A Guided Method for Testing Timed Input Output Automata

Abdeslam En-Nouaary and Rachida Dssouli

Department of Electrical and Computer Engineering
Concordia University, 1455 de Maisonneuve W., Montréal
Québec H3G 1M8, Canada
{ennouaar,dssouli}@ece.concordia.ca

Abstract. Real-time systems are those systems whose behaviors are time dependent. Reliability is one of the characteristics of such systems and testing is one of the techniques that can be used to ensure reliable real-time systems. This paper presents a method for testing real-time systems specified by Timed Input Output Automata (TIOA). Our method is based on the concept of test purposes. The use of test purposes helps reduce the number of test cases generated since an exhaustive testing of a TIOA causes the well-known state explosion problem. The approach we present in this paper consists of three main steps. First, a synchronous product of the specification and test purpose is computed. Then, a sub-automaton (called Grid Automata) representing a subset of the state space of this product is derived. Finally, test cases are generated from the resulting grid automata. The test cases generated by our method are executable and can easily be represented in TTCN (Tabular Tree Combined Notation).

Keywords: Real-Time Systems, Timed Input Output Automata, Testing, Test Purposes.

1 Introduction

Testing plays a key role in software life cycles. It consists of executing a physical implementation of a computer system with the intention of finding and discovering errors. This is done by submitting a set of test cases (also called test suite) to the implementation and observing its reactions. If the outputs of the implementation for a test case do not match those derived from the specification, the implementation is said faulty (i.e., a fault is detected). A test case is a sequence of input actions allowed by the environment. The test cases we apply to the implementation of a system are systematically generated from the formal specification of that system. A test cases generation algorithm should be practical in the sense that it must derive few test cases while ensuring good fault coverage. The term fault coverage refers to the ability of a test cases generation method to detect the potential faults in the implementation under test.

Over the last three decades, many algorithms have been developed for testing untimed specification models such as Finite State Machines (FSMs) and

D. Hogrefe and A. Wiles (Eds.): TestCom 2003, LNCS 2644, pp. 211–225, 2003.

Extended FSMs (EFSMs). However, testing real-time systems is still a new research field since researchers have started investigating the issue only at the mid-nineties. Even some algorithms have been devised for testing real-time systems (see for example [ENDKE98,SVD01,MMM95,CL97,COG98,KAD+00, KENDA00], [FAUD00,SPF01,KLC98,HNTC99,NS98,ENDK02,EN02,Hog01]), most of these methods suffer from the state space explosion problem and generate a great number of test cases. So, the necessity for the development of new techniques that are practical and with good fault coverage still exists.

In this paper, we present a framework for testing real-time systems using test purposes. A test purpose is a precise representation of the functionality to be tested. Thus, test purposes allow us to reduce the number of test cases generated and incrementally carry out the testing process. The formal model we use to describe both the specification and test purposes is *Timed Input Output Automaton (TIOA)* [AD94,NSY92,LA92]. To generate test cases from TIOA, our approach proceeds in three steps. First, a synchronous product of the specification and test purpose is computed. Then, an automaton representing a subset of the state space of this product is constructed. Finally, test cases are derived from the resulting grid automata.

The remainder of this paper is structured as follows. Section 2 introduces the TIOA model and the test purpose concept as well as the theoretical results needed for the rest of the paper. Section 3 presents our approach for timed test cases generation. Section 4 discusses the results and concludes the paper.

2 Backgrounds

This section presents the TIOA model and the test purpose concept as well as the theoretical ingredients we need for the subsequent sections. All these concepts and results are illustrated with simple examples so that the reader later later understands each step of our approach.

Definition 1. *Timed Input Output Automaton*
A 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. Each input action begins with "?".*
- O_A *is a finite set of output actions. Each output action 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*.*
- $T_A \subseteq L_A \times (I_A \cup O_A) \times \Phi(C_A) \times 2^{C_A} \times L_A$ *is the set of transitions.*

A transition in a TIOA, denoted by $l \xrightarrow{\{?,!\}a,G,\lambda}_A l'$, consists of a source location l, an input or output action $\{?,!\}a$, a clock guard G, which should hold to execute the transition, a set of clocks λ to be reset when the transition is fired, and a target location l'. We assume that the transitions are instantaneous and the clock guards over C_A are conjunctions of formulas of the form $x\ op\ m$, where: $x \in C_A$, $op \in \{<, \leq, =, >, \geq\}$ and m is a natural. Moreover, we suppose that

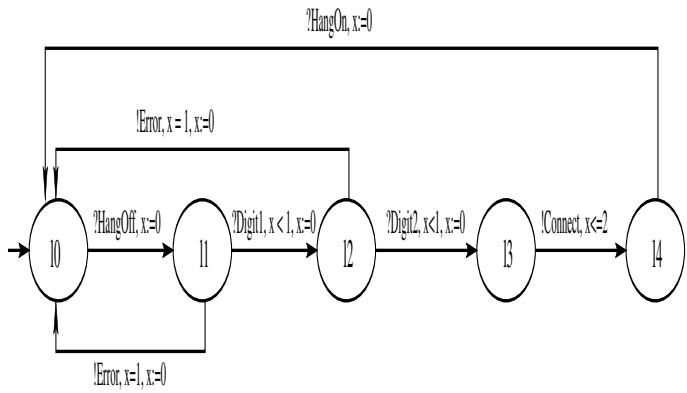

Fig. 1. An Example of TIOA.

each clock $x \in C_A$ has a bounded domain $[0, B_x] \cup \{\infty\}$ [SV96], where B_x is the largest integer constraint appearing in the constraints over x in the automaton. This means that each clock x is relevant only under the integer constant B_x, and all the values of x greater than B_x are represented by ∞.

