Timed Wp-Method: Testing Real-Time Systems

Abdeslam En-Nouaary, Rachida Dssouli, and Ferhat Khendek

Abstract—Real-time systems interact with their environment using time constrained input/output signals. Examples of real-time systems include patient monitoring systems, air traffic control systems, and telecommunication systems. For such systems, a functional misbehavior or a deviation from the specified time constraints may have catastrophic consequences [27]. Therefore, ensuring the correctness of real-time systems becomes necessary. Two different techniques are usually used to cope with the correctness of a software system prior to its deployment, namely, verification and testing. In this paper, we address the issue of testing real-time software systems specified as a *Timed Input Output Automaton (TIOA)*. TIOA is a variant of timed automaton [1], [2], [22], [29]. We introduce the syntax and semantics of TIOA. We present the potential faults that can be encountered in a timed system implementation. We study these different faults based on TIOA model and look at their effects on the execution of the system using the region graph. We present a method for generating timed test cases. This method is based on a state characterization technique and consists of the following three steps: First, we sample the region graph using a suitable granularity, in order to construct a subautomaton easily testable, called *Grid Automaton*. Then, we transform the Grid Automaton into a *Nondeterministic Timed Finite State Machine (NTFSM)*. Finally, we adapt the Generalized Wp-method [23] to generate timed test cases from NTFSM. We assess the fault coverage of our test cases generation method and prove its ability to detect all the possible faults. Throughout the paper, we use examples to illustrate the various concepts and techniques used in our approach.

Index Terms—Testing, specification, implementation, timed automaton, real-time systems, fault coverage.

1 Introduction

Nowadays, software is used to control safety critical systems such as patient monitoring, air traffic control, plant control, and telecommunication equipments. Moreover, we witness the rapid development and deployment of new time dependent applications such as multimedia applications. Contrary to nontimed systems, the functions of real-time systems are time constrained and dependent. Indeed, the behavior of a real-time system does not depend only on the values of input and output signals, but also on their time of occurrence. Ensuring the correctness of such systems before the deployment, i.e., ensuring that it functions correctly within the specified time constaints, is a difficult and complex task.

Formal methods are often used as means to cope with complexity. Many formal models have been proposed for real-time systems [2], [26]. They are mainly real-time enrichments of well-known models, such as Finite State Machines, Petri Nets, etc. It has been reported in [27] that misbehaviors of time dependent systems are often due to nonrespect of time constraints. Verification and testing are two different techniques that are usually used to cope with the correctness of a system. Verification deals with the specification of the system under consideration and aims to ensure that the designed specification satisfies predefined functional and timing requirements (see [2] for instance). The correctness of the system specification does not guarantee the correctness of its implementation (the end product). Testing is an important activity, which aims to ensure the quality of the implementation. Testing procedure consists of generating test suites and applying them to the implementation which is referred to as an *Implementation Under Test (IUT)*. There exist mainly three testing strategies: *white-box testing*, *black-box testing*, and *gray-box testing*. In white-box testing, the structure of the implementation is known and the test suite is generated from the implemented structures. In black-box testing, however, the structure of the implementation is not known; we use the specification of the required functionality at defined interfaces for test generation, execution, and evaluation. Finally, in gray-box testing, we assume that the modular structure of the implementation is known but not the details of the programs within each component. In this paper, we focus on gray-box testing and we refer to it as testing.

An important aspect in the testing of an implementation of a system is the fault model. It refers to all the potential basic faults that can exist in an implementation. The test cases we generate, with a test cases generation technique, are intended to detect these faults. Actually, some techniques may be able to detect all the potential faults, while other techniques may fail in detecting some faults. The power of a test cases generation technique to detect faults in an implementation is referred to as *fault coverage* [36], [39], [38], [6], [8].

Test cases generation methods can be compared based on their respective fault coverage. We can say that a method A is more powerful than a method B, if A has a better fault coverage than B. In other words, a method A is more powerful than a method B, if A detects more faults than B. However, for a more accurate comparison between test cases generation techniques, other parameters such as the length of test suites should be taken into account.

• *The authors are with the Department of Electrical and Computer Engineering, Concordia University, 1455 de Maisonneuve W., Montreal, Quebec H3G 1M8, Canada.*
E-mail: {ennouaar, dssouli, khendek}@ece.concordia.ca.

Manuscript received 31 Jan. 2000; revised 15 June 2001; accepted 21 Dec. 2001.
Recommended for acceptance by A.A. Andrews.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 111424.

In this paper, we first present a fault model for timed systems. Then, we present the algorithm for test cases generation, and assess its fault coverage. The model we use to specify timed systems is the *Timed Input Output Automaton (TIOA)*, a variant of timed automaton [2], [29], [22], in which clocks are real-valued variables, increase synchronously at the same speed and measure the amount of time elapsed since last initialization or reset. In addition, this model consists of all clocks having bounded domains [31] to indicate that clock values are relevant only under a certain integer constant. In order to generate timed test cases, we proceed as follows: First, as in [2], we use the region graph as semantics for the TIOA to explicit the elapsing of time and the relationship between clocks. Secondly, we sample the region graph for a defined granularity in a way that each state has an outgoing transition labeled with the same delay (i.e., the granularity of sampling). This leads to a reduction of the region graph into a *Grid Automaton*, which is then transformed into a *Nondeterministic Timed Finite State Machine (NTFSM)*. Finally, we adapt a well-known method [18], [23] for the generation of test cases. As a result of these transformations, our test method, Timed Wp-method, has a good fault coverage and certainly a practical value.

The remainder of this paper is structured as follows: Section 2 is devoted to the notations and definitions we use in addition to the syntax and the semantics of TIOA. Section 3 presents the test hypotheses used to generate timed test cases and to assess to fault coverage of our method. Section 4 introduces our test architecture. Section 5 discusses the conformance relation that should hold between the IUT and its specification. Section 6 presents our fault model for timed systems based on TIOA model. Section 7 is devoted to the timed test cases generation method, Timed Wp-method. In Section 8, we assess the fault coverage of Timed Wp-method. In Section 9, we discuss the other timed test cases generation techniques and compare them to our method. We conclude in Section 10.

2 Notations and Definitions

In the subsequent sections of the paper, $\mathbf{R}$ denotes the set of reals, $\mathbf{R}^{\geq 0}$ the set of nonnegative reals, $\mathbf{R}^{+\infty} = \mathbf{R}^{\geq 0} \cup \{+\infty\}$, $\mathbf{N}$ the set of nonnegative integers, $\mathbf{N}^{>0} = \mathbf{N}\backslash\{0\}$ and $\mathbf{N}^{+\infty} = \mathbf{N} \cup \{+\infty\}$. For $t \in \mathbf{R}^{\geq 0}$, $\lfloor t \rfloor$ denotes the largest number in $\mathbf{N}$ that is not greater than t, i.e.,

$$\lfloor t \rfloor = max\{m \in \mathbf{N} | m \leq t\},$$

and $fract(t)$ denotes the fractional part of t, i.e., $fract(t) = t - \lfloor t \rfloor$. An interval I is a convex subset of $\mathbf{R}^{+\infty}$. An integer interval is an interval I with its both bounds ($Inf(I)$ and $Sup(I)$) in $\mathbf{N}^{+\infty}$.

The concatenation of two finite sets V_1 and V_2 is denoted by "." and defined as follows:

$$V_1.V_2 = \{v_1.v_2 \mid v_1 \in V_1 \wedge v_2 \in V_2\},$$

where $v_1.v_2$ stands also for the concatenation of the sequences v_1 and v_2. Let V^n denotes n-times concatenation of V ($V^n = V.V^{n-1}$) and $V^0 = \varepsilon$, where ε is the empty sequence.

Definition 2.1: Timed Input Output Automaton. *A Timed Input Output Automaton (TIOA) A is a tuple* $(I_A, O_A, L_A, l_A^0, C_A, T_A)$, *where:*

- I_A *is a finite set of input actions that the TIOA receives from the environment. Each input begins with "?,"*
- O_A *is a finite set of output actions that the TIOA sends to the environment. Each output begins with "!,"*
- L_A *is a finite set of locations,*
- $l_A^0 \in L_A$ *is the initial location,*
- C_A *is a finite set of clocks all initialized to zero in* l_A^0, *and*
- T_A *is the set of transitions.*

A tuple $(l, l', \{?,!\}a, R, G) \in T_A$, denoted in the rest of the paper with $l \xrightarrow{\{?,!\}a,R,G}_A l'$, represents a transition from location l to location l' on input or output action a (denoted by $\{?,!\}a$). The subset $R \subseteq C_A$ specifies the clocks to be reset in this transition and $G \in \Phi(C_A)$ is a clock guard (time constraint) for the execution of the transition. We assume that transitions are instantaneous in TIOA. The term $\Phi(C_A)$ denotes the set of all guards over C_A built using Boolean conjunction over atomic formulas of the form $x < m$, $x \leq m$, $x = m$, $x > m$, and $x \geq m$, where $x \in C_A$ and $m \in N$. The operators $\leq$, $=$, $\geq$, and $>$ are particularly used in output action constraints. The choice of naturals as bounds in constraints will help us, later, in the discretization of the set of reals into integer intervals, reducing thereby the state space of timed systems. In this definition, we assume that each clock $x \in C_A$ has a domain $[0, C_x] \cup \{\infty\}$, where C_x is the largest integer appearing in a constraint over x in the automaton, i.e., $C_x = max\{c \mid ((x < c) \vee (x \leq c) \vee (x = c) \vee (x > c) \vee (x \geq c))$ *is a constraint over* $x\}$. This means that the value of each clock x is relevant only under the integer constant C_x. So, we will represent each clock value greater than this constant by $+\infty$ or simply ∞, and we write: $\forall \varepsilon > 0, \forall x \in C_A, C_x + \varepsilon = \infty$. Moreover, we define the inclusion between clock domains as follows: A clock domain $[0, C_x] \cup \{\infty\}$ is included into another clock domain $[0, C'_x] \cup \{\infty\}$ if and only if $C_x \leq C'_x$.

Fig. 1 shows a TIOA with an input action $?In$, an output action $!Out$, two locations l_0 (the initial location) and l_1, two clocks x and y, and three transitions. For example, the transition $l_1 \xrightarrow{?In,\{x,y\},x<1 \wedge y<2} l_0$ is executed on input action $?In$ only when the values of x and y are, respectively, less than 1 and 2. In that case, it resets clocks x and y and brings the machine back to its initial location l_0.

Informally, the semantics of TIOA are as follows: The TIOA starts at initial location with all clocks initialized to zero. Then, the values of clocks increase synchronously and measure the amount of time elapsed since they have been last initialized or reset. At any time, the machine whose current location is l can make a transition $l \xrightarrow{\{?,!\}a,R,G} l'$ provided the current values of clocks satisfy the time constraint G. In this case, all the clocks in R are reset and the machine goes to location l'. To formally describe these

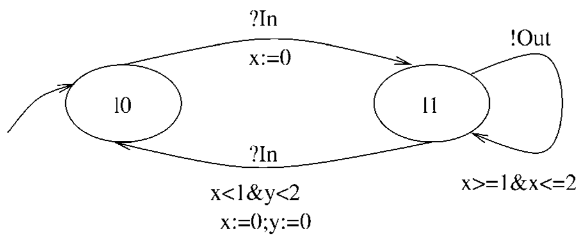

Fig. 1. A TIOA example.

semantics, we need to define first the clock valuation and the state of a TIOA.

