values of $C1$ and $Y$, we consider the path $\rho_1$ of Fig. 7.13. In this path, we have changed $SC(t_1,x)$ by $inf(RV(t_1,x))$ for every clock of $t_1$.

The reached values of $h$ for this path are: $RV_{\rho_1}(t_1,h) = [0,0]$, $RV_{\rho_1}(t_2,h) = [2,3]$, $RV_{\rho_1}(t_3,h) = [4,5]$, and $RV_{\rho_1}(t_4,h) = [5,5]$. It means that:

- $sup(RV_\rho(t_2,C1)) = sup(RV_{\rho_1}(t_2,h)) - sup(RV_{\rho_1}(t_1,h)) = 3$.

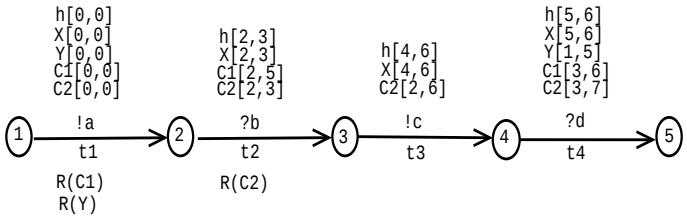


Figure 7.13: Phase two algorithm application.

- $sup(RV_\rho(t_3, C1)) = sup(RV_{\rho_1}(t_3, h)) - sup(RV_{\rho_1}(t_1, h)) = 5.$
- $sup(RV_\rho(t_4, C1)) = sup(RV_{\rho_1}(t_4, h)) - sup(RV_{\rho_1}(t_1, h)) = 5.$

In the same way, after computing the reached values of this path, we obtain Fig. 7.14.

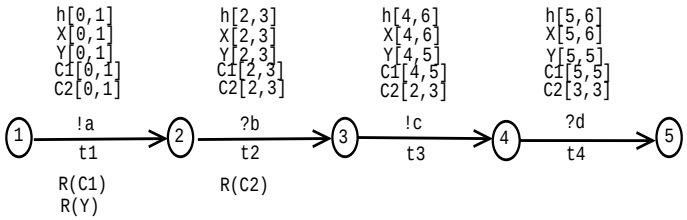


Figure 7.14: The reached values of the synchronous product.

We note by :

- FETC / SETC: Fastest/Slowest execution according to every transition crossed.
- FECT / SECT: Fastest/Slowest execution according to every clock of transition.

Table 7.15 summarizes the FETC, SETC, FECT and SECT of this path. For this example, the FETC for every transition is the same as the FECT. Transition $t_2$ has a SETC different from the SECT one. The total number of different executions of FETC, SETC, FECT and SECT, is between 1 and $2n$. The number the We notice that for $\lambda \in [0, 1]$, and for two executions $E$ and $E^{'}$ of executions given in table 7.15, $\lambda \times E + (1 - \lambda) \times E^{'}$ is also an execution of this path. All these executions can be used as test cases.

| Transitions | FETC | FECT |
|---|---|---|
| $t_1$ | $(0,!a).(2,?b).(4,!c).(5,?d)$ | $(0,!a).(2,?b).(4,!c).(5,?d)$ |
| $t_2$ | $(0,!a).(2,?b).(4,!c).(5,?d)$ | $(0,!a).(2,?b).(4,!c).(5,?d)$ |
| $t_3$ | $(0,!a).(2,?b).(4,!c).(5,?d)$ | $(0,!a).(2,?b).(4,!c).(5,?d)$ |
| $t_4$ | $(0,!a).(2,?b).(4,!c).(5,?d)$ | $(0,!a).(2,?b).(4,!c).(5,?d)$ |
| Transitions | SETC | SECT |
| $t_1$ | $(1,!a).(3,?b).(6,!c).(6,?d)$ | $(1,!a).(3,?b).(6,!c).(6,?d)$ |
| $t_2$ | $(1,!a).(3,?b).(6,!c).(6,?d)$ | $(0,!a).(3,?b).(5,!c).(5,?d)$ |
| $t_3$ | $(1,!a).(3,?b).(6,!c).(6,?d)$ | $(1,!a).(3,?b).(6,!c).(6,?d)$ |
| $t_4$ | $(1,!a).(3,?b).(6,!c).(6,?d)$ | $(1,!a).(3,?b).(6,!c).(6,?d)$ |

Figure 7.15: Test cases.