Example 1. Figure 1 shows an TIOA with one clock. It is a specification of an hypothetical telephone system somewhat similar to that presented in [CL97]. The task of the telephone system is to issue an output *Connect* whenever a user hangs off (input $Hangoff$ on the Figure) and composes two digits ($Digit1$ and $Digit2$ on the Figure) forming the number to be called. After the connection is established, the user hangs on and the system goes back to its idle state and starts waiting for another connection request. The behavior of the system should respect the following time constraints. First, the user should type the first digit within 1 time-unit after having hanged off. Moreover, the amounts of time separating $Digit1$ and $Digit2$ must be no more than 1 time-unit. Finally, the system must respond with *Connect* within 2 time-units after the last digit has been typed. Whenever an input's time constraint is not respected by the user, the system times out, issues *Error* and goes back to its idle state.

The TIOA introduced so far is an abstract model because it doesn't explicit all the possible executions. Such executions, called the operational semantics, can informally stated as follows. The TIOA starts at its initial location with all clocks initialized to zero. Then, the values of clocks increase synchronously and measure the amount of time elapsed since the last initialization or reset. At any time, the TIOA can make a transition $l \xrightarrow{\{?,!\}a,G,\lambda} l'$ provided that the current location is l and the values of clocks satisfy the clock guard G. In this case, all the clocks in λ are reset and the TIOA changes its location to l'. To formalize the operational semantics of TIOA, we need the following definitions.

Definition 2. *Clock valuation*
Let $A = (I_A, O_A, L_A, l_A^0, C_A, T_A)$ *be a* $n-$*clocks TIOA (i.e., a TIOA with* n *clocks).*

- *A clock valuation of A (or over C_A) is an application $v : C_A \rightarrow [\mathbf{R} \cup \{\infty\}]^n$, which assigns a non-negative real number or ∞ to each clock $x \in C_A$. We represent a clock valuation by a vector $(v_{x_1}, v_{x_2}, .., v_{x_n})$ and denote the set of all clock valuations by $V(C_A)$.*
- *For any clock valuation $v \in V(C_A)$ and any non-negative real number d, $v + d$ is also a clock valuation that assigns the value $v(x) + d$ to each clock $x \in C_A$. $v + d$ is the clock valuation reached from v by letting time elapses by d time units.*
- *For any clock valuation $v \in V(C_A)$ and any subset of clocks $X \in C_A$, $[X := 0]v$ is also a clock valuation that assigns the value 0 to each clock $x \in X$ and agrees with v on the rest of clocks. $[X := 0]v$ is the clock valuation obtained from v by resetting clocks X.*
- *A clock valuation $v \in V(C_A)$ satisfies a clock guard G, denoted by $v \models G$, if and only if G holds under v.*

Definition 3. *States of TIOA*
Let $A = (I_A, O_A, L_A, l_A^0, C_A, T_A)$ be a TIOA.

- *A state of A is a pair (l, v) consisting of a location $l \in L_A$ and a clock valuation $v \in V(C_A)$.*
- *The initial state of A is the pair (l_A^0, v_0), where $v_0(x) = 0$ for each clock $x \in C_A$. We denote the set of states of A by $S(A)$.*

The operational semantics of a TIOA A is formally given by a timed labeled transition system, called the regions graph. The latter is constructed using the equivalence relation $\sim$ [AD94] on the set of clock valuations $V(C_A)$.

Definition 4. *Equivalence between Clock Valuations*
Let $A = (I_A, O_A, L_A, l_A^0, C_A, T_A)$ be a TIOA, and v and $v' \in V(C_A)$. We say v and v' are equivalent, written $v \sim v'$, iff:

- $\forall x \in C_A, \lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$
- $\forall x, y \in C_A$ *such that* $((v(x) \neq \infty) \wedge (v(y) \neq \infty)), (fract(v(x)) \leq fract(v(y)) \Leftrightarrow fract(v'(x)) \leq fract(v'(y)))$
- $\forall x \in C_A$ *such that* $v(x) \neq \infty, (fract(v(x)) = 0 \Leftrightarrow fract(v'(x)) = 0)$
  Here, $\lfloor t \rfloor$ and $fract(t)$ denote the integer and fractional parts of t respectively.

Definition 5. *Clock region*
Let $A = (I_A, O_A, L_A, l_A^0, C_A, T_A)$ be a TIOA. A clock region of A (or over C_A) is an equivalence class generated by the relation $\sim$ given in definition 4. We denote the clock region of a clock valuation v by $[v]$ and the set of all clock regions of A by $Reg(A)$.

Example 2. The set of regions for the TIOA of Figure 1 is given in Figure 2. For instance, the clock valuations $v_1 = \frac{1}{2}$ and $v_2 = \frac{1}{10}$ have the same behaviors when time progresses and so are equivalent (i.e., they belong to the same region). This means that if a state (l, v_2) accepts a trace then the state (l, v_1) also does.