Definition 2.2: Clock valuation. *A clock valuation over a set of clocks C is a map v that assigns to each clock $x \in C$ a value in $\mathbf{R}^{+\infty}$. We denote the set of clock valuations by $V(C)$.*

A clock valuation v satisfies a clock guard G, denoted $v \models G$, if and only if G holds under v.

For $d \in \mathbf{R}^{\geq 0}$, $v + d$ denotes the clock valuation that assigns a value $v(x) + d$ to each clock x. For $X \subseteq C$, $[X := d]v$ denotes the clock valuation for C that assigns the value d to each $x \in X$, and agrees with v over the remaining clocks.

Definition 2.3: States of TIOA. *The state of a TIOA $A = (I_A, O_A, L_A, l_A^0, C_A, T_A)$ is a pair (l, v), where l is a location ($l \in L_A$), and v is a clock valuation ($v \in V(C_A)$). The initial state of A is represented by (l_A^0, v_0), where $v_0(x) = 0$ for each clock $x \in C_A$.*

The set of all states is denoted by S_A.

The semantic model of a TIOA A is given by a timed labeled transition system $S(A)$ that consists of the state set S_A, the label set $\mathbf{R}^{\geq 0} \cup (\mathbf{I_A} \cup \mathbf{O_A})$ (input/output actions and time increments), and the transition relation $\xrightarrow{a}$, for $a \in \mathbf{R}^{\geq 0} \cup (\mathbf{I_A} \cup \mathbf{O_A})$. The timed labeled transition system $S(A)$ is infinite (because the delay transitions are infinite). So, it cannot be used for test generation. The solution is therefore to cluster equivalent states of $S(A)$ into equivalent classes (*clock regions*) by using the equivalence relation $\sim$ [2] on the set of clock valuations $V(C_A)$. The resulting automata is called *region graph* or *region automata*.

Definition 2.4: Clock region. *Let $A = (I_A, O_A, L_A, l_A^0, C_A, T_A)$ be a timed input output automaton, v and v' $\in V(C_A)$; we say $v \sim v'$ iff :*

1. $\forall x_i \in C_A, \lfloor v(x_i) \rfloor = \lfloor v'(x_i) \rfloor$,
2. $\forall x_i, x_j \in C_A \mid ((v(x_i) \neq \infty) \wedge (v(x_j) \neq \infty))$, $(fract(v(x_i)) \leq fract(v(x_j)) \Leftrightarrow fract(v'(x_i)) \leq fract(v'(x_j)))$, *and*
3. $\forall x_i \in C_A \mid v(x_i) \neq \infty, (fract(v(x_i)) = 0 \Leftrightarrow fract(v'(x_i)) = 0)$.

A clock region for A is an equivalence class of clock valuations induced by $\sim$. Let $[v]$ denotes the clock region to which v belongs.

The clock regions corresponding to the TIOA in Fig. 1 are shown in Fig. 2. Here, we have 44 clock regions, each of them is identified by a number and characterized by a set of linear inequations. Moreover, we distinguish between three kinds of clock regions: the corner points (e.g., region R1: $x = y = 0$), the open line segments (e.g., region R10: $0 < x < 1$ and $y = 0$), and the open regions (e.g., region R32: $0 < y < x < 1$).

Definition 2.5: Region Graph. *Let $A = (I_A, O_A, L_A, l_A^0, C_A, T_A)$ be a timed input output automaton. A region graph (or a region automaton) of A is an automaton $RG = (\Sigma_{RG}, S_{RG}, s_{RG}^0, T_{RG})$ where:*

- $\Sigma_{RG} = (I_A \cup O_A) \cup \mathbf{R}^{>0}$,
- $S_{RG} = \{(l, [v]) \mid l \in L_A \wedge v \in V(C_A)\}$,
- $s_{RG}^0 = (l_A^0, [v_0])$, *where* $v_0(x) = 0$ *for all* $x \in C_A$,
- *RG has a transition* $s \xrightarrow{\{?,!\}a} s'$, *from* $s = (l, [v])$ *to* $s' = (l', [v'])$ *on action* $\{?,!\}a \in (I_A \cup O_A)$ *iff there is a transition* $l \xrightarrow{\{?,!\}a,R,G}_A l'$ *such that* $v \models G$ *and* $v' = [R := 0]v$, *and*
- *RG has a delay transition* $s \xrightarrow{d} s'$, *from* $s = (l, [v])$ *to* $s' = (l, [v'])$ *on time increment* $d > 0$, *iff* $[v'] = [v + d]$.

Fig. 3 shows the region graph for the TIOA given in Fig. 1. The symbol d ($0 < d < 1$) is a real number representing the progression of time. The terms l_0 and l_1 are the locations of TIOA, and the symbols $Ri, i \in \mathbf{N}^{>0}$ are clock regions of Fig. 2. Contrary to Fig. 2, Fig. 3 shows only the clock regions reachable from the initial state $(l_0, R_0) = (l_0, x = y = 0)$. The system starts at its initial state with all its clocks set to 0 (the clock region $R_1 : x = y = 0$). Then, with the progression of time, the system reaches the region $R_{16} : 0 < x = y < 1$, $R_5 : x = y = 1$, $R_{22} : 1 < x = y < 2$, $R_7 : x = y = 2$, and $R_{42} : x = y > 2$. At any of these states, the system can make a transition on input $?In$ and enters a new state defined by location l_1 with clock x set to 0 and clock y being unchanged. From this new state, the system evolves by letting time elapsing as before and/or making explicit transitions on actions $?In$ and $!Out$.

The region graph corresponds to the reachability analysis graph. It is at the heart of any formal technique for timed systems (see, for example, [2], [29], [32], [13], [14]). Here, we use it as basis for testing real-time systems. In the definition of the region graph, we explicitly represent the delay transitions of the system because they are necessary for the generation of timed test cases.

Prior to the introduction of our timed test cases generation method, we introduce our testing hypotheses, our test architecture, the conformance relation that should hold between the implementation and the specification, and the timed fault model.

3 Test Hypotheses

The potential number of implementations for a given system specification, even a very simple one, is infinite. We use test hypotheses to reduce the number of possible implementations to consider in testing. We study fault coverage and other properties of testing techniques under these hypotheses. In this paper, as in [7], [18], [32], [23], we make few assumptions about the specification (a TIOA A) and the implementation under test. Some of these assumptions are necessary for the existence of the characterization set (see Definition 7.6) for the generation of test cases. Other assumptions are necessary for the testing of the implementation. We assume the following:

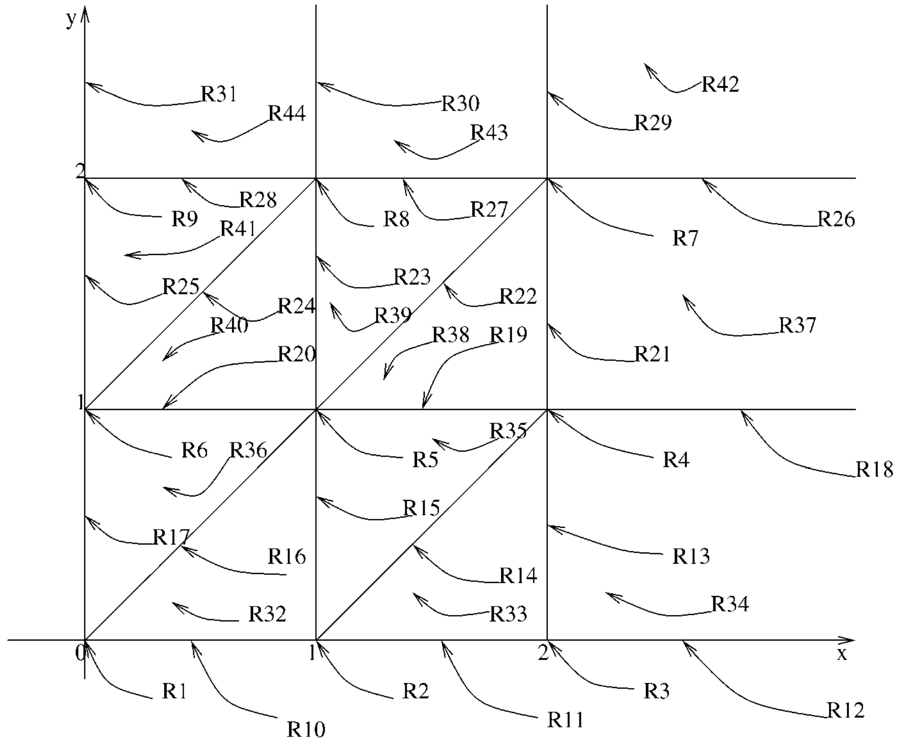

Fig. 2. Clock regions.

1. The IUT is modeled as a TIOA: In testing, we usually assume that the implementation and the specification are given in the same formal model (e.g., FSM, EFSM, LTS, TIOA, ...) in order to define the meaning of conformance of the implementation to its specification (i.e., what does it mean the implementation conforms to the specification ?). The specification is given as a TIOA, the implementation can also be seen as a TIOA.
2. The IUT is deterministic: For each location, there are no two (or more) outgoing transitions where time constraints may be simultaneously satisfied.
3. The IUT is minimal on the number of clocks [12]. This assumption is necessary to reduce the number of clock regions in the implementation.
4. The time domain of each clock in the IUT is included in the time domain of its corresponding clock in the specification (see Definition 2.1). This hypothesis ensures that the number of clock regions of the IUT is not greater than that of the specification.
5. The IUT has the same alphabet as the specification.
6. The IUT has the same number of locations as the specification. Used with hypotheses 4 and 3, this assumption guarantees that the number of states in the region graph of a correct IUT is not greater than that of the region graph of the specification. So, if we do not use hypothesis 6, the region graph of the IUT may be larger than that of the specification. Consequently, it become very difficult to determine the suitable granularity for the sampling algorithm (see Section 7.1). Indeed, the difference between the number of states in the implementation and the specification influences the granularity of sampling. For example, if the implementation has more states than the specification the granularity should be reduced in order to reach the extra states.
7. The IUT is completely specified: at any state, the implementation either accepts all inputs or accepts none.
8. For testing the implementation, we have to bring it back to its initial state after each single test case in order to apply the next test case. Therefore, we assume the existence of a reset action, which always brings the implementation to its initial state.
9. The complete-testing assumption: It is possible, by applying a given input sequence to a given implementation a finite number of time, to cover all execution paths that can be traversed by this sequence. This is due to the outputs which are unpredictable. So, the hypothesis 9 is needed to cover their time domains by verifying successively all possible intervals.

4 Test Architecture

The region graph defines the semantics of the TIOA model. However, it is very difficult to test since it is not explicit how to handle the clock variables. For this reason, we propose a testable model [13], [20], [21] which consists of two parts (see Fig. 4): the *control part* and the *clock part*.