## 8. Discussion and Future Work

In this paper, we have presented a solution to the feasibility problem for a path in time $o((n \times k)^2)$. By introducing a global clock to a given timed automaton[6], the analysis of the feasibility problem becomes easier to solve. The use of this clock allows us to compute both the two versions of the fastest and slowest timed executions[7],and to introduce the strong/weak feasibility concept. Standard algorithms [4, 1] solve a two-terms inequalities in $o((n \times k)^3)$ time, and not allow the computation of the reached values for every clock except the global clock.

Applying our method to a synchronous product of the specification and the test purpose, we can have all test verdicts: conformance to specification and test purpose (pass verdict), conformance to specification but not to the test (inconclusive verdict) and conformance to test purpose but not to the specification or no conformance to specification and test purpose (fail verdict).

The test generation method proposed is potentially inefficient, because many paths may have to be synthesized and thereafter rejected by the feasibility analysis. The solution here is to modify the synchronous product by adding a checking step that consists in applying the method to every new node added to the already calculated path, and rejecting nodes that made this path unfeasible.

Our real-time model is quite restrictive and a generalization will be of benefit for real-time systems: especially the use of invariants and assignments. For the invariant of a state *s*, this invariant can be seen as the guard of a transition *t* that lead to *s*. If the specification includes constraints in the form $x - y \leq c \mid$

[6] If this automaton does not have already one.

[7] If bounded intervals.

$c \leq x - y$, by adding the inequality $\tau_k - \tau_j \leq c \mid c \leq \tau_k - \tau_j$[8] to algorithm (Fig. 4.3), we obtain the same result for the computation of the reached values of the global clock. We are interested in building a finite and complete test set for real-time systems as in [18]. We are working to widen the assignments, such that for a clock *x*, we allow *x=:a*, to add variables and parameters to the model and to see how to apply the method in the case of systems interoperability.

## Appendix A. Proof of Lemma 6

Proof. For $i$ in $[1, n]$, and a clock $x \in C(t_i)$, being last reset in $t_k$, $k < i$, we notice that: $sup(RV(t_i, h)) - sup(RV(t_k, h)) \in SC(t_i, x)$. In fact, according to lemma 3:

- For $sup(RV(t_i, h))$, $\exists \tau_k \in RV(t_k, h)$ such that $sup(RV(t_i, h)) - \tau_k \in SC(t_i, x)$. Then $sup(RV(t_i, h)) - sup(RV(t_k, h)) \leq sup(SC(t_i, x))$.
- For $sup(RV(t_k, h))$, $\exists \tau_i \in RV(t_i, h)$ such that $\tau_i - sup(RV(t_k, h)) \in SC(t_i, x)$. Then $inf(SC(t_i, x)) \leq sup(RV(t_i, h)) - sup(RV(t_k, h))$.

We construct then, the execution $E_S$ defined by, for every $i$ in $[1, n]$ and every $x$ in $C(t_i)$:

1. $E_S(t_i, h) = sup(RV(t_i, h))$.
2. $E_S(t_i, x) = sup(RV(t_i, h)) - sup(RV(t_k, h))$, where $x$ has been last reset in $t_k$.

The two-dimensional array $E_S$ is an execution of $\rho$ that verifies the first assertion of the lemma. We use a similar proof for the second assertion of the lemma.

## Appendix B. Proof of Lemma 7

Proof. Let us note by: for every $i$ in $[1, n]$, $\rho_{i\rho} = t_1 ... t_i$ the sub-path of $\rho$ obtained while replacing $SC(t_j, x)$ by $RV(t_j, x)_\rho$ for every $j$ in $[1, i]$. As the lemma assumes the feasibility of $\rho$, then $\rho_{i\rho}$ is also feasible.

Now, by induction, let us prove that, $\forall i \in [1, n]$, $\forall E^{'} \in ES(\rho_{i\rho}) \Rightarrow \exists E \in ES(\rho)$ such that $\forall j \in [1, i]$, $\forall x \in C(t_j)$, $E(t_j, x) = E^{'}(t_j, x)$.