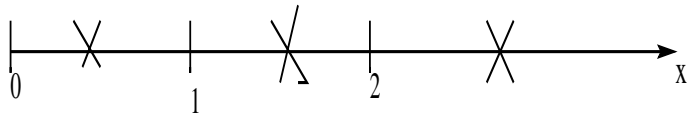

Fig. 2. An Example of Clock Regions.

Table 1. Number of Clock Regions in TIOA.

Formula	1-clock	2-clocks	3-clocks	4-clocks
[AD94]	8	128	3072	98304
[EEN98]	4	13	88	474

An upper bound of the number of regions in a $n-$clocks TIOA A has been given in [AD94]. However, the exact number of these regions is given in [EEN98]. Table 1 gives the number of regions resulting from applying both formulas to three TIOAs with different numbers of clocks having all the same domain $[0,1] \cup \{\infty\}$.

An important property of clock valuations is that each clock valuation can be represented by an equivalent one with all coordinates having the form $\frac{m}{n+1}$, where m is a non-negative integer and n is the number of clocks in TIOA. The following proposition formalizes this property.

Proposition 1. *Relationships between Clock Valuations*
Let $A = (I_A, O_A, L_A, l_A^0, C_A, T_A)$ be a $n-$clock TIOA.
For all $v \in V(C_A)$, there exists non-negative integers, $m_1, m_2, .., m_n$, and $v' \in V(C_A)$ such that $v' = (\frac{m_1}{n+1}, \frac{m_2}{n+1}, .., \frac{m_n}{n+1})$ and $v' \sim v$.

Proof. For the sake of space, the proof is given in the technical report of this paper.

The following proposition generalizes the result of proposition 1 to clock regions in a TIOA.

Proposition 2. *Characterization of Clock Regions*
Let $A = (I_A, O_A, L_A, l_A^0, C_A, T_A)$ be a $n-$clock TIOA.
For every $R \in Reg(A)$, there exists at least one clock valuation $v = (\frac{m_1}{n+1}, \frac{m_2}{n+1}, .., \frac{m_n}{n+1})$ ($m_1, m_2, .., m_n$ are non-negative integers) that characterizes R (i.e., $[v] = R$).

Proof. For the sake of space, the proof is given in the technical report of this paper.

Definition 6. Regions Graph
Let $A = (I_A, O_A, L_A, l_A^0, C_A, T_A)$ be a TIOA. The regions graph of A is an automaton $RG = (\Sigma_{RG}, S_{RG}, s_{RG}^0, T_{RG})$ where:

- $\Sigma_{RG} = I_A \cup O_A \cup \mathbf{R}^{>\mathbf{0}}$,
- $S_{RG} = \{\langle l, [v] \rangle \mid l \in L_A \wedge v \in V(C_A)\}$,

- $s^0_{RG} = \langle l^0_A, [v_0] \rangle$, *where* $v_0(x) = 0$ *for all* $x \in C_A$,
- *RG has a transition* $s \xrightarrow{\{?,!\}a} s'$, *from* $s = \langle l, [v] \rangle$ *to* $s' = \langle l', [v'] \rangle$ *on action* $\{?,!\}a$ *iff there is a transition* $l \xrightarrow{\{?,!\}a,G,\lambda}_A l'$ *such that* $v \models G$ *and* $v' = [\lambda := 0]v$.
- *RG has a delay transition* $s \xrightarrow{d} s'$, *from* $s = \langle l, [v] \rangle$ *to* $s' = \langle l, [v'] \rangle$ *on time increment* $d > 0$, *iff* $[v'] = [v + d]$.

Since the regions graph can be seen as a timed reachability analysis graph of TIOA, it is heavily used for the verification and testing of timed dependant systems. Moreover, it is clear from the definition 6 above that each state of the regions graph has a delay transition labeled with the symbol d. Here, the value of d is in the interval $]0, 1[$. The delay transitions on d greater than or equal to 1 are obtained using the following rule: if $s_1 \xrightarrow{d_1} s_2$ and $s_2 \xrightarrow{d_2} s_3$ then $s_1 \xrightarrow{d_1+d_2} s_3$.

As the symbol d in the definition 6 takes an infinite number of values between 0 and 1, we can use propositions 1 and 2 to instantiate the delay transitions with the same value and obtain a finite sub-automaton of the regions graph. This operation is termed *sampling the regions graph* and the resulting sub-automaton is called *Grid Automaton (GA)*[LY93,ENDKE98,SVD01,ENDK02]. The following proposition formalizes the result.

Proposition 3. *Sampling the Region Graph*
Let $A = (I_A, O_A, L_A, l^0_A, C_A, T_A)$ *be a* $n-$*clock TIOA.*
There exists a sub-automaton of the regions graph of A with all delay transitions labeled with the same delay $\frac{1}{n+1}$.

Proof. For the sake of space, the proof is given in the technical report of this paper.

After introducing the TIOA model and all theoretical results we need for subsequent sections, we now define the concept of test purpose [ISO91,Tre92,KLC98], [KC00,GHN93,BGRS01,SPF01] that plays a key role in our contribution.

Definition 7. Timed Test Purpose
A timed test purpose TP is an acyclic TIOA $(I_{TP}, O_{TP}, L_{TP}, l^0_{TP}, C_{TP}, T_{TP})$ *with a special set of accepting locations.*

Informally, the test purpose represents the property we want to verify. For a real-time system implementation, this property is a set of interactions with the environment as well as the time constraints of these interactions. For example, a test purpose may consist of checking whether a sequence of interactions is permitted by an implementation of a real-time system within a certain time interval. The accepting locations of a test purpose represent the locations where the test verdict should be "Pass".