The control part represents the communication of the timed system with its environment by receiving inputs and responding with outputs. The clock part handles the clock variables used in the specification of the timed system. Each clock variable gives rise to a process to perform the operations on the variable. The communication between

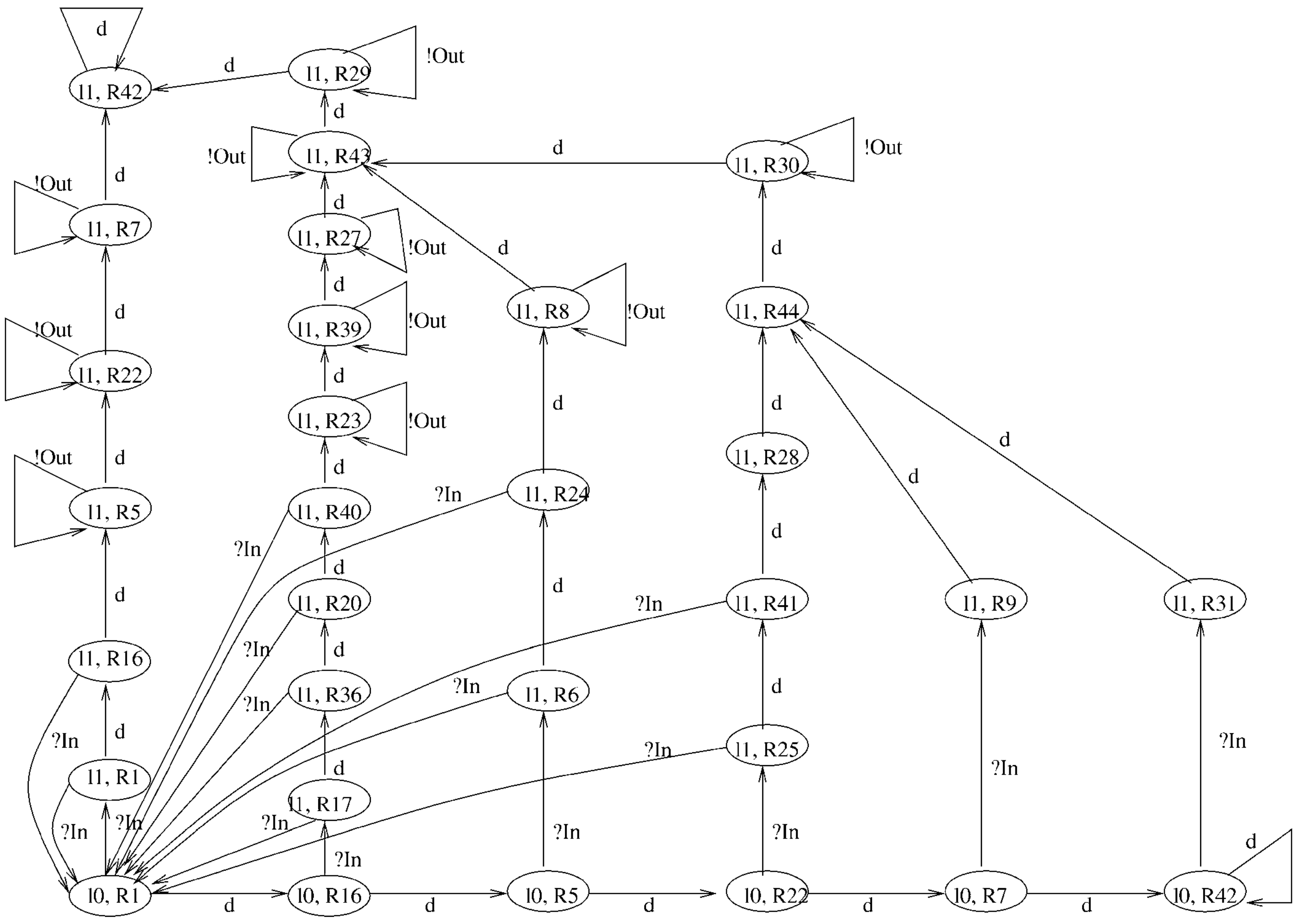

Fig. 3. The region graph of the TIOA given in Fig. 1.

the two parts is achieved with the exchange of the predefined internal signals *PleaseValue*, *SendValue*, and *ResetClock*. When the control part receives an input from the environment, it checks whether or not the input constraint is satisfied. It sends the *PleaseValue* signal to the involved clock processes asking for the clock values. After

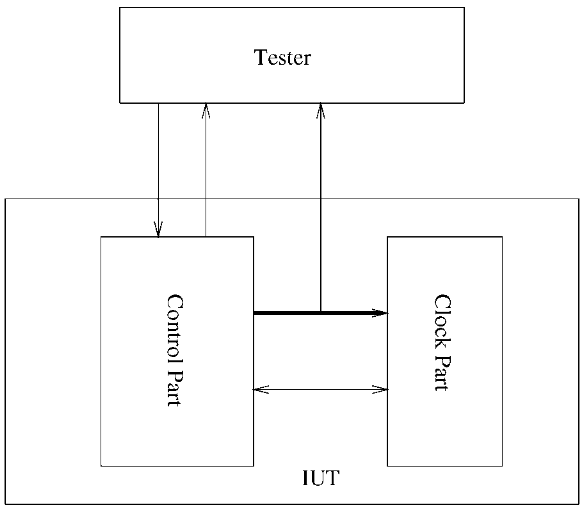

Fig. 4. Test architecture.

the reception of this signal, each process computes the current value of its clock and passes it to the control part using the signal *SendValue*. To reset a clock to zero, the control part sends the signal *ResetClock* to the corresponding clock process. In this architecture, we explicitly distinguish the *ResetClock* channel from the other signals channel (it is made in bold in Fig. 4) because *ResetClock* is the only internal signal we want to observe. We are assuming a synchronous communication.

The model of Fig. 4 represents an efficient and simple manner to implement and test real-time systems. It corresponds to the *gray-box testing* where we assume that some parts of the IUT are known and reachable by the tester. Moreover, our model is well-known in hardware where the clock component is separated from the other components. The primary advantage of this architecture is that it makes explicit the reset to zero of clocks and allows the tester to observe it. So, it helps enough to ensure a complete fault coverage of our method. Without it, some faults can not be detected by the test cases generated by our approach.

5 Conformance Relation

The starting point for conformance testing is the definition of the conformance relation between an IUT and the

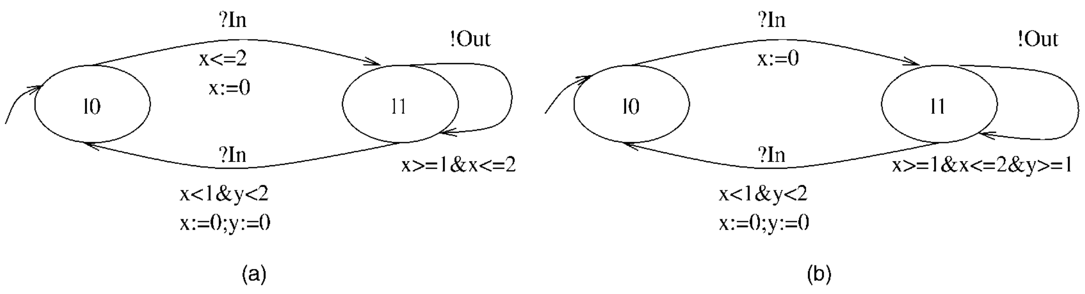

Fig. 5. Effective and noneffective timing faults.

specification. A conformance relation defines the meaning of "an implementation is conform to its specification." Many conformance relations have been introduced and used to generate test cases from FSM and LTS models [18], [23], [4]. They assume the environment behaves accordingly to the reference specification and the verdict for whether or not the IUT conforms to the specification is given by observing the implementation reactions (i.e., outputs) to the applied inputs. To test an implementation of a real-time system, we have to check whether or not the IUT when stimulated with inputs responds with expected outputs within the *allowed time intervals*.

In this paper, we define the conformance relation between an IUT and its specification as a trace-equivalence for observable nondeterministic FSM [23]. The trace-equivalence relation is based on the notion of traces of a state. By definition, a sequence of input/output actions σ is a trace of a state s if the automaton can evolve from s to another state s' on σ (s' can be equal to s). We denote the set of traces of a state s by $traces(s)$ and we say that two states s and s' are trace-equivalent, notation $s =_{trace} s'$, if and only if $traces(s) = traces(s')$. Furthermore, two FSMs S and I with their respective initial states s_0 and i_0, are trace-equivalent, notation $S =_{trace} I$, if and only if $s_0 =_{trace} i_0$. The trace-equivalence is an equivalence relation.

6 Fault Model

A fault model specifies the faults we can encounter in an implementation of a system. The fault model is always dependent on the specification model. In this section, we study the timed fault model based on TIOA.

6.1 TIOA Based Fault Model

The TIOA based fault model introduced in [15], [16] consists of two types of faults:

- Timing faults (see Section 6.2), and
- Action and Transfer faults (see Section 6.3).

Moreover, we distinguish between two types of timing faults: *effective* and *noneffective* timing faults. A timing fault is said to be effective if it has an effect on the execution of the system; otherwise, it is noneffective. In other words, a timing fault is effective if it changes the region graph of the system except for transitions on output actions (see Section 6.2.2). So, an implementation containing noneffective faults has the same region graph as the specification. Fig. 5 illustrates the effective and noneffective faults relative to the TIOA in Fig. 1. The first TIOA restricts the time constraint of the transition such that the implementation does not accept the input $?In$ when the value of clock x is greater than 2. This restriction affects the execution of the implementation in the sense that the region graph may have a small number of states and/or transitions. However, the second TIOA shows a modification of the time constraint of the transition from l_1 to l_1. This modification has no effect on the execution of the implementation since the region graph remains unchanged. Indeed, in location l_1 clock y is no less than clock x. Therefore, if the constraint $x \geq 1$ holds then the constraint $y \geq 1$ holds too. Consequently, the time constraint $x \geq 1 \& x \leq 2 \& y \geq 1$ is equivalent to $x \geq 1 \& x \leq 2$. The following subsections describe in details the effective timing faults (referred to, in the rest of this paper, as timing faults), and action and transfer faults.

6.2 Timing Faults

Timing faults are due to the violation of transition time constraints. A timing fault is either related to the reset of a clock, the restriction of a transition constraint, or the widening of a transition constraint.

6.2.1 Reset of a Clock Fault

An implementation is said to have a reset of clock fault if it does not reset a clock as stated in the specification, or it resets a clock that is not reset in the specification. As an example, we consider again the TIOA in Fig. 1. Fig. 6 and Fig. 7 show the two types of faults. The implementation in Fig. 6 does not reset clock x with the transition from l_0 to l_1. However, the implementation in Fig. 7 resets clock y with the transition from l_1 to l_1.

The faults related to a reset of a clock change the ordering between clocks in the implementation. In the example above, the implementation in Fig. 6 changes the ordering between clocks from $(x \leq y)$ to $(x = y)$ since the specification requires that clock x must be no greater than clock y in location l_1. However, the implementation in Fig. 7

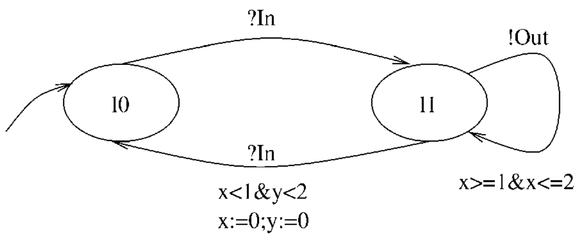

Fig. 6. "Nonreset of a clock" fault.