For $i = 1$, and $E^{'} \in ES(\rho_{1\rho})$. All clocks are reset in the initial state, then $\forall x \in C(t_1) \Rightarrow E^{'}(t_1, x) = E^{'}(t_1, h)$. Let $\lambda \in [0, 1]$ such that $E^{'}(t_1, h) = \lambda \times E_{S\rho}(t_1, h) + (1 - \lambda) \times E_{F\rho}(t_1, h)$. Then $E := \lambda \times E_{S\rho} + (1 - \lambda) \times E_{F\rho}$ is an execution of $\rho$ that meets $E^{'}$ in $t_1$.

[8] Where *x* (resp. *y*) was last reset in $t_k$ (resp. $t_j$).

Let us assume that $\forall E^{'} \in ES(\rho_{(i-1)\rho}) \Rightarrow \exists E \in ES(\rho)$ such that $\forall j \in [1, i-1]$, $\forall x \in C(t_j)$, $E(t_j, x) = E^{'}(t_j, x)$, and let us prove that $\forall E^{''} \in ES(\rho_{i\rho}) \Rightarrow \exists E \in ES(\rho)$ such that $\forall j \in [1, i]$, $\forall x \in C(t_j)$, $E(t_j, x) = E^{'}(t_j, x)$.

While $ES(\rho_{i\rho}) \subset ES(\rho_{(i-1)\rho})$, according to the induction assumption, there is $E \in ES(\rho)$ such that $\forall j \in [1, i-1]$, $\forall x \in C(t_j)$, $E(t_j, x) = E^{''}(t_j, x)$. Let us define the paths $\rho_1 = t_1...t_n$ and $\rho_2 = t_1...t_n$ such that:

- For $\rho_1$: $\forall x \in C(t_i)$, $SC(t_i, x) := [E(t_i, x), E(t_i, x)]$.
- For $\rho_2$: $\forall x \in C(t_i)$, $SC(t_i, x) := [E^{''}(t_i, x), E^{''}(t_i, x)]$.

For these paths, we have changed the specification constraints over clocks in transition $t_i$. $E$ is an execution of $\rho_1$, then the constraints graph $G_{\rho_1}$, associated to the system $S_{\rho_1}$ obtained from the algorithm of Fig. 4.3 for $\rho_1$, does not have negative cycles. Let us prove that the constraints graph $G_{\rho_2}$, associated to the system $S_{\rho_2}$ does not have negative cycles. In fact, these two graphs differ only for edges weight between vertexes $\tau_i$ and $\tau_j$, for $j$ less than $i$. For each cycle $\sigma$ which passes only once by a vertex of $G_{\rho_2}$, we have:

- If $\sigma$ does not pass by $\tau_i$: $\sigma$ is also a cycle of $G_{\rho_1}$, and can not be a negative cycle.
- If $\sigma$ passes by $\tau_i$. Let $\tau_j$ and $\tau_k$ be the neighbor nodes of $\tau_i$ in $\sigma$. Let us assume that there is an edge form $\tau_j$ to $\tau_i$ of weight $p_{ji}$ and an edge from $\tau_i$ to $\tau_k$ of weight $p_{ik}$. We have the following cases:
  - If $j < i$ and $k < i$. From the construction of the constraints graph, $p_{ji}$ is negative and $p_{ik}$ is positive. Now if we consider the cycle $\sigma^{'}$ from $G_{\rho_1}$ which has the same nodes as $\sigma$, then $\sigma$ and $\sigma^{'}$ differ only on the weight of the two edges from $\tau_j$ to $\tau_i$ and from $\tau_i$ to $\tau_k$. Let be $p^{'}_{jk}$ and $p^{'}_{ik}$ the weights of this two edges in $G_{\rho_1}$. From the construction of the execution $E$, there is a $d \in \mathbb{R}$ such that $\forall x \in C(t_i)$, $E(t_i, x) = E^{''}(t_i, x) + d$ or $E(t_i, x) = E^{''}(t_i, x) - d$ (because $E$ and $E^{''}$ have the same values in every $t_j$, $j$ less than $i$). Then $p_{ji} + p_{ik} = p^{'}_{jk} + p^{'}_{ik}$. As $\sigma^{'}$ is positive then $\sigma$ is also positive.
  - If $j < i < k$ or $k < i < j$ . Let $x$ be the clock such that $SC(t_i, x) : [-p_{ji}, p_{ji}]$. If $\sigma$ is a negative cycle. Let $\rho^{'} = t_1...t_n$ be the path such as: $SC(t_i, x) := [-p_{ji}, p_{ji}]$. Then $\sigma$ is negative cycle of $\rho^{'}$. It means that $E^{''}(t_i, x) = p_{ji} \notin RV_{\rho}(t_i, x)$ is not a reached values of $x$ in $t_i$ of $\rho$. Then $\sigma$ is positive.