Example 3. To illustrate the concept of test purpose, consider again the telephone system of Figure 1. Figure 3 shows an example of test purposes, which consists of checking whether the implementation accepts *Digit*1 within 1-time

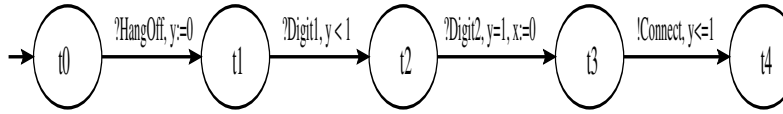

Fig. 3. An Example of Test Purpose.

unit and $Digit2$ 1-time unit after the user has hanged off, and responds with $Connect$ at the latest 1-time unit after $Digit2$ is dialed. Here, instead of generating test cases for the whole system, the user needs just some of them to verify whether or not the implementation permits his/her test purpose. The advantage of using test purposes is therefore the saving of time and money needed for testing implementations.

To test an implementation against a test purpose, the implementation must, at the same time, respect the specification and satisfy the test purpose. Therefore, we should compute a special composition [Kan95](called synchronous product in [KLC98]) of the specification and test purpose before generating test cases.

Definition 8. Synchronous Product
Let Let $A = (I_A, O_A, L_A, l_A^0, C_A, T_A)$ *be a specification and* $TP = (I_{TP}, O_{TP}, L_{TP}, l_{TP}^0, C_{TP}, T_{TP})$ *be a timed test purpose. The synchronous product of* A *and* TP *is the TIOA* $SP = (I_{SP}, O_{SP}, L_{SP}, l_{SP}^0, C_{SP}, T_{SP})$ *such that:*

- $I_{SP} = I_A \cup I_{TP}$ *and* $O_{SP} = O_A \cup O_{TP}$.
- $L_{SP} \subseteq L_A \times L_{TP}$.
- $l_{SP}^0 = (l_A^0, l_{TP}^0)$.
- $C_{SP} = C_A \cup C_{TP}$.
- L_{SP} *and* T_{SP} *are the smallest relations defined by the following two rules:*
  - $(l_1, l_2) \in L_{SP} \wedge l_1 \xrightarrow{\{?,!\}a,\lambda_1,G_1}_A l_1' \in T_A \wedge l_2 \xrightarrow{\{?,!\}a,\lambda_2,G_2}_A l_2' \notin T_{TP} \Longrightarrow (l_1', l_2) \in L_{SP} \wedge (l_1, l_2) \xrightarrow{\{?,!\}a,\lambda_1,G_1}_A (l_1', l_2) \in T_{SP}$.
  - $(l_1, l_2) \in L_{SP} \wedge l_1 \xrightarrow{\{?,!\}a,\lambda_1,G_1}_A l_1' \in T_A \wedge l_2 \xrightarrow{\{?,!\}a,\lambda_2,G_2}_A l_2' \in T_{TP} \Longrightarrow (l_1', l_2') \in L_{SP} \wedge (l_1, l_2) \xrightarrow{\{?,!\}a,\lambda_1 \cup \lambda_2,G_1 \& G_2}_A (l_1', l_2') \in T_{SP}$.

Example 4. Figure 4 shows the synchronous product of the specification in Figure 1 and the test purpose in Figure 3. As we can see from this figure, some executions of the specification are not allowed by the synchronous product because the time constraints of a transition in the specification and its corresponding one in the test purpose might not be simultaneously satisfied. For example, the execution $?HangOff.\frac{2}{3}.?Digit1.\frac{2}{3}.?Digit2.!Connect$ is allowed by the specification but not by the synchronous product.

3 Test Cases Generation

This section is devoted to test cases generation from a TIOA using the definitions and results of the previous section. Our approach is based on test purposes to

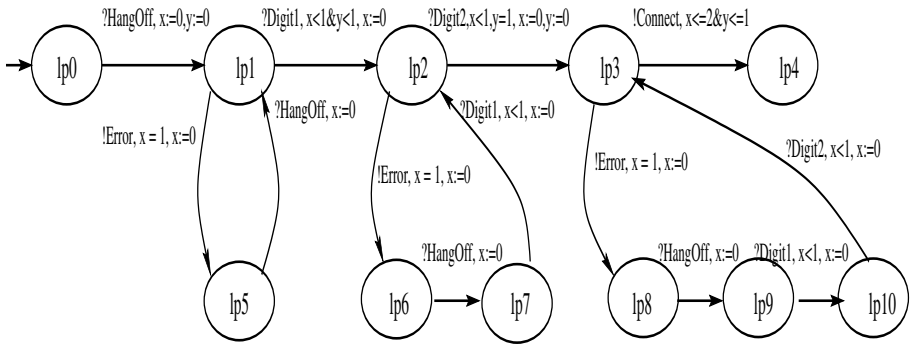

Fig. 4. Synchronous Product of Figures 1 and 3.

test only the critical parts of the system and therefore minimize the number of test cases generated. The proposed method consists of three main steps:

- The construction of a synchronous product.
- The sampling of the regions graph of the synchronous product.
- The traversal of the resulting sub-automaton.

In what follows, we will explain each of these steps and illustrate it using Figures 1 and 3.