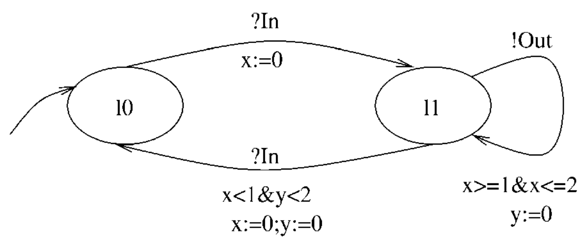

Fig. 7. "Reset of a clock" fault.

changes the ordering between clocks from $(x < y)$ to $(x \geq y)$. Concerning the number of states in the system, notice that:

- When an implementation does not implement a reset of a clock, the number of states in its region graph will decrease. This means that some states of the system will be unreachable (e.g., $(l_0, 1 < x < y < 2)$). As an example, the region graph of the faulty implementation in Fig. 6 contains 12 states while the regions graph of its specification has 31 states (Fig. 3).
- When an implementation resets a clock that remains unchanged in the specification, it is considered faulty and the number of states in its region graph will increase (some extra states will be reachable (e.g., $(l_1, x = 1 \& y = 0)$)). As an example, the number of states in the region graph of the faulty implementation in Fig. 7 is 42 instead of 31 as stated in the specification (Fig. 3).

6.2.2 Time Constraint Restriction Faults

We distinguish between the restriction of a time constraint for inputs and outputs. The environment controls the system inputs. Therefore, an implementation that rejects inputs satisfying the time constraint given by the specification is considered faulty. However, an implementation that restricts the time constraint of an output is seen as a valid reduction of the specification. Obviously, this is acceptable since outputs cannot be controlled by the environment. To further clarify the restriction of a time constraint, let us consider again the specification in Fig. 1. Fig. 8 and Fig. 9 show, respectively, restrictions of time constraint for inputs and outputs.

The implementation of Fig. 8 narrows the time constraint of the transition from l_1 to l_0 on input $?In$. This implementation does not accept the input $?In$ when the value of clock x is less than 1 and the value of clock y is between 1 and 2, exclusively. This reduces the number of states and transitions in its region graph compared with the specification region graph. In our example, the number of states in the region graph of the implementation is 23 while the number of states in the specification region graph is 31. On the other hand, the implementation of Fig. 9 modifies the time constraint of the transition from l_1 to l_1 on output $!Out$. This implementation always responds with output $!Out$ before the value of clock x reaches 2. But, since output actions are controlled by the implementation, the constraint of the transition in the specification is always satisfied. Therefore, the implementation is not faulty but considered as a valid reduction of the specification.

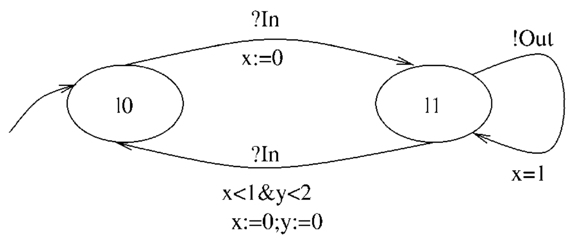

Fig. 9. Restriction of constraints for outputs.

6.2.3 Time Constraint Widening Faults

These faults occur when the implementation either increases the upper bound or decreases the lower bound of a constraint but satisfies hypothesis 4. More precisely, there are four types of this fault:

- the replacement of a constraint $x < m_1$ by $x < m_1'$ such that $m_1' > m_1$ and $m_1' \in [0, C_x]$.
- the replacement of a constraint $x = m_2$ by $x > m_2'$ such that $m_2' < m_2$.
- the replacement of a constraint $x = m_3$ by $x < m_3'$ such that $m_3' > m_3$ and $m_3' \in [0, C_x]$.
- the replacement of a constraint $x > m_4$ by $x > m_4'$ with $m_4' < m_4$.

Unlike constraint restriction, the constraint widening is considered as a fault for both inputs and outputs. Fig. 10 and Fig. 11 show two faulty implementations of the specification given in Fig. 1. The implementation in Fig. 10 enlarges the time constraint of the transition from l_1 to l_0 on input $?In$. Contrary to the specification, this implementation accepts the input $?In$ even when the value of clock x is between 1 and 2 and the value of clock y is less than 2. However, the implementation in Fig. 11 enlarges the time constraint of the transition from l_1 to l_1 on output $!Out$. This implementation can respond with the output $!Out$ even when the value of clock x is greater than 2.

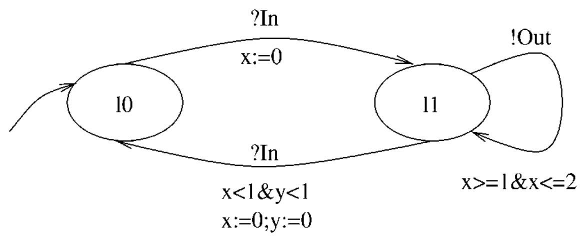

Fig. 8. Restriction of constraints for inputs.

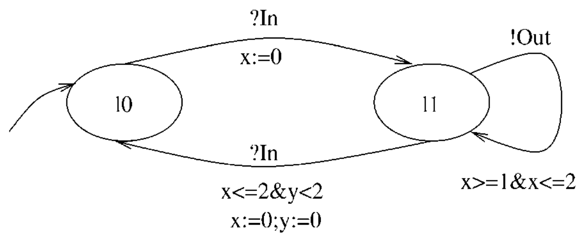

Fig. 10. Widening of constraints for inputs.

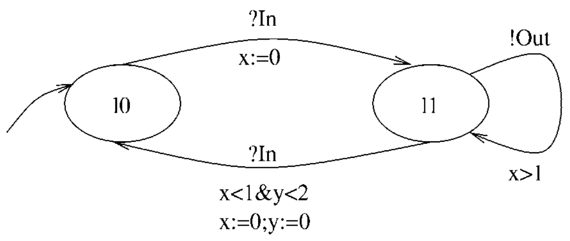

Fig. 11. Widening of constraints for outputs.

The widening of a time constraint increases the number of states and/or transitions in the region graph of the implementation compared with the region graph of the specification. In our example (Fig. 10 and Fig. 11), the number of states in the region graph remains unchanged but some extra transitions are added (e.g., the transition $(l_1, 1 < x = y < 2) \xrightarrow{?In} (l_0, x = y = 0)$ in the region graph of the implementation in Fig. 10).

6.3 Action and Transfer Faults

These faults are similar to the output and transfer faults in the Finite State Machine (FSM) Model [3], [7]. An implementation is said to have an output fault if in a state, it does not respond by an expected output. An implementation is said to have a transfer fault if in a state and on an input or output the implementation enters a state different from the expected one. Fig. 12 and Fig. 13 show two faulty implementations. The implementation in Fig. 12 has an output fault in location l_1. It does not respond with output $!Out$ after the application of the input sequences

$$\left\{\frac{1}{4}.?In.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}, \frac{1}{4}.?In.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}, \frac{1}{4}.\right.$$
$$?In.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}, \frac{1}{4}.?In.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}, \frac{1}{4}.$$
$$\left.?In.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}\right\}.$$

However, the implementation in Fig. 13 has a transfer fault in state $(l_1, x = 1\&y = 5/4)$. Indeed, after the sequence $\frac{1}{4}.?In.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.\frac{1}{4}.!Out$ the implementation enters the state $(l_0, x = 1\&y = 5/4)$ instead of the state $(l_1, x = 1\&y = 5/4)$.

Once the possible faults in an implementation of a TIOA are presented and discussed, we present now a method to generate timed test cases able to detect all these faults.

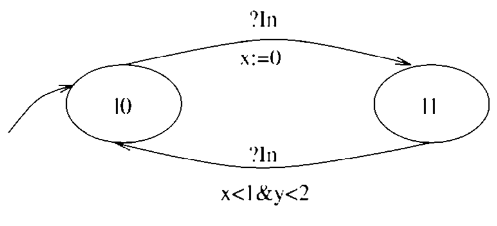

Fig. 12. An output fault.

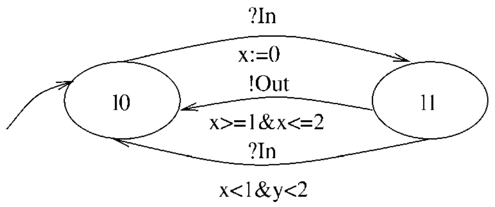

Fig. 13. A transfer fault.

7 Timed Test Cases Generation

Our approach for generating timed test cases introduced in [14] consists of three main steps:

- the sampling of region graph,
- the transformation of the resulting subautomaton into a nondeterministic FSM, and
- the adaptation of the generalized Wp-method [23].

7.1 Sampling Region Graph

Sampling has been introduced in [25] for the verification of real-time systems. Here, we use sampling to generate test cases for real-time systems. By sampling the region graph, we aim at deriving a subautomaton easily testable, called *Grid Automaton (GA)*. This automaton is then transformed into a finite state machine in order to reuse the existing test case generation methods. The idea behind the construction of the grid automaton is to represent each clock region with a finite set of clock valuations, referred to as the *representatives* of the clock region. The coordinates of each representative are defined from the grid points with granularity $\frac{1}{k}$ [25].

Definition 7.1: Set of Grids. *Let $k \in \mathbf{N}^{>0}$, we define the set of grids with granularity $\frac{1}{k}$ to be the set $N_k = \{\frac{m}{k} | m \in N\}$. We extend this notion in the usual way to any vector of n elements to define the grid points with granularity $\frac{1}{k}$ as the set $N_k^n = \{(r_1, r_2, ..., r_n) | 1 \leq i \leq n, r_i \in N_k\}$.*

The set of representatives of each clock region is determined from the set of grid points. The properties of sampling are studied in [25]. In particular, it is proven that for each clock region of a n-clock TIOA, there exists a set of its representatives in the grid points with granularity at most $\frac{1}{n+1}$. But, in order to represent all clock regions reachable by delay transitions, we have to use grid points with granularity at most $\frac{1}{n+2}$ if the number of clocks is greater than 2 and $\frac{1}{2}$ otherwise. In other words, the use of $\frac{1}{n+2}$ instead of $\frac{1}{n+1}$ is justified by the fact that we want to match every delay transition in the region graph by a transition on $\frac{1}{n+2}$ in the grid automaton so that the relationship between clocks be kept. When the number of clocks is 1, there will be no relationship between clocks to keep. So, the granularity $\frac{1}{2}$ is sufficient to sample the region graph and to match every delay transition in it by a transition on $\frac{1}{2}$ in the grid automata.

Notice that the granularity of grid points constitutes the steps by which the clocks are authorized to pass from one clock region to another one, thereby allowing the automaton to make transitions from one location to another. The construction of the grid automaton leads to the explicit extension of the alphabet of the automaton with delay

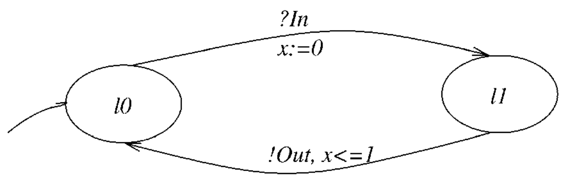

Fig. 14. A very simple TIOA.

actions of value $\frac{1}{n+2}$ (or $\frac{1}{2}$ if $n = 1$). In the rest of the text, we will use the symbol g to refer to the granularity of sampling. However in the examples, we instantiate g according to the number of clocks.

Since the construction of the region graph and our sampling algorithm (see later) are exponential, we will use the simple automaton of Fig. 14, instead of Fig. 1, to illustrate the different steps of our approach. The automaton has one clock, x. Therefore, the grid points we calculate are of granularity $\frac{1}{2}$. So, the set of representatives is $\{(0), (\frac{1}{2}), (1), (\infty)\}$. If we consider the TIOA of Fig. 1, the granularity will be $\frac{1}{4}$ and the set of representatives will be large. This does not mean that we can not apply our approach but it becomes difficult to draw the corresponding grid automaton.