We have showed that $G_{\rho_2}$ does not have negative cycles, then $\rho_2$ is feasible. Let $E_1$ be an execution of $\rho_2$. The execution $E_2$ defined by:

- For every $j$ in $[1, i]$, and for every clock $x$ in $C(t_j)$: $E_2(t_j, x) = E^{''}(t_j, x)$.

- For every $k$ in $[i+1, n]$, and for every clock $x$ in $C(t_k)$: $E_2(t_k, x) = E_1(t_j, x)$.

is an execution of $\rho$ that meets $E^{''}$. We use a similar proof for the second assertion of the lemma.

## Appendix C. Proof of Lemma 8

Let $(a_1, a_2)$, $(b_1, b_2) \in \mathbb{R}^2$, and the suite $M$ and $N$ defined by:
$M_0 = N_0 = (a_1, a_2)$
$M_1 = N_1 = (b_1, b_2)$
$M_i = (m_{1i}, m_{2i})$, $N_i = (n_{1i}, n_{2i})$ such that:

- 1. If $m_{1(i-1)} = max(a_1, b_1)$:
  Let $j < i$ be the greatest index such that $m_{1j} < max(a_1, b_1)$, and $\alpha_i := min(m_{1(i-1)} - \frac{m_{1(i-1)}+m_{1j}}{2}, m_{2j} - \frac{m_{2(i-1)}+m_{2j}}{2})$.
  2. Else
  Let $j < i$ be the greatest index such that $m_{2j} < max(a_2, b_2)$ and $\alpha_i := min(m_{1j} - \frac{m_{1(i-1)}+m_{1j}}{2}, m_{2(i-1)} - \frac{m_{2(i-1)}+m_{2j}}{2})$.
  Then $M_i := (\frac{m_{1(i-1)}+m_{1j}}{2} + \alpha_i, \frac{m_{2(i-1)}+m_{2j}}{2} + \alpha_i)$.
- 1. If $n_{1(i-1)} = min(a_1, b_1)$:
  Let $j < i$ be the greatest index such that $n_{1j} \leq min(a_1, b_1)$ and $\alpha_i := min(\frac{n_{1(i-1)}+n_{1j}}{2} - n_{1(i-1)}, \frac{n_{2(i-1)}+n_{2j}}{2} - n_{2j})$.
  2. Else
  Let $j < i$ be the greatest index such that $n_{2j} \leq min(a_2, b_2)$ and $\alpha_i := min(\frac{n_{1(i-1)}+n_{1j}}{2} - n_{1j}, \frac{m_{2(i-1)}+m_{2j}}{2} - n_{2(i-1)})$.
  Then $N_i := (\frac{n_{1(i-1)}+n_{1j}}{2} - \alpha_i, \frac{n_{2(i-1)}+n_{2j}}{2} - \alpha_i)$.

**Lemma 12.** *Let be $(a_1, a_2)$ and $(b_1, b_2) \in \mathbb{R}^2$. Then:*

1. *The suite $M$ converges to $(max(a_1, b_1), max(a_2, b_2))$*
2. *The suite $N$ converges to $(min(a_1, b_1), min(a_2, b_2))$*

*Proof.* If we consider $(P_i)_i$ and $(Q_i)_i$ such that $P_i = m_{1i}$ if $m_{1i} \neq max(a_1, b_1)$ and $Q_i = m_{2i}$ if $m_{2i} \neq max(a_2, b_2)$. Then we can verify that $(P_i)_i$ and $(Q_i)_i$ are increasing, bounded and converge to $(max(a_1, b_1), max(a_2, b_2))$. □

**Lemma 13.** *Let us assume that $\rho$ is feasible. If $Card(C(t_n)) = 2$:*

1. *If for every $i$ in $[1, n]$, $RV(t_i, h)_\rho$ is bounded. Then there is an execution that takes simultaneously the $sup(RV(t_n, x)_\rho)$, for every $x$ in $C(t_n)$.*
2. *There is an execution that takes simultaneously the $inf(RV(t_n, x)_\rho)$, for every $x$ in $C(t_n)$.*