3.1 Construction of a Synchronous Product

Since our approach is based on test purposes, the goal of this step is to construct a synchronous product of the specification of the system to be tested and the test purpose to be verified. This construction is based on the definition 8, which is transformed into the algorithm presented in Figure 5.

The algorithm takes a specification and a test purpose as inputs and returns their synchronous product as output. The synchronous product is a TIOA representing somehow an intersection between the TIOA of the specification and that of test purpose. This is needed because when we want to test an implementation using test purposes, we would like to check whether or not the implementation satisfies both the specification and test purpose. The test verdict is given based on the following three rules. If the implementation satisfies both the specification and test purpose, the verdict is "Pass". However, if the implementation respects the specification but does not satisfy the test purpose, the verdict is "Inconclusive". Finally, if the implementation does not respect the specification, the verdict is "Fail".

To construct the synchronous product, the algorithm first creates the initial location of the TIOA by concatenating the initial location of the specification and that of the test purpose. Then, the algorithm incrementally constructs the transitions of the synchronous product and adds the resulting states to the set of states. The transitions and states of the synchronous product are created according to the three rules stated in definition 8.

INPUT: - A specification's TIOA $S = (I_S, O_S, L_S, l_S^0, C_S, T_S)$.
- A test purpose TIOA $TP = (I_{TP}, O_{TP}, L_{TP}, l_{TP}^0, C_{TP}, T_{TP})$.
OUTPUT: - A synchronous product $SP = (I_{SP}, O_{SP}, L_{SP}, l_{SP}^0, C_{SP}, T_{SP})$.
$l_{SP}^0 \leftarrow (l_S^0, l_{TP}^0)$.
Add l_{SP}^0 to L_{SP}.
$C_{SP} \leftarrow (C_S \cup C_{TP})$.
$RL \leftarrow l_{SP}^0$ // the set of reachable locations.
$HL \leftarrow \emptyset$ // the set of handled locations.
While $(RL \backslash HL \neq \emptyset)$ do
Get a location $l = (l_1, l_2)$ from $RL \backslash HL$.
Add l to HL.
If $l_1 \xrightarrow{\{?,!\}a,G_1,\lambda_1}_S l_1' \in T_S$ and $l_2 \xrightarrow{\{?,!\}a,G_2,\lambda_2}_{TP} l_2' \notin T_{TP}$ then
Add (l_1', l_2) to RL.
Add $(l_1, l_2) \xrightarrow{\{?,!\}a,G_1,\lambda_1}_{SP} (l_1', l_2)$ to T_{SP}.
EndIf
If $l_1 \xrightarrow{\{?,!\}a,G_1,\lambda_1}_S l_1' \in T_S$ and $l_2 \xrightarrow{\{?,!\}a,G_2,\lambda_2}_{TP} l_2' \in T_{TP}$ then
Add (l_1', l_2') to RL.
Add $(l_1, l_2) \xrightarrow{\{?,!\}a,G_1\&G_2,\lambda_1\cup\lambda_2}_{SP} (l_1', l_2')$ to T_{SP}.
EndIf
EndWhile

Fig. 5. Synchronous Product's Construction Algorithm.

The complexity of the algorithm is $\theta((|L_S| \times |L_{TP}|) \times |T_S|)$. Indeed, the loop *while* in step3 of the algorithm executes for each reachable location. At the worst case, the number of locations in the synchronous product is $(|L_S| \times |L_{TP}|)$ (see definition 8). For each iteration of the loop *while*, we have to traverse all the transitions of both the specification and test purpose to see if there is any transition leaving from l_1 and l_2 respectively. The complexity of this traversal is $\theta(|T_S| + |T_{TP}|)$, which is equivalent to $\theta(|T_S|)$ since $|T_{TP}|$ is less than $|T_S|$.

3.2 Sampling the Regions Graph of the Synchronous Product

Since the synchronous product obtained in the previous step is a TIOA, its operational semantics is given by its regions graph. However, as one can see from the definition 6, each state in the regions graph consists of an infinite number of clock valuations and has a delay transition labeled with the generic symbol d. To generate test cases from the regions graph of the synchronous product, we have to choose a set of representatives for each state and accordingly instantiate the delay transition d. The objective of this step of our approach is to construct a sub-automaton of the regions graph of the synchronous product according to proposition 3. This operation is called the sampling of the regions graph and the resulting sub-automaton is called the *Grid Automata (GA)* [LY93,ENDKE98,SVD01,ENDK02]. Figure 6 shows the algorithm used to construct such sub-automaton.

INPUT : - A synchronous product $SP = (I_{SP}, O_{SP}, L_{SP}, l^0_{SP}, C_{SP}, T_{SP})$.
OUTPUT: - A sub-automaton GA of the regions graph of SP.
$s^0 \leftarrow (l^0_{SP}, 0)$.
granularity $\leftarrow \frac{1}{k+1}$.
STEP3: Initialize the sets to be used
$RS \leftarrow l^0_{SP}$ // the set of reachable states.
$HS \leftarrow \emptyset$ // the set of handled states.
While $RS \backslash HS \neq \emptyset$ do
Get a state $s = (l, v)$ from $RS \backslash H_S$
Add s to HS
For each transition $l \overset{\{?,!\}a,G,\lambda}{\longrightarrow} l'$ in TIOA do
If $v \models G$ then Add $(l, v) \overset{\{?,!\}a}{\longrightarrow} (l', [\lambda := 0]v)$ to GA if it does not exist
Add $(l', [\lambda := 0]v)$ to RS if it does not exist
EndIf
EndFor
Add $(l, v) \overset{granularity}{\longrightarrow} (l, v + granularity)$ to GA if it does not exist.
Add $(l, v + granularity)$ to RS if does not exist
EndWhile

Fig. 6. Sampling Algorithm.