To derive the grid automaton of Fig. 15, we proceed in many steps. Given an n-clock TIOA, we first derive the maximal granularity we can use (if $n = 1$ then the granularity is $\frac{1}{2}$ else the granularity is $\frac{1}{n+2}$). In a second step, we create the initial state formed with the initial location of the TIOA and a valuation that sets all clocks to zero. In a third step, we create all states reachable from the initial state with repetitive $\frac{1}{n+2}$ (or $\frac{1}{2}$ if $n = 1$) delay transitions. Then, for each state (l, v), we create a transition $((l, v), \{!, ?\}a, (l', [R := 0]v))$ for each transition $(l, l', \{!, ?\}a, R, G)$ in the TIOA such that v satisfies G. Afterwards, we repeat the same process starting with state $(l', [R := 0]v)$. The complete algorithm is shown in Fig. 16.

Except for the granularity, our algorithm constructs a grid automaton similar to the one obtained in [32] by the definition of the grid automaton for a given TIOA. The complexity of our algorithm is exponential in terms of the number of clocks and the integer constants used in the TIOA. This is inherent for any verification or testing method based on the region graph.

7.2 Transformation of Grid Automata into Nondeterministic FSM

In testing real-time systems, the controllability of the time at which an output is produced is difficult or impossible without time stamp instrumentation. To avoid this problem, we will not focus on which exact time an output is produced but we ensure if it occurs in the allowable time interval or not. This is guaranteed by the use of $\leq$, $=$, $>$, and $\geq$ in output constraints in the specification and the transformation of the grid automaton into an equivalent *Nondeterministic Timed FSM (NTFSM)*.

Definition 7.2: Nondeterministic Timed FSM. *A Nondeterministic Timed FSM (NTFSM) is a tuple $M = (S, s_0, I, O, T)$, where:*

- *S is a finite set of states.*
- *$s_0 \in S$ is the initial state.*
- *I is a finite set of inputs.*
- *O is a finite set of outputs.*
- *T is a finite set of transitions. Each transition is denoted by $s_i \xrightarrow{i|o} s_j$, where i is an input, o is an output, and s_i and s_j are, respectively, the source and the target states.*

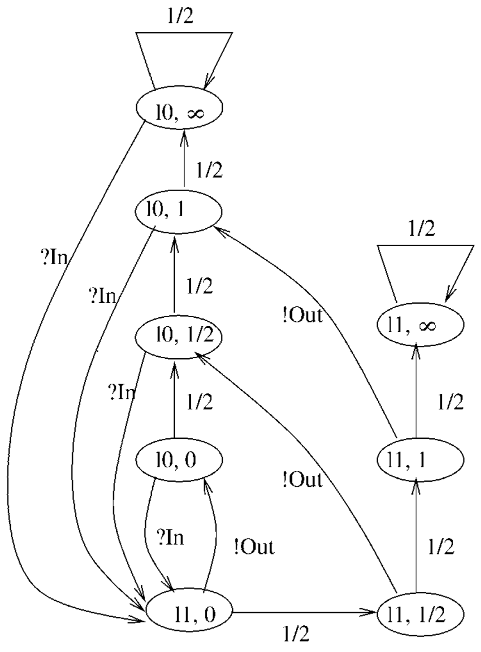

Fig. 15. The grid automaton of the TIOA given in Fig. 14.

The nondeterminism is mainly due to the delay transitions. In this transformation, the action corresponding to the elapsing of time is interpreted as an input action. Following this interpretation, during testing, the testers (Upper and Lower Testers) will check regularly (each g time units depending on the granularity of sampling g) the implementation output queue looking for an output. By choosing a suitable granularity, we test the outputs of the IUT at the bounds of their constraints and at least one point within the allowable time interval.

The transformation of the grid automaton into an NTFSM is based on the two schemes shown in Fig. 17. This transformation is very simple. It consists of coupling each output with the input or the delay preceding it in the GA. If the GA has a transition on input $?a$ from a state s_i to a state s_j followed by a transition on output $!b$ from the state s_j to a state s_k, we create two transitions

$$s_i \xrightarrow{a|b} s_k \text{ and } s_i \xrightarrow{a|-} s_j$$

in the NTFSM. However, if the GA has a transition on input $?a$ from a state s_i to a state s_j followed by no transition on output from the state s_j, we create only one transition $s_i \xrightarrow{a|-} s_j$ in the NTFSM. We note that a delay transition is handled as a transition on an input. It is clear that this transformation preserves the expressed behavior and one can easily prove the equivalence between the grid automaton and the NTFSM. Moreover, the NTFSM is

```
INPUT : - A $k$-clock TIOA $A = (\Sigma, L, l^0, C, T)$.
OUTPUT: - A grid automaton (GA).
STEP0: Compute the granularity to use:
    g $\leftarrow$ if $k = 1$ then $\frac{1}{2}$ else $\frac{1}{k+2}$.
STEP1: Initialize the sets to use:
    $R_S \leftarrow \{(l_0, v_0)\}$ (Reachable States)
    $H_S \leftarrow \emptyset$ (Handled States)
STEP2: Construct the grid automaton:
    While $R_S \backslash H_S \neq \emptyset$ do
      Get a state $s = (l, v)$ from $R_S \backslash H_S$
      $H_S \leftarrow H_S \cup \{s\}$
      For each $t = (l, l', \{!, ?\}a, R, G) \in T$ do
        If $v \models G$ then Add $((l, v), \{!, ?\}a, (l', [R := 0]v))$ to the set of transitions of GA
        if it does not exist
          $R_S \leftarrow R_S \cup \{(l', [R := 0]v)\}$ if it does not exist in $R_S$
        EndIf
      EndFor
      Add $((l, v), g, (l, (v + g)))$ to the set of transitions of GA if it does not exist
      (Notice that for each $x \in C$, if $v(x) = C_x$ then $v(x) + g = \infty$)
      $R_S \leftarrow R_S \cup \{(l, (v + g))\}$ if it does not exist in $R_S$
    EndWhile
```

Fig. 16. The sampling algorithm.

observable which means that the tester is able to determine the state where the IUT will be after the execution of each transition. As an example, the NTFSM corresponding to the grid automaton of Fig. 15 is shown in Fig. 18.

In the NTFSM model, we distinguish between two categories of states:

- States that are connected only by delay transitions: these states have the same first component and an incoming g-delay transition. They cannot be distinguished by input sequences because they are substates of the same super-state. However, the semantics of time distinguishes these states if no clock has been reset to zero during the last g-unit of time. The semantics of time guarantees that by letting the time elapses with g-unit, we change the state of the system. Furthermore, the model we use ensures whether or not a clock has been reset to zero. Notice that the set of time connected states, denoted by S_T, is a set of subsets of S.
- States that are identified by input sequences as in the FSM model. Contrary to S_T, the set of input identified states, denoted by S_I, is a subset of S.

As an example, the states $\{(l_0, 0), (l_1, 0)\}$ in the Fig. 18 are *input identified* while the set of states

$$\{\{(l_0,0),(l_0,\frac{1}{2}),(l_0,1),(l_0,\infty)\},\{(l_1,0),(l_1,\frac{1}{2}),(l_1,1),(l_1,\infty)\}\}$$

are *time connected*.

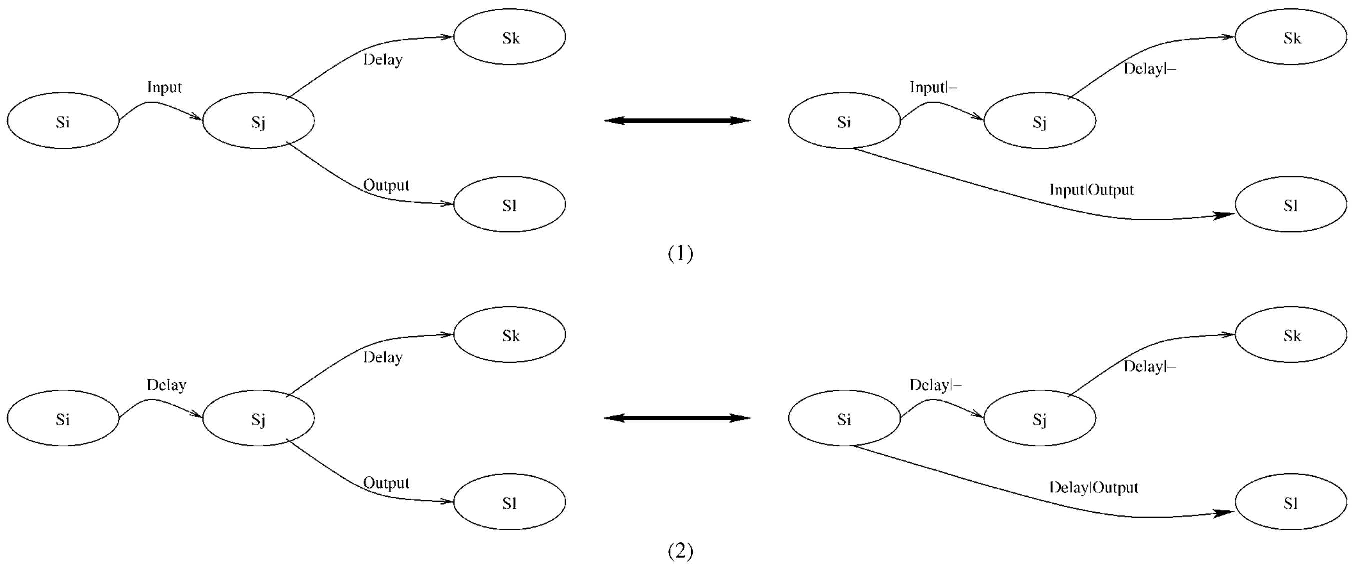

Fig. 17. The basic transformation schemes.

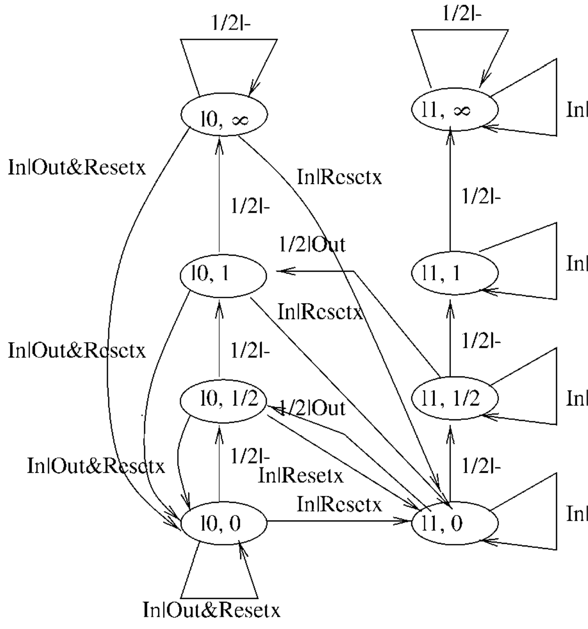

Fig. 18. The NTFSM corresponding to the grid automaton of Fig. 15.

7.3 Timed Wp-Method

As mentioned previously, by constructing the grid automaton we explicitly extend the alphabet of the TIOA with the delay action g in a way that each state has a transition labeled with this action and can be completely specified for the other input actions. Furthermore, we use our test architecture to make explicit the reset to zero operation on clocks. Therefore, we can apply the state characterization technique to generate timed test cases for the control part of our test model. The state characterization methods generate test cases that are conceptually applied in two phases. The first phase checks if all the states defined in the specification are identifiable in the implementation. The second phase checks if all transitions defined in the specification are implemented correctly. In general, the second phase does not check for the transitions which are already covered by test cases in the first phase.