*Proof.* Let $x$ and $y$ be the two clocks of $C(t_n)$. There are two executions of $\rho$, $E_1$ and $E_2$, such that: $E_1(t_n, x) = sup(RV(t_n, x)_\rho)$, $E_2(t_n, y) = sup(RV(t_n, y)_\rho)$. Let $F = (F_i)_i$ be the suite of execution defined by:

- $F_0 = E_1$, $F_1 = E2$.
- $F_i = 1/2 \times F_{i-1} + 1/2 \times F_j$, $F_i(t_n, x) = F_i(t_n, x) + \alpha_i$, and $F_i(t_n, y) = F_i(t_n, y) + \alpha_i$ such that:
  1. $\alpha_i := min(F_{i-1}(t_n, x) - \frac{F_{i-1}(t_n,x)+F_j(t_n,x)}{2}, F_j(t_n, y) - \frac{F_{(i-1)}(t_n,y)+F_{2j}(t_n,y)}{2})$, if $F_{i-1}(t_n, x) = sup(RV(t_n, x))$. $j < i$ is the greatest index such that $F_j(t_n, x) < sup(RV(t_n, x)_\rho)$.
  2. $\alpha_i := min(F_j(t_n, x) - \frac{F_{i-1}(t_n,x)+F_j(t_n,x)}{2}, F_{i-1}(t_n, y) - \frac{F_{(i-1)}(t_n,y)+F_{2j}(t_n,y)}{2})$ otherwise. $j < i$ is the greatest index such that $F_j(t_n, y) < sup(RV(t_n, y)_\rho)$.

According to lemma 12, the suite $F$ converges to an execution $E$ such that $E(t_n, x) = sup(RV(t_n, x)_\rho)$ and $E(t_n, y) = sup(RV(t_n, y)_\rho)$. We have a similar proof for the second result. □

After proving the result of this lemma, we will use the induction to generalize to $m$ used clocks.

Proof of Lemma 8. We present here the proof of the lemma in the case of $Card(C(t_n)) = 3$. For any $m$ less than the number of clocks, the same proof holds by induction. We assume that the clocks of $C(t_n)$ are respectively $x$, $y$ and $h$. Let us note by:

- $A := sup(RV(t_n, h)_\rho)$
- $B := sup(RV(t_n, x)_\rho)$
- $C := sup(RV(t_n, y)_\rho)$

Let $E_1$, $E_2$ and $E_3$ be three executions of $\rho$ such that: $E_1(t_n, h) = A$, $E_2(t_n, x) = B$ and $E_3(t_n, y) = C$. Now we define three paths $\rho_1$, $\rho_2$ and $\rho_3$ such that:

- $\rho_1$ is the path $\rho$ without the constraint over $h$, and $SC(t_n, x) := [B, B]$, $SC(t_n, y) := [C, C]$.
- $\rho_2$ is the path $\rho$ without the constraint over $x$, and $SC(t_n, h) := [A, A]$. $SC(t_n, y) := [C, C]$.
- $\rho_3$ is the path $\rho$ without the constraint over $y$, and $SC(t_n, h) := [A, A]$, $SC(t_n, x) := [B, B]$.

All these paths are feasible. In fact, if $\rho_1^{'}$ is the path $\rho$ without the constraint over $h$, then $E_2$ and $E_3$ are executions of $\rho_1^{'}$. According to lemma 13, for this path there is an execution $E_1^{'}$ of $\rho_1^{'}$ such that : $E_1^{'}(t_n, x) = B$ and $E_1^{'}(t_n, y) =$

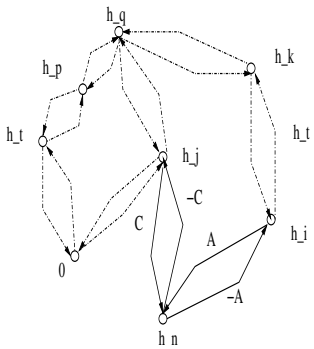


Figure C.16: Constraints graph associated to $\rho_2$.

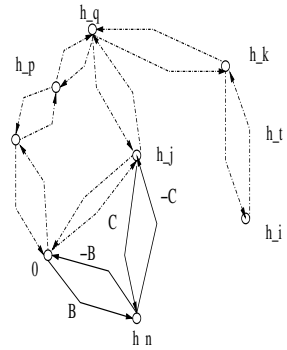


Figure C.17: Constraints graph associated to $\rho_1$.