The algorithm takes a synchronous product as input and constructs a grid automaton of its regions graph. The algorithm proceeds in many steps. Given the TIOA of the synchronous product, we first calculate the granularity of sampling. This granularity is equal to $\frac{1}{k+1}$, where k is the number of clocks in the TIOA. In a second step, we create the initial state formed with the initial location of the TIOA and a clock valuation that sets all clocks to zero. In a third step, we create all reachable states from the initial state with repetitive $\frac{1}{k+1}$ delay transitions. Then, for each reachable state (l, v), we create a transition $(l, v) \overset{\{?,!\}a}{\longrightarrow} (l', [\lambda := 0]v)$ for each transition $l \overset{\{?,!\}a,G,\lambda}{\longrightarrow} l'$ in TIOA such that v satisfies G. Afterwards, we repeat the same process starting with state $(l', [\lambda := 0]v)$.

Example 5. The granularity used to sample the regions graph of the synchronous product of Figure 4 is $\frac{1}{3}$. The resulting grid automata containing only the executions, which lead to a verdict "Pass" is shown in Figure 7. Notice that each state of this automata has an outgoing delay transition labeled with the same delay $\frac{1}{3}$ (the transition between $(lp3, (0, 0))$ and $(lp3, (1, 1))$ on delay 1 is the sum of three consecutive transitions on delay $\frac{1}{3}$).

The complexity of the algorithm is exponential on the number of clocks of TIOA. Indeed, the outer loop *while* of the algorithm executes for each reachable state. At the worst case, the number of states in a $n-$clocks TIOA is exponential on the number of clocks n(see [EEN98]). For each iteration of the loop *while*, we have to traverse all the transitions of the TIOA leaving from l (l is the location of the state we choose in the current iteration). The complexity of this traversal is $\theta(|T_S|)$.

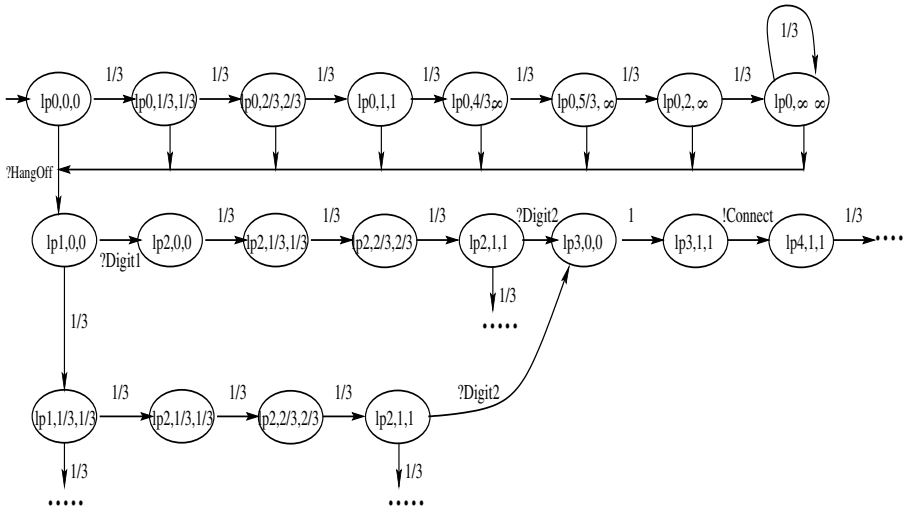

Fig. 7. Grid Automata of Figure 4.

3.3 Traversal of the Sub-automaton

This step of our approach consists of traversing the automaton derived during the previous step to obtain test cases for the system being tested. The algorithm used here depends on the data structure adopted to represent the sub-automaton. An important representation of the sub-automaton is to use a graph. Therefore, timed test cases can be derived from the graph using a depth traversal. So, each traversal represents a test case that starts at the initial state of the grid automaton and finishes when a leaf is reached. Figure 8 shows the algorithm used to generate test cases.

The algorithm takes the sub-automaton extracted during the second step of our approach as input and produces test cases as outputs. The algorithm proceeds as follows. After initializing all the variables to be used (VS and TC), we traverse all the states of GA, one by one, starting from the initial state and we add the chosen state to the set VS to indicate that it has been visited. Then, we add all the neighbors of the chosen state to the set NS and we recursively handle them all before going back to choose another state from the set of previous states.

Example 6. The test cases resulting from applying step3 on Figure 7 are given in Table 2. To lessen the length of test cases, we have summed up all consecutive delays in each of them. Here, the test case $?HangOff.?Digit1.1.?Digit2.1$ means that when testing the implementation, the tester should apply the input $?HangOff$ followed by the input $?Digit1$, waits 1 time unit, applies the input $?Digit2$ and observes the output $!Connect$ within 1 time unit.