For the generation of timed test cases, we have adapted the generalized Wp-method [23]. Our algorithm uses many sets which are defined as follows:

Definition 7.3: State Cover. *A set $Q \subseteq I_A^*$ is a state cover of an NTFSM A if, for every state s_i, Q contains an input sequence that brings the machine A from the initial state s_0 to s_i.*

Definition 7.4: Transition Cover. *A set $P \subseteq I_A^*$ is a transition cover of an NTFSM A if, for every transition $s_i \xrightarrow{a|b}_A s_j$, P contains input sequences α and $\alpha.a$ that bring the machine A from the initial state s_0 to s_i and s_j, respectively.*

Definition 7.5: Delay Sequence Set. *A set $D \subseteq (\{g\} \cup I_A)^*$ is a delay sequence set of an NTFSM A if, for every transition $s_i \xrightarrow{g|b}_A s_j$, D contains an input sequence α and $\alpha.g$ that bring the machine A from the initial state s_0 to s_i and s_j, respectively.*

Definition 7.6: Characterization Set. *A set $W \subseteq I_A^*$ is a characterization set for an NTFSM iff $\forall s_i, s_j \in S_I, i \neq j \Rightarrow \exists \sigma$ such that: $\sigma \in Traces(s_i)$ xor $\sigma \in Traces(s_j)$, and $\sigma \rceil I_A \in W$, where $\sigma \rceil I_A$ denotes the projection of σ on I_A obtained by removing the output actions from σ. With the completeness assumption, this means $(\sigma \rceil I_A)$ applied to s_i and s_j will yield different output traces.*

Definition 7.7: Prefix Set. *Let V be a set of input sequences. We define the prefix set of V as follows:*

$$Pref(V) = \{t_1 \mid t_1 \neq \varepsilon \wedge t_2 \in I^* \& t_1.t_2 \in V\}.$$

Definition 7.8: State Identification Set. *Given an NTFSM and a characterization set W, $\{W_0, W_1, ..., W_{n-1}\}$ is said to be a set of identification sets if, for $i = 0, 1, ..., n-1, W_i$ is a minimal set such that: $W_i \subseteq Pref(W)$ and for $j = 0, 1, ..., n-1, j \neq i \Rightarrow \exists \sigma \in Traces(s_i)$ xor $Traces(s_j)$ and $\sigma \rceil I_A^* \in W_i$.*

Fig. 19 shows our algorithm for the generation of a timed test suite. It consists of six steps. After transforming the TIOA into NTFSM (step 0 and step 1), it constructs in the following steps (step 2, step 3, and step 4) the sets S_T, S_I, Q, W, and $\{W_0, W_1, ..., W_{n-1}\}$ as defined before. Then, the algorithm derives the transitions cover set $P = Q.(I \cup \{\varepsilon\})$ and the set R. R denotes all transitions of NTFSM except those belonging to Q (i.e., $R = P \backslash Q$). Finally (in step 6), the algorithm generates the timed test suite $\prod$ which consists of two subsets $\prod 1$ and $\prod 2$ which are directly derived. $\prod 1$ consists of the concatenation of the sets Q and W. However, $\prod 2$ is derived by concatenating each sequence $(x \rceil I) \in R$ with the state identification W_i of each state s_i reachable from the initial state of the NTFSM by the input sequence $x \rceil I$. It suffices to have the set of all states reachable by each input sequence $x \rceil I$.

The algorithm generates test cases to check the conformance of an implementation to the specification given as TIOA. Since the grid automaton is a subautomaton of the region graph, the generated test cases ensures the correct behavior of the grid automaton in the implementation but do not cover the entire region graph. However, with the set of hypotheses we use, we prove that the test cases generated by our algorithm detect all the faults discussed in Section 6. In order to reduce the set of hypotheses, we have to use a very small granularity for the construction of the grid automaton. In fact, if we use a fine granularity, we will reduce the set of hypotheses and increase the fault coverage. However, the use of a fine granularity leads to a very large number of test cases and makes the method impracticable. This is due to the complexity of the sampling algorithm, which is exponential on the number of clocks and constants used in the time constraints of the TIOA. If we consider again the TIOA of Fig. 14 and use the granularity $\frac{1}{4}$, we will get an NTFSM with at least 12 states. Consequently, timed Wp-method will generate more than 40 test cases. The number of test cases would increase much faster if the TIOA contains more than one clock. For all the previous reasons, we used an appropriate granularity in order to come out with a practical timed test cases generation method.

The timed test cases generated by the algorithm for the TIOA in Fig. 14 are given in Fig. 20.

```
INPUTs:
  - A specification: A $k$-clock TIOA $A$ with an alphabet $\Sigma = I \cup O$,
  - An IUT $M$,
OUTPUTs:
  - A test suite for the IUT $M$.
STEP0: Construct a grid automaton from the TIOA
with an alphabet $\Sigma \cup \{g\}$ and $n$ states,
using the algorithm given in section 7.1,
STEP1: Transform the resulting grid automaton into
an NTFSM as shown in section 5,
STEP2: Determine the sets of timed connected and
input identified states $S_T$ and $S_I$ respectively.
STEP3: Construct the set cover $Q$ that includes the
set of delay sequences $D$ for $S_T$,
STEP4: Construct a characterization set $W$ and the
set of state identification set $\{W_0, W_1, ..., W_{n-1}\}$ of $S$.
STEP5: Construct two sets $P$ and $R$ such that:
$P = Q.(I \cup \{\varepsilon\})$ and $R = P \backslash Q$.
STEP6: Construct the timed test suite $\prod$ as follows:
$\prod = \prod 1 \cup \prod 2$, where
$\prod 1 = Q.W$,
$\prod 2 = \bigcup_{\beta_x} \{x]I\}.W_i$, where $\beta_x = S_0 \xrightarrow{x} S_i \& x]I \in R$
```

Fig. 19. The adapted generalized Wp algorithm.

8 Fault Coverage of Timed Wp-Method

In this section, we study the fault coverage of Timed Wp-method introduced in the previous section. More specifically, we discuss what are the faults, among those presented in Section 6, detected by the test cases generated with Timed Wp-method.

8.1 Reset of a Clock Fault

Timed Wp-method is based on a particular architecture consisting of the separation of the control and the clock parts (see Fig. 4). Using this architecture, the reset of a clock is *observable* at the interface of the control and clock parts via the internal signal *ResetClock*. Therefore, if an implementation does not reset a clock that is initialized to zero in the specification (e.g., Fig. 6), the tester will not observe the signal *ResetClock* expected at the interface of the control part. Consequently, the fault will be detected and the implementation will so be declared faulty. Similarly, if an implementation resets a clock that remains unchanged in the specification (e.g., Fig. 7), the tester will observe the nonexpected signal *ResetClock* at the interface of the control part. Consequently, the fault will be detected and the implementation will be declared faulty.

Proposition 8.1: Detection of Reset of a Clock Fault. *The reset of a clock faults are detected by timed Wp-method, because of the test architecture we use.*

Proof. The proof is simple. It is based on the assumption that the implementation uses the same number of clocks as the specification (Test hypothesis 3). □

Notice that the test architecture we use makes the test of reset of a clock easy. In fact, the reset of a clock is viewed as an internal action like the τ-transition in labeled transition systems (LTS) model (see [35], [34], for instance). Therefore, the testing of the reset of a clock without making it explicit is very difficult.

8.2 Time Constraint Restriction Fault

From Section 6.2.2, we have seen that the restriction of time constraints for outputs is not considered as a fault; so, it will not be studied in this section. However, the restriction of a time constraint of an input is considered a fault as discussed previously in Section 6.2.2. Each transition constraint gives rise to at least one clock region (system state) where the transition is executable. The restriction of a transition constraint leads to states or transitions missing in the implementation. Test cases are generated from the specification where all the system states are considered. Thus, they cover all the states and all the transitions of the specification. The following proposition states the relation between test cases and transitions constraints.

Proposition 8.2: Transitions Time Space. *For any input transition, there exists at least one test case that covers the time space of the transition.*

Proof. Let us consider an input transition $l \xrightarrow{?a,R,G} l'$ of the TIOA A. In testing, we deal with a correct specification; so, the transition is executable. Therefore, the region graph contains at least one state (l, Z) (Z is a clock region) where the transition is enabled (i.e., $\forall v \in Z, v \models G$). When we sample the region graph, we obtain at least one representative of state (l, Z). Therefore, there exists at least one test case that covers the time space of the input transition $l \xrightarrow{?a,R,G} l'$. □

Thus, if the implementation restricts a time constraint of a transition, some states and/or transitions will be missing in its region graph. This is true because of the test hypothesis 6. The missing states and/or transitions are covered by some test cases generated by the test cases generation algorithm. So, when the tester applies one of these test cases, the implementation will reject it, i.e., the IUT will remain in its current state, and the tester will not observe the expected output. Consequently, the tester logically concludes that a time constraint restriction fault has been introduced and the implementation will be declared faulty.

Proposition 8.3: Detection of Input Time Constraint Restriction Faults. *The test cases generated by the timed Wp-method detect input time constraint restriction faults.*

Proof. Let us consider an input transition $l_s \xrightarrow{?a,R,G} l'_s$ of the specification and its corresponding transition $l_i \xrightarrow{?a,R,G'} l'_i$ in the implementation such that G' is a restriction of G. Assume that the restriction concerns the constraint over a clock x such that we have $x < m$ in G and $x < m'$ in G' ($m' < m$). This modification of the bound of the constraint over clock x reduces the set of clock regions corresponding to G (noted $Regions(G)$). In other words, the set $Regions(G) \backslash Regions(G')$ is not empty ($Regions(G) \backslash Regions(G') \neq \emptyset$). Using test hypotheses 4 and 6, and Proposition 8.2, we conclude that the regions (the states) $Regions(G) \backslash Regions(G')$ are covered by some test cases. These test cases will be rejected by the implementation. Therefore, the fault is detected and the implementation is declared faulty. The cases of formulas $x \leq m$, $x = m$, $x > m$, and $x \geq m$ are proven similarly. □

The NTFSM States
$s_0 = (l_0, 0), s_1 = (l_0, \frac{1}{2}), s_2 = (l_0, 1), s_3 = (l_0, \infty)\}, s_4 = (l_1, 0), s_5 = (l_1, \frac{1}{2}), s_6 = (l_1, 1), s_7 = (l_1, \infty)$
The Timed Connected States
$\{\{(l_0, 0), (l_0, \frac{1}{2}), (l_0, 1), (l_0, \infty)\}, \{(l_1, 0), (l_1, \frac{1}{2}), (l_1, 1), (l_1, \infty)\}\}$
The Input Identified States
$\{(l_0, 0), (l_1, 0)\}$
The State Cover Q
$\{\varepsilon, \frac{1}{2}, \frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}.In, \frac{1}{2}.In.\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\}$
The Transition Cover P
$\{\varepsilon, \frac{1}{2}, \frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}.In, \frac{1}{2}.In.\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}, In, \frac{1}{2}\cdot\frac{1}{2}.In, \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}.In, \frac{1}{2}.In.In, \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}.In, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}.In, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}.In, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\}$
The Delay Sequence Set D
$\{\frac{1}{2}, \frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\}$
The R Set
$\{In, \frac{1}{2}\cdot\frac{1}{2}.In, \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}.In, \frac{1}{2}.In.In, \frac{1}{2}.In.\frac{1}{2}.In, \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}.In, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}.In, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\}$
The Characterization Set W
$\{\frac{1}{2}, In\}$
The State Identification Sets W_i
$W_0 = W_1 = W_2 = W_3 = W_6 = W_7 = \{In\}$ and $W_4 = W_5 = \{\frac{1}{2}\}$
Test Cases $\prod 1$:
$\{\frac{1}{2}, In, \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}\cdot\frac{1}{2}.In, \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}.In, \frac{1}{2}.In.In, \frac{1}{2}.In.\frac{1}{2}.In, \frac{1}{2}.In, \frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}.In, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}.In\}$
Test Cases $\prod 2$:
$\{In.In, In.\frac{1}{2}, \frac{1}{2}\cdot\frac{1}{2}.In.In, \frac{1}{2}\cdot\frac{1}{2}.In.\frac{1}{2}, \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}.In.In, \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}.In.\frac{1}{2}, \frac{1}{2}.In.In.In, \frac{1}{2}.In.In.\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}.In.In, \frac{1}{2}.In.\frac{1}{2}.In.\frac{1}{2}, \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}.In, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}.In.In, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}.In.In, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}.In.\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}.In.\frac{1}{2}, \frac{1}{2}.In.\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}.In\}$

Fig. 20. Timed test cases for the NTFSM of Fig. 18.