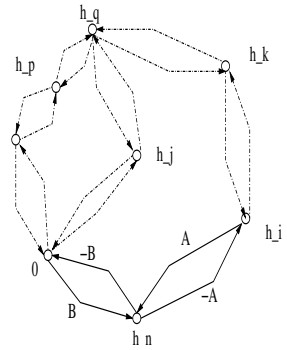


Figure C.18: Constraints graph associated to $\rho_3$.

$C$. Then $E_1^{'}$ is an execution of $\rho_1$. We have the same result for $\rho_2$ and $\rho_3$. Let us consider now, the path $\rho^{''}$ defined by:

- $\rho^{''}$ is the path $\rho$ such that $SC(t_n, h) := [A, A]$, $SC(t_n, x) := [B, B]$ and $SC(t_n, h) := [C, C]$.

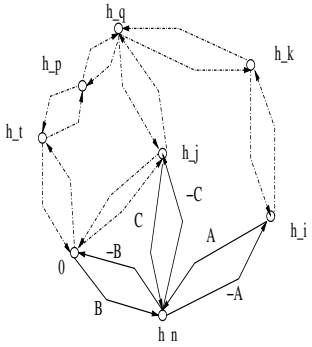


Figure C.19: Constraints graph associated to $\rho^{''}$.

Let us prove that $\rho^{''}$ is feasible. In fact, let $S_{\rho_1}$, $S_{\rho_2}$, $S_{\rho_3}$ and $S_{\rho^{''}}$ be the constraints systems obtained from the algorithm of Fig. 4.3 respectively for paths $\rho_1$, $\rho_2$, $\rho_3$ and $\rho^{''}$. Let $G_{\rho_1}$, $G_{\rho_2}$, $G_{\rho_3}$ and $G_{\rho^{''}}$ be the constraints graphs respectively associated to $S_{\rho_1}$, $S_{\rho_2}$, $S_{\rho_3}$ and $S_{\rho^{''}}$. The graphs $G_{\rho_1}$, $G_{\rho_2}$, $G_{\rho_3}$

(Fig. C.16, Fig. C.18 and Fig. C.17) do not have a negative cycle because theirs paths are feasible. The graph $G_{\rho''}$ (Fig. C.19) does not have either a negative cycle. In fact, if there is a negative cycle then there is at least one cycle $\sigma$ which passes one time by some nodes of the graph. If $\sigma$ passe by $h\_n$, then one of the graphs $G_{\rho_1}$, $G_{\rho_2}$, $G_{\rho_3}$ has a negative cycle. We use a similar proof for the second assertion of the lemma.

## Appendix D. Proof of Lemma 10

Proof. Let $E$ be in $ES(\rho)$. Let us prove, by induction, $\forall i \geq j$, $E_1(t_i, h) - E_1(t_j, h) \leq E(t_i, h) - E(t_j, h)$.

For $i = j + 1$, since $E_1$ is making the minimum delay between transitions after $t_j$, then it takes the minimum of the interval of a clock $y$ in $t_{j+1}$.

- If $y$ is last reset in $t_j$, then we have $E_1(t_{j+1}, y) \leq E(t_{j+1}, y)$. Since $E_1(t_{j+1}, h) - E_1(t_j, h) = E_1(t_{j+1}, y)$ et $E(t_{j+1}, h) - E(t_j, h) = E(t_{j+1}, y)$, we have then the result of the lemma.
- Else, Since $E(t_j, y) \leq E_1(t_j, y)$ and $E_1(t_{j+1}, y) \leq E(t_{j+1}, y)$, then $E_1(t_{j+1}, y) - E_1(t_j, y) \leq E(t_{j+1}, y) - E(t_j, y)$, which means that $E_1(t_{j+1}, h) - E_1(t_j, h) \leq E(t_{j+1}, h) - E(t_j, h)$.

Now, let us us assume the lemma for $j < i$ and let us prove the lemma for $i + 1$. Since $E_1$ is making the minimum delay between transitions after $t_j$, then it takes the minimum of the interval of a clock $y$ in $t_{i+1}$ :$E_1(t_{i+1}, y) \leq E(t_{i+1}, y)$.