The complexity of the algorithm is $\theta(max(a, n)$, where a and n are respectively the number of transitions and the number of states in GA. Indeed, the

INPUT : - A sub-automaton of the regions graph of SP.
OUTPUT: - A set of timed test cases.
$VS \leftarrow \emptyset$. (the set of visited states)
$TC \leftarrow \emptyset$. (a test case)
While ($S_{SP} \backslash VS \neq \emptyset$) do
Choose a state s from $S_{SP} \backslash VS$ (the first time, the initial state is chosen).
Add s to VS.
$NS \leftarrow$ all the neighbors of s. (the set of neighbor states)
While ($NS \neq \emptyset$) do
Choose and remove a state s_1 from NS such that $s \xrightarrow{\{?,!\}a} s_1 \in T_{GA}$.
Concatenate $\{?,!\}a$ with TC.
Add all the neighbors of s_1 to NS.
If s_1 has no outgoing transition then
Print TC.
$TC \leftarrow TC \backslash \{?,!\}a$.
EndIf
EndWhile
EndWhile

Fig. 8. Timed Test Cases Generation Algorithm.

Table 2. Timed Test Cases Generated.

$?HangOff.?Digit1.1.?Digit2.1.!Connect$
$\frac{1}{3}.?HangOff.?Digit1.1.?Digit2.1.!Connect$
$\frac{2}{3}.?HangOff.?Digit1.1.?Digit2.1.!Connect$
$1.?HangOff.?Digit1.1.?Digit2.1.!Connect$
$\frac{4}{3}.?HangOff.?Digit1.1.?Digit2.1.!Connect$
$\frac{5}{3}.?HangOff.?Digit1.1.?Digit2.1.!Connect$
$2.?HangOff.?Digit1.1.?Digit2.1.!Connect$
$\frac{7}{3}.?HangOff.?Digit1.1.?Digit2.1.!Connect$
$\frac{8}{3}.?HangOff.?Digit1.1.?Digit2.1.!Connect$
$?HangOff.\frac{1}{3}.?Digit1.\frac{2}{3}.?Digit2.1.!Connect$
$\frac{1}{3}.?HangOff.\frac{1}{3}.?Digit1.\frac{2}{3}.?Digit2.1.!Connect$
$\frac{2}{3}.?HangOff.\frac{1}{3}.?Digit1.\frac{2}{3}.?Digit2.1.!Connect$
$1.?HangOff.\frac{1}{3}.?Digit1.\frac{2}{3}.?Digit2.1.!Connect$
$\frac{4}{3}.?HangOff.\frac{1}{3}.?Digit1.\frac{2}{3}.?Digit2.1.!Connect$
$\frac{5}{3}.?HangOff.\frac{1}{3}.?Digit1.\frac{2}{3}.?Digit2.1.!Connect$
$2.?HangOff.\frac{1}{3}.?Digit1.\frac{2}{3}.?Digit2.1.!Connect$
$\frac{7}{3}.?HangOff.\frac{1}{3}.?Digit1.\frac{2}{3}.?Digit2.1.!Connect$
$\frac{8}{3}.?HangOff.\frac{1}{3}.?Digit1.\frac{2}{3}.?Digit2.1.!Connect$

traversal of all the states of GA (i.e., the outer *while* loop) takes a time of $\theta(n)$ and for each state the inner *while* loop executes a number of times equal to the number of transitions outgoing from that state. By summing up all the transitions outgoing from all states, one can easily see that the complexity of the inner *while* loop is $\theta(a)$.

4 Conclusion and Discussion

We presented a timed test cases generation method based on test purposes and using TIOA model. Our approach consists of three steps. First, we construct a synchronous product of the specification and test purpose. Then, we sample the regions graph of the synchronous product in order to construct an automaton (Grid Automata) easily testable in the sense that each of its state has an outgoing delay transition labeled with the same delay. Finally, we traverse the grid automaton to extract test cases for the system.

Our method generates few test cases even for huge specification. Moreover, the test cases derived by our approach are executable in that the predicate of each transition traversed by each test case is satisfied. To study in much more details the scalability and the test coverage of our method, we are currently implementing it in order to apply it on different examples with different sizes.

Comparing our approach to timed test purpose based methods we are aware of [KLC98,SPF01], we point out the following similarities and differences. Castanet et al. [KLC98] construct a synchronous product of the specification and test purpose, as defined in this paper, for tests generation. However, the authors use Timed Input Output Machine ($TIOM$) model to describe both the specification and test purpose. $TIOM$ is different from TIOA in that the time constraints of the transitions in $TIOM$ are given as intervals and so the number of clocks used is just one. Moreover, the testing of $TIOM$ consists of calculating two types of intervals for each transition: the final potential interval and the success interval. The former defines the lower and upper bounds for the execution of the transition with respect to the beginning of the test (i.e., the initial state). The latter narrows the final potential interval by taking into account the minimum and maximum delays between the transition and the preceding one. The test verdict is $fail$ whenever a transition is executed outside its final potential interval; it is $Pass$ when all transitions are executed within their success intervals; and it is *inconclusive* if a transition is fired within its final potential interval but outside its success interval.

On the other hand, Fauchal et al. [SPF01] use timed automata (TA), as in our approach, to describe the specification and test purposes. However, timed test cases are generated based on a synchronous product of the regions graphs of the specification and test purpose rather than their TAs. This is done in three steps. First, the set of specification transitions sequences containing the same actions and in the same order as the test purpose are extracted. Then, the regions graph of each of these sequences and that of the test purpose are constructed. Finally, the regions graph of each sequence is synchronized with that of test purpose to generate test cases. For each transition in the resulting synchronous product, the authors generate two test cases to cover the borders of each clock region.