8.3 Time Constraint Widening Faults

The TIOA used as basis for the specification of the system has the property that all irrelevant values of a clock (i.e., the values greater than the greatest constant used in constraints over that clock) are represented by the constant ∞. So, when we sample the region graph, we obtain at least one state of the grid automaton in which the value of the clock is ∞. This state serves to verify whether or not the implementation enlarges a transition constraint, especially a constraint like $x < m$. The following corollary states the relation between a transition constraint widening and the grid automaton.

Proposition 8.4: Transition Constraint Widening. *For each transition $l \overset{\{?,!\}a,R,G}{\longrightarrow} l'$ in the TIOA, the grid automaton possibly contains a state (l, v) such that $v \not\models G$.*

Proof. Let us consider a transition $l \overset{\{?,!\}a,R,G}{\longrightarrow} l'$ in the specification. The constraint G gives rise to at least one clock region (system state) where the transition is executable. However, inorder to verify the existence of a reachable state (l, v) such that $v \not\models G$ we study the following cases.

- If the constraint G contains a formula $x < m$ (respectively, $x \leq m$ or $x = m$), there exists at least one state in the grid automaton where $x = m$ (respectively, $x > m$). This is true because each state in the grid automata has an outgoing delay transition labeled with the granularity of sampling.
- If the constraint G contains a formula $x \geq m$ or $x > m$, the region graph may contain a state (l, Z) where G is not satisfied. But, when such state exists, the grid automaton will have a state (l, v) where the value of clock x is less than m. The existence of states (l, Z) in the region graph depends on the constraints of the transitions preceeding $l \overset{\{?,!\}a,R,G}{\longrightarrow} l'$ in the specification. □

Now, since we assume the implementation does never reject an input (see test hypotheses in Section 3), we can easily see that timed Wp-method tests time constraint widening. For outputs, the constraint widening faults are detected by waiting the expected output within the allowed time interval. If the tester does not observe the output, it concludes that the implementation enlarges the constraint of that output, or does not respond with that output. For inputs, the tester applies them in states where their time constraints are not satisfied, to verify if they will be accepted. If it is the case, the implementation will be declared faulty.

Proposition 8.5: Detection of Time Constraint Widening Faults. *The test cases generated by Timed Wp-method detect all time constraint widening faults in an implementation.*

Proof. For each input transition $l \overset{?a,R,G}{\longrightarrow} l'$, there is a state (l, v) in the grid automaton such that $v \not\models G$. The implementation is completely specified; so, the state (l, v) has a delay transition and another transition for each input of the TIOA. Moreover, hypotheses 3, 4, and 6 guarantee that the number of states in the IUT is no greater than that of the specification. So, since timed Wp-method covers all states and all transitions in the grid automaton, it generates a test case that tests the time constraint widening of the transition $l \overset{?a,R,G}{\longrightarrow} l'$. □

8.4 Output Faults

These faults can be detected easily by visiting all the transitions in the TIOA.

Proposition 8.6: Detection of Output Faults. *Timed Wp-method guarantees the detection of output faults.*

Proof. Timed Wp-method covers all the states and transitions of the NTFSM. Therefore, it detects output faults.□

8.5 Transfer Faults

The transfer faults are detected by any method based on a state identification facility [7], [18], [30]. It suffices to reach a state, apply the input, observe the expected output, and then apply the identification sequence of the target state to verify whether it is the correct one.

Like W-method [7] and Wp-method [18], the Timed Wp-method is based on state characterization technique. So, it checks the target state of each transition of the system. This means that timed Wp-method detects the transfer faults.

Proposition 8.7: Detection of Transfer Faults. *The test cases generated by the Timed Wp-method detect all the transfer faults in an implementation.*

Proof. The test suite generated by Timed Wp-method consists of two parts: $Q.W$ and $(P\backslash Q).\{W_i\}$. Let us consider a specification transition $t : l_s \overset{\{?,!\}a,R,G}{\longrightarrow} l'_s$, and assume that the corresponding transition,

$$t' : l_i \overset{\{?,!\}a,R,G}{\longrightarrow} l'_i,$$

in the implementation contains a transfer fault. Since the sets of clocks to be reset with t and t' are the same (i.e., R), the target state, (l'_i, v), of t' will be equivalent to a state (l, v) in the specification (because of hypotheses 3, 4, and 6). Suppose that the expected state is (l''_i, v), which is equivalent to the state (l'_s, v) in the specification, and the identification set of (l'_s, v) is W_j. W and W_j distinguish between states (l'_s, v) and (l, v). So, there exists a test case in $Q.W$ or $(P\backslash Q).W_j$ that will detect the fault. □

9 Related Work

Contrary to the (untimed) finite state models (FSM, LTS, EFSM), test cases generation and testing of timed systems is still a new research area which is being investigated by different teams with different backgrounds. In this section, we summarize some of these works [27], [24], [8], [32], [9], [19], [17], [20], [21], [28] and compare them to our approach.

In [24], Liu proposes a testing methodology to generate test cases from an FSM with timers and counters. The author adapted Wp-Method [18] as follows: The author first draws two FSMs to represent respectively the behavior of timer and counter. Then, the three FSMs are combined into one and Wp-method is applied on the resulting FSM. Compared to our approach, this methodology does not deal with a general specification of real-time systems. Moreover, the author makes stronger assumptions about the IUT in order to increase the fault coverage of the method.

In [27], the authors propose a tool for the generation of test cases from specifications given as temporal logic formulas extended with time measures. The time domain is discretized into integer values and test cases are generated on the basis of what is called *Histories*. Compared to our approach, this methodology is restrictive in the sense that the logic formulas used as basis for test cases generation uses only one clock. Moreover, the generated test cases cover only integer values of time. Therefore, the fault coverage of the method is limited.

In [8], the authors introduce a framework for testing the constraints in timed systems. Test cases are derived from a specification described in the form of a constraint graph (CG). This method differs from our approach at least in the following three points. First, the test cases are generated based on the satisfaction of some test criteria for real-time systems, like EFSM-based testing. Second, the constraint graph is not general since it is restricted to the description of a minimum and a maximum allowable delay between input/output events in the execution of a system, following Taylor's and Dasarathy's [33], [10] classification of timing requirements. Finally, the test cases generated by the proposed approach do not cover all the potential faults in an implementation of the CG.

In [32], the authors present a theoretical framework for testing timed automaton. This is the first real approach for the generation of test cases from timed finite state models using methods developed for untimed models. In fact, the W-method [7] is used for the generation of test cases. The approach proposed in [32] is based on a previous work by Cerans [5] for the reduction of the test of a region automaton into the test of a subautomaton, referred to also as a *Grid Automaton*. Even if this methodology and our approach are generally similar and use the same model (i.e., TIOA), the TIOA model used in [32] is restrictive in the sense that outputs can occur only on integer values of clocks. Moreover, the authors themselves claim that their approach has no practical value in its original form since the number of test cases generated with their method is astronomically large. Indeed, to generate test cases from a TIOA, the authors use a granularity of 2^{-n}, where n is at least equal to the number of clock regions of the TIOA resulting from the composition of the TIOA of the specification and that of the implementation.

In [9], the authors propose a technique to generate timed test cases from the model of timed transition systems (TTSs). A TTS is a couple consisting of an initial state and a set of transition rules. Each transition must take place between a lower and an upper time bounds. To generate test cases, the authors first transform the TTS into a labeled transition system containing all the observable traces of the system and, then, apply an adapted version of W-method [7] on the resulting automata. The proposed approach has two drawbacks. First, the authors use a discrete time model that does not allow the specification of a system with clocks assuming continuous values. Second, the number of generated test cases may be very large if the implementation has more states than the specification.

In [20], the authors present an approach and an architecture to test real-time protocols specified by timed input output automata with discrete time domain (an extension to the continuous time domain is presented in [21]). The proposed approach consists of two steps. First, the timed specification is transformed into an equivalent untimed one using the well-known timer operations $Set(T, d)$ and $Exp(T)$. Then, the Wp-method [18] is applied on the resulting automata. The test architecture presented in this work represents a refinement to the one introduced in [13]. The main advantage of the proposed work is that it avoids the state explosion problem inherent to the use of the region graph. However, the major drawback of the approach is that it may introduce undesirable nondeterminism during testing and may produce nonexecutable test cases.

In [19], the authors propose a method to generate a test sequence from a timed input output automata. Each input/output action must be executed at an adequate timing, which satisfies all timing constraints in the test sequence. However, since the implementation outputs are uncontrollable, a tester cannot designate the timing of their occurrence in advance. The authors define two types of executable test sequences, must-traceable and may-traceable, and present an algorithm to decide if a given test sequence is executable. The authors also present in their work a test generation technique using UIOv-method [37]. The major problem of the proposed approach is that it may take a long time to terminate.

In [17], the authors present a method for testing real-time protocols with multiple conflicting timers. Here, the authors assume that each input/output exchange takes a certain time to realize and timers can be started or stopped in arbitrary transitions. The proposed method generates an executable test sequence after many transformations. However, the method does not guarantee a complete fault coverage.

In [28], the authors presents a test generation technique for a restricted class of timed automata, called Event Recording Automata (ERA). An ERA is a timed automata in which each action a is associated with a clock x_a such that whenever a transition is executed on action a, the clock x_a is reset to zero. To generate test cases, the authors extend the Hennessy test theory [11] in order to take into account the timing aspect of real-time systems. The test cases selection is not based on the region graph but on a equivalence class graph. This graph is a coarse partition of the system time space. The test generation method proposed in [28] consists of two steps. First, a class equivalence graph is constructed for the ERA. Then, some test cases satisfying Hennessy criteria are selected using a symbolic reachability analysis. The main advantage of this approach is that it generates a small number of test cases. However, its main difficulty is that it does not ensure a complete fault coverage.

10 Conclusion

We introduced a method, called Timed Wp-method, for the derivation of test cases for real-time systems modeled as a TIOA, a variant of the Alur and Dill timed automata model. In order to generate test cases, we first discussed the model at the syntactic and semantic levels. Then, we studied a fault model for real-time systems and described a timed test architecture. For the generation of test cases, we made explicit the elapsing of time by sampling the region graph with a known granularity and obtained a reduction called the grid automaton. Furthermore, to ensure a good coverage of the potential faults in a timed system, we transformed the grid automaton into an observable nondeterministic finite state machine and we generated test cases from the resulting automaton using a generation technique based on state characterization sets. Finally, we proved that the test cases generated by our method detect all the faults of our fault model, provided that our test hypotheses are satisfied.