- If $y$ is last reset in $t_k$, $k \geq j$. While:
  $E(t_{i+1}, h) - E(t_j, h) = E(t_{i+1}, h) - E(t_k, h) + E(t_k, h) - E(t_j, h)$,
  $E_1(t_k, h) - E_1(t_j, h) \leq E(t_k, h) - E(t_j, h)$, and $E(t_{i+1}, y) = E(t_{i+1}, h) - E(t_k, h) \leq E(t_{i+1}, y) = E(t_{i+1}, h) - E(t_k, h)$, then $E_1(t_{i+1}, y) - E_1(t_j, y) \leq E(t_{i+1}, y) - E(t_j, y)$
- Else, $y$ is last reset in $t_k$, $k < j$. While $E(t_j, y) \leq E_1(t_j, y)$ and $E_1(t_{i+1}, y) \leq E(t_{i+1}, y)$, then $E_1(t_{i+1}, y) - E_1(t_j, y) \leq E(t_{i+1}, y) - E(t_j, y)$

We use a similar proof for the second assertion of the lemma.

## References

[1] Robert W. Floyd. Algorithm 97 (shortest path),*Communications of the ACM*,18(3):165-172, 1964.

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

[3] R. Alur, R. Kurshan, and M. Viswanathan. Membership problems for timed and hybrid automata, 19th IEEE Real-Time Systems Symposium, 1998.

[4] B. Aspvall, Y. Shiloach. A Polynomial Time Algorithm for Solving Systems of Linear Inequalities with two Variables per Inequalities, *20th Annual Symp. on Foundations of Computer Sciences,* Oct. 1979, 205-217.

[5] B. Bérard, P. Castéran, E. Fleury, L. Fribourg, J.-F. Monin, C. Paulin, A. Petit, and D. Rouillard. Automates temporisés calife, Fourniture F1.1, Calife, 2000.

[6] S. Bloch, H. Fouchal, E. Petitjean, S. Salva. Some issues on testing real time systems. *Int. Journal of Computer and Information Science*, n2, Vol. 4, December 2001.

[7] Rachel Cardell-Oliver. Conformance Testing of Real-Time Systems with Timed Automata Specifications, *Formal Aspects of Computing*, 12(5):350-371,2000.

[8] Duncan Clarke and Insup Lee. Automatic Test Generation for the Analysis of a Real-Time System: Case Study. In *3rd IEEE Real-Time Technology and Applications Symposium*, 1997.

[9] A. En-Nouaary, R. Dssouli, F. Khenedek, and A. Elqortobi. Timed test cases generation based on state characterization technique, *In 19th IEEE Real Time Systems Symposium (RTSS'98)*, Madrid, Spain, 1998.

[10] H. Fouchal. Adapted test cases for timed systems. *Journal of Electronics and Computer Science*, n1, vol. 3, pp:1-8, December 2001.

[11] Anders Hessel, Kim G. Larsen, Brian Nielson, Paul Pettersson, and Arne Skou. Time-optimal Real-time Test Case generation using UPPAAL, To Appear.

[12] T. Higashino, A. Nakata, K. Taniguchi, and R. Cavalli. Generating Test Cases for a Timed I/O Automaton model, *TESTCOM99*, Budapest, Hungary, September 1999.

[13] O. Koné, P. Laurencot and R. Castanet. On the Fly Test Generation for Real Time Protocols. In *International Conference on Computer Communications and Networks*, Louisiana U.S.A, 1998.

[14] A. Koumsi, M. Akalay, R. Dssouli, A. En-Nouaary, L. Granger. An approach for testing real time protocols, *TESTCOM*, Ottawa, Canada, 2000.

[15] L. Kaiser. Interopérabilité temporelle d'automates temporisés, PhD thesis, Université H. Poincaré, Nancy, 2002.

[16] Dino Mandrioli, Sandro Morasca, and Angelo Morzenti. Generating Test Cases for Real-Time Systems from Logic Specifications, *ACM Transactions on Computer Systems*, 13(4):365-398, 1995.

[17] E. Petitjean. Etude des méthodes de test sur les systèmes temporisés, PhD thesis, Université de Reims, France, November 2000.

[18] Jan Springintveld, Frits Vaandrager, Pedro R. D'Argenio. Testing Timed Automata. *Theoretical Computer Science*, 252(1-2):225-257, March 2001.

**Authors addresses:**
LaBRI - CNRS - UMR 5800 Université Bordeaux 1
33405 Talence cedex France
berrada@labri.fr, castanet@labri.fr, felix@labri.fr