References

AD94. R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

BGRS01. P. Baker, J. Grabowski, E. Rudolph, and I. Schieferdecker. A Message Sequence Chart-profile for Graphical Test Specification, Development and Tracing . In *18th International Conference and Exposition on Testing Computer Software, Washington, DC. USA* , June 2001.

CL97. D. Clarke and I. Lee. Automatic Generation of Tests for Timing Constraints from Requirements. In *Proceedings of the Third International Workshop on Object-Oriented Real-Time Dependable Systems,* Newport Beach, California, February 1997.

COG98. Rachel Cardell-Oliver and Tim Glover. A Practical and Complete Algorithm for Testing Real-Time Systems. In *FTRTFT1998 - Formal Techniques for Real-Time Fault Tolerant Systems, Lyngby, Danmark*, 1998.

EEN98. A. Elqortobi and A. En-Nouaary. Dénombrement du Nombre des Régions dans un Automate Temporisé. Technical Report TR-1116, Département IRO, Universite de Montréal, Montréal, Canada, January 1998.

EN02. A. En-Nouaary. Testing Real-Time Systems using Test Purposes. In *International Workshop on Communication Software Engineering (IWCSE),Marrakech, Morocco*, December 2002.

ENDK02. A. En-Nouaary, R. Dssouli, and F. Khendek. Timed Wp-Method: Testing Real-Time Systems. *IEEE Transactions on Software Engineering, November*, November 2002.

ENDKE98. A. En-Nouaary, R. Dssouli, F. Khendek, and A. Elqortobi. Timed Test Cases Generation Based on State Characterisation Technique. In *19th IEEE Real-Time Systems Symposium (RTSS'98), Madrid, Spain*, December, 2-4 1998.

FAUD00. M. A. Fecko, P. D. Amer, M. U. Uyar, and A. Y. Duale. Test Generation in the presence of Conflicting Timers. In *TESTCOM Ottawa, Canada*, August-September 2000.

GHN93. J. Grabowski, D. Hogrefe, and R. Nahm. Test Case Generation with Test Purpose Specification by MSCs . In *SDL'93*, October 1993.

HNTC99. T. Higashino, A. Nakata, K. Taniguchi, and A. Cavalli. Generating Test Cases for a Timed I/O Automaton Model. In *Proceedings of the International Workshop on Testing Communica ting Systems (IWTCS'99), Budapest, Hungary*, 1999.

Hog01. D. Hogrefe. Some Implications of MSC, SDL and TTCN Time Extensions for Computer-aided Test Generation . In *10th SDL-Forum, Copenhagen, Danemark*, June 2001.

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

KAD+00. A. Khoumsi, M. Akalay, R. Dssouli, A. En-Nouaary, and L. Granger. An Approach For Testing Real-Time Protocols. In *TESTCOM Ottawa, Canada*, August-September 2000.

Kan95. I. Kang. *CTSM A Formalism for Real-Time System Analysis based on State-Space Exploration.* PhD Thesis, University of Pennsylvania, 1995.

KC00. Osmane Koné and Richard Castanet. Test Generation for Internetworking Systems. *Computer Communications*, 23:642–652, 2000.

KENDA00. A. Khoumsi, A. En-Nouaary, R. Dssouli, and M. Akalay. A New Method for Testing Real-Time Systems. In *RTCSA , Cheju Island, South Korea*, December 2000.

KLC98. Osmane Koné, Patrice Laurencot, and Richard Castanet. On the Fly Test Generation for Real-Time Protocols. In *International Conference on Computer Communications and Networks, Louisiane, USA*, 1998.

LA92. N.A. Lynch and H. Attiya. Using Mappings to Prove Timing Properties. *Distributed Computing*, 6(2):121–139, 1992.

LY93. K.G. Larsen and W. Yi. Time Abstracted Bisimulation: Implicit Specification and Decidability. In *Proceedings Mathematical Foundations of Programming Semantics (MFPS 9)*, volume 802 of *Lecture Notes in Computer Science*, New Orleans, USA, April 1993. Springer-Verlag.

MMM95. D. Mandrioli, S. Morasca, and A. Morzenti. Generating Test Cases for Real-Time Systems from Logic Specifications. *ACM Transactions on Computer Systems*, 13(4):365–398, November 1995.

NS98. Brian Nielsen and Arne Skou. Automated Test Generation from Timed Automata. In *5th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems FTRTFT'98*, September 1998.

NSY92. Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Compiling Real-Time Specifications into Extended Automata. *IEEE transactions on Software Engineering*, 18(9):794–804, September 1992.

SPF01. Sébastien Salva, Eric Petitjean, and Hacène Fouchal. A Simple Approach to Testing Timed Systems. In *Proceedings of the Workshop on Formal Approaches to Testing of Software, (FATES'01), Aalborg, Denmark*, Aug 2001.

SV96. J. Springintveld and F. Vaandrager. Minimizable Timed Automata. In B. Jonsson and J. Parrow, editors, *Proceedings of the 4th International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems,* Uppsala, Sweden, volume 1135 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

SVD01. J. Springintveld, F. Vaadranger, and P. Dargenio. Testing Timed Automata. *Theoretical Computer Science*, 254:225–257, 2001.

Tre92. J. Tretmans. *A Formal Approach to Conformance Testing*. PhD Thesis, University of Twente, August 1992.