We have implemented our method and applied it to various examples with different sizes. The method generates successfully timed test cases for a large number of these examples. Even if the sampling algorithm is exponential on the number of clocks and constants used as bounds in time constraints, our approach remains scalable because most of the real-time systems we are aware of are specified with at most three clocks and small integer constants. However, we recognize that, in some cases, the number of test cases is large but we can optimize it in one of the following ways:

- We can use test purposes to generate test cases only for the critical parts we want to test.
- We can also use an optimization algorithm to delete any test case which is a prefix of another one.
- We can make a tradeoff between the fault coverage and the number of generated test cases. For instance, instead of choosing all the representatives of a clock region we can select only one representative. We can also use a large granularity for sampling the region graph of the system.

Finally, we would like to mention that the timed Wp-method can be used to derive timed test cases for a real time system specified as a set of communicating TIOA. This can be done in two steps. In the first step, the complete or a partial product of the system is computed by composing all the parts nedeed to be tested. In the second step, the timed Wp-method is applied on the resulting automaton.

Acknowledgments

This work was supported by NSERC grants for R. Dssouli and F. Khendek, and Bell Canada through its Bell University Laboratories R&D program.

References

[1] R. Alur and D. Dill, “Automata for Modeling Real-Time Systems,” *Proc. 17th Int'l Colloquium Automata, Languages, and Programming,* M. Paterson, ed., pp. 322–335, July 1990.

[2] R. Alur and D. Dill, “A Theory of Timed Automata,” *Theoretical Computer Science,* vol. 126, pp. 183–235, 1994.

[3] G.V. Bochmann, A. Das, R. Dssouli, M. Dubuc, A. Ghedamsi, and G. Luo, “Fault Model in Testing,” *Proc. Fourth IFIP TC6 Int'l Workshop Protocol Test System,* 1992.

[4] E. Brinksma, G. Scollo, and C. Steenbergen, “LOTOS Specification, Their Implementations and Their Tests,” *Proc. Protocol Specification, Testing, and Verification, VI,* B. Sarikaya and G. von Bochmann, eds., pp. 349–360, 1986.

[5] K. Cerans, “Decidability of Bisimulation Equivalences for Parallel Timer Processes,” *Proc. Fourth Int'l Workshop Computer Aided Verification,* G. von Bochmann and D.K. Probst, eds., pp. 302–315, 1992.

[6] O. Charles, “Application des Hypothésde Test á une Définition de la Couverture,” PhD thesis, Université Henri Poincaré-Nancy 1, 1997.

[7] T.S. Chow, "Testing Software Design Modeled by Finite State Machine," *IEEE Trans. Software Eng.*, vol. 4, no. 3, pp. 178–187, 1978.

[8] D. Clarke and I. Lee, "Automatic Generation of Tests for Timing Constraints from Requirements," *Proc. Third Int'l Workshop Object-Oriented Real-Time Dependable Systems,* Feb. 1997.

[9] R. Cardell-Oliver and T. Glover, "A Practical and Complete Algorithm for Testing Real-Time Systems," *Formal Techniques for Real-Time Fault Tolerant Systems,* 1998.

[10] B. Dasarathy, "Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them," *IEEE Trans. Software Eng.*, vol. 11, no. 1, pp. 80–86, Jan. 1985.

[11] R. De Nicola and M. Hennessy, "Testing Equivalences for Processes," *Theoretical Computer Science,* vol. 34, pp. 83–133, 1984.

[12] C. Daws and S. Yovine, "Reducing the Number of Clock Variables of Timed Automata," *Proc. 1996 IEEE Real-Time Systems Symp., (RTSS '96),* 1996.

[13] A. En-Nouaary, R. Dssouli, and A. Elqortobi, "Génération de Tests Temporisés," *Proc. Sixth Colloque Francophone de l'ingénierie des Protocoles,* 1997.

[14] A. En-Nouaary, R. Dssouli, F. Khendek, and A. Elqortobi, "Timed Test Cases Generation Based on State Characterisation Technique," *Proc. 19th IEEE Real-Time Systems Symp. (RTSS '98),* Dec. 1998.

[15] A. En-Nouaary, F. Khendek, and R. Dssouli, "Fault Coverage in Testing Real-Time Systems," *Proc. Sixth Int'l Conf. Real-Time Systems Computing Systems and Applications (RTCSA '99),* Dec. 1999.

[16] A. En-Nouaary, F. Khendek, and R. Dssouli, "Fault Coverage in Testing Real-Time Systems," Technical Report TR-1162, Dept. IRO, Univ. de Montréal Sept. 1999.

[17] M.A. Fecko, P.D. Amer, M.U. Uyar, and A.Y. Duale, "Test Generation in the Presence of Conflicting Timers," *TESTCOM* Aug./Sept. 2000.

[18] S. Fujiwara, G.V. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test Selection Based on Finite-State Models," *IEEE Trans. Software Eng.*, vol. 17, no. 6, pp. 591–603, 1991.

[19] T. Higashino, A. Nakata, K. Taniguchi, and A. Cavalli, "Generating Test Cases for a Timed I/O Automaton Model," *Proc. Int'l Workshop Testing Communicating Systems (IWTCS '99),* 1999.

[20] A. Khoumsi, M. Akalay, R. Dssouli, A. En-Nouaary, and L. Granger, "An Approach for Testing Real-Time Protocols," *TESTCOM,* Aug./Sept. 2000.

[21] A. Khoumsi, A. En-Nouaary, R. Dssouli, and M. Akalay, "A New Method for Testing Real-Time Systems," *RTCSA,* Dec. 2000.

[22] N.A. Lynch and H. Attiya, "Using Mappings to Prove Timing Properties," *Distributed Computing,* vol. 6, no. 2, pp. 121–139, 1992.

[23] G. Luo, G.V. Bochmann, and A. Petrenko, "Test Selection Based on Communicating Nondeterministic Finite-State Machines Using a Generalized Wp-Method," *IEEE Trans. Software Eng.*, vol. 20, no. 2, pp. 149–162, 1994.

[24] F. Liu, "Test Generation Based on an FSM Model with Timers and Counters," master thesis, Dept. dé Informatique et de Recherche Opérationnelle, Univ. de Montréal, 1993.

[25] K.G. Larsen and W. Yi, "Time Abstracted Bisimulation: Implicit Specification and Decidability," *Proc. Math. Foundations of Programming Semantics (MFPS 9),* Apr. 1993.

[26] P. Merlin and D.J. Farber, "Recoverability of Communication Protocols," *IEEE Trans. Comm. Protocols,* vol. 24, no. 9, 1976.

[27] D. Mandrioli, S. Morasca, and A. Morzenti, "Generating Test Cases for Real-Time Systems from Logic Specifications," *ACM Trans. Computer Systems,* vol. 13, no. 4, pp. 365–398, Nov. 1995.

[28] B. Nielsen and A. Skou, "Automated Test Generation from Timed Automata," *Proc. Workshop Tools and Algorithms for the Construction and Analysis of Systems,* Apr. 2001.

[29] X. Nicollin, J. Sifakis, and S. Yovine, "Compiling Real-Time Specifications into Extended Automata," *IEEE Trans. Software Eng.*, vol. 18, no. 9, pp. 794–804, Sept. 1992.

[30] K. Sabnani and A. Dahbura, "A Protocol Test Generation Procedure," *Computer Networks and ISDN Systems,* vol. 15, pp. 285–297, 1988.

[31] J. Springintveld and F. Vaandrager, "Minimizable Timed Automata," *Proc. Fourth Int'l School and Symp. Formal Techniques in Real Time and Fault Tolerant Systems,* B. Jonsson and J. Parrow, eds., 1996.

[32] J. Springintveld, F. Vaadranger, and P. Dargenio, "Testing Timed Automata," *Theoretical Computer Science,* vol. 254, pp. 225–257, 2001.

[33] B. Taylor, "Introducing Real-Time Constraints into Requirements and High Level Design of Operating Systems," *Proc. Nat'l Telecomm. Conf.*, vol. 1, pp. 18.5.1–18.5.5, 1980.

[34] J. Tretmans, "A Formal Approach to Conformance Testing," PhD Thesis, Univ. of Twente, Aug. 1992.

[35] J. Tretmans, "Testing Labelled Transition Systems with Inputs and Outputs," Technical Report 95-26, University of Twente, Aug. 1995.

[36] S.T. Vuong and J. Curgus, "On Test Coverage Metrics for Communication Protocols," *Proc. Fourth Int'l Workshop Protocol Test System (IWPTS '91),* Oct. 1991.

[37] S.T. Vuong, W.W.L. Chan, and M.R. Ito, "The UIOv Method for Protocol Test Sequence Generation," *Proc. Second Int'l Workshop Protocol Test System,* 1989.

[38] E.J. Weyuker and S. Rapps, "Selecting Software Test Data Using Data Flow Information," *IEEE Trans. Software Eng.*, vol. 11, no. 4, 1985.

[39] M. Yao, "On the Development of Conformance Test Suites in View of Their Fault Coverage," PhD thesis, Univ. de Montréal, Dept. IRO, 1995.

Abdeslam En-Nouaary received the engineer degree in computer engineering from ENSIAS (École Nationale Supérieure d'Informatique et d'Analyse des Systémes), Rabat, Morocco, in 1996, and the Masters and PhD degrees in computer science, from the Universty of Montreal in 1998 and 2001, respectively. He is currently an assistant professor in the Electrical and Computer Engineering Department, Concordia University, Montreal. His research interests include software engineering, protocol engineering, and real-time systems.

Rachida Dssouli received the Doctorat d'Université degree in computer science from the Université Paul-Sabatier of Toulouse, France, in 1981, and the PhD degree in computer science in 1987, from the University of Montréal, Canada. She is a professor in the Department of Electrical and Computer Engineering (ECE) Concordia University. She has been a professor at the Université Mohamed 1er, Oujda, Morroco, from 1981 to 1989, assistant professor at the Université de Sherbrooke, from 1989 to 1991, and a full professor at the Université de Montréal until May 2001. She spent a sabbatical year at NORTEL (1995-1996). Her research area is in communication protocol engineering, requirements engineering, and multimedia applications. Ongoing projects include incremental specification and analysis of reactive systems based on scenario language, multimedia applications, conformance testing, design for testability, conformance testing based on FSM /EFSM, and timed automata. She has often served as a member on the committee of many workshops and conferences (IWPTS, IWTCS, FORTE, CFIP, MMNS, MMM, FIW, NOTERE). She organized or corganized several international workshops and conferences (IWPT '93, CFIP '93, FORTE '95, CFIP '96, NOTERE '98, SDL Forum '99, CPWCSE 2001).

Ferhat Khendek received the engineering degree in computer engineering, option software, from the University of Tizi-Ouzou, Algeria, in 1986, and the MSc and PhD degrees in computer science from the University of Montreal in 1989 and 1995, respectively. During his PhD studies, he held a research fellowship from the IBM Center for Advanced Studies in Toronto. He is currently an associate professor with the Electrical and Computer Engineering Department at Concordia University where he holds a research chair position in telecommunications software engineering. From 2001 to 2002, he was a visiting researcher at Ericsson Research Canada in Montréal. His research interests are mainly in the use of formal methods (SDL and MSC) and UML for the design and analysis of communication software and service engineering techniques.