A Method for Testing the Conformance of Real Time Systems

Ahmed Khoumsi

University of Sherbrooke, Dep. GEGI, Sherbrooke J1K2R1, Canada
khoumsi@gel.usherb.ca

Abstract. The aim of *conformance testing* is to check whether an implementation conforms to a specification. We consider the case where the specification contains timing constraints and is described by a model called Timed Automata (TA). The state space of a TA can be infinite due to the infinite number of time values. In a recent work, we proposed a method to finitely represent the state space. The proposed method transforms a TA into an equivalent finite state automaton using two special types of events, *Set* and *Exp*, and denoted se-FSA.
In the present article, we propose a conformance testing method which is applicable when the specification is described by a TA. First, we use the above-mentioned transformation procedure for representing the specification by a se-FSA. Second, we propose a procedure for generating test sequences from the se-FSA describing the specification. And third, we propose a simple architecture for executing the generated test sequences.

Keywords: Real-time systems, Conformance testing, Generalized Wp-Method, Timed Automata (TA), se-FSA, *Set*, *Exp*, Test cases generation, Test architecture.

1 Introduction

1.1 Modelling Timed Systems

Real-time systems are systems for which, in addition to the correct order of events, constraints on delays separating certain events must be satisfied. Among the models that have been developed for studying rigorously timed systems, we will consider the model of Timed Automata (TA) introduced in [1,2]. In such a model, time measures are modeled by real variables and represent exact values of time. The state of such TA is defined by a *location* and (real) values of clocks. Its state space can be infinite, due to the infinite number of clock values. The region graph approach [3] proposes a finite representation of the state space obtained by merging states, which are in some sense equivalent, into a region. However, the region graph approach can still suffer from state explosion [3]. In the worst case, the number of states is exponential in the number of clocks and polynomial in the magnitudes of the constants used in timing constraints.

W. Damm and E.-R. Olderog (Eds.): FTRTFT 2002, LNCS 2469, pp. 331–351, 2002.

1.2 Testing Timed Systems

Conformance testing (or more simply *testing*) aims to check whether an implementation, which is referred to as an *implementation under test* ($\mathcal{IUT}$), conforms to a specification. In the case of timed systems, the specification contains timing constraints and is generally described by a real-time enrichment of well known models. As an example, a TA [1] is obtained from a finite state automaton (FSA) by annotating every transition of the FSA by a timing constraint and by clock(s) to reset. Few work has been done for *testing* timed systems, for example [4,5,6,7,8,9,10,11,11,12,13,14]. For lack of space, let us introduce only the methods in [8,9,10,11] because they use the same approach as us, that is, they are inspired by test methods for untimed systems.

- The authors of [8] propose a theoretical framework for testing timed systems which are described by a variant of the TA of [1]. The TA describing the specification is transformed into a region automaton, which in turn is transformed into another FSA, referred to as a Grid Automaton. Test sequences are then generated from the Grid Automaton. 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 main limitations of this approach are:
  - the Grid Automaton has more states than the corresponding region graph,
  - the authors assume that outputs can occur only on integer values of clocks.
- The authors of [9] improve [8] by providing a more complete method for generating test sequences. They do not reduce the state explosion problem but, contrary to [8], the exact instants of outputs remain unknown, only the respect or the violation of time constraints is important.
- The authors of [10] propose a method which reduces the state explosion problem induced by the methods of [8,9]. An important limitation of this method is that it is applicable only when a *single* clock is used to describe all the timing constraints of the specification.
- The authors of [11] remove the single-clock limitation of [10], but they create another limitation by assuming that in the TA which describes the specification, the set of possible values of clocks in a location *does not depend* on the path which has been executed to reach this location.

1.3 Our Contribution and Structure of the Article

The purpose of this paper is to propose a method for testing dense real-time systems. The proposed method is based on a procedure which transforms a timed automaton (TA) into an equivalent finite state automaton (FSA) using two special types of events: *Set* and *Exp*. Such FSA is denoted se-FSA. The transformation method is presented in detail in [15]. Let us note that [15] and the present article are complementary in the sense that [15] shows how the

transformation is realized, while the present article uses the transformation as a black box and shows its application for testing purpose.

Let us present our contribution in comparison to [8,9,10,11].

Comparison to [8,9]: With our method, the TA describing the specification is transformed into a se-FSA, while in [8,9] the same TA is transformed into a region automaton (RA) and then into a grid automaton (GA). The advantage of our method is that the se-FSA can contain much less states than the corresponding RA and GA. In fact, state explosion in RA and GA increases with 1) the number of clocks and 2) the magnitudes of the constants used in the transitions' timing constraints, while using se-FSA allows to avoid the state explosion due to (2). After the transformation of the TA describing the specification into a se-FSA, we develop a procedure for the automatic generation of test sequences and propose an architecture for executing the generated test sequences.

Comparison to [10,11]: Similarly to our method, the methods in [10,11] reduce the state explosion problem of the methods in [8,9]. But contrary to [10], our method is applicable when *several* clocks are used. And contrary to [11], our method is applicable even if the set of possible values of clocks in a location *depends* on the path which has been executed to reach this location. We also propose a test architecture which is simpler than the one of [10,11]. Besides, unlike [10,11], nondeterminism and complete specification aspects are studied more rigorously.

Let us discuss about the fact that our method avoids the state explosion due to the magnitudes of the constants used in the transitions' timing constraints. In reachability analysis, several minimization methods allow to reduce significantly the state space, sometimes even more than our method [16,17,18,19,20]. There is also a method that transforms region automata (RA) into zone automata (ZA). In the worst theoretical case, this method does not reduce the state space, but in practice (i.e., in most cases), the state space is significantly reduced [21]. But all those methods keep only the minimal necessary information that allows to determine the reachable transitions. In our case, we need to keep *all* the (order and timing) information that is contained in the original TA.

The rest of the paper is organized as follows. Sect. 2 describes Timed automata (TA) and presents the fault model considered. Sect. 3 presents the transformation procedure of [15]. Sect. 4 presents the conformance relation and proposes a simple test architecture. In Sect. 5, we show how to generate test sequences from a se-FSA. And in Sect. 6, we conclude and discuss some future research issues.

2 TA Model and Fault Model

2.1 Timed Automata (TA)

We will use the following definitions and notations: "derivative" means "derivative w.r.t. (physical) time", "reset" means "set to zero", "$\sup(a, b)$" means "the

greatest of a and b", $[a,b]$ and $[a,b[$ are the intervals defined by $\{x|a \leq x \leq b\}$ and $\{x|a \leq x < b\}$, and $\mathcal{C} = \{c_1, \cdots, c_{N_c}\}$ is a set of clocks.

Definition 1 *(Continuous time, clock)* Continuous time is a real variable that evolves indefinitely and the derivative of which is equal to 1. A clock is a continuous time which can be reset with the occurrence of an event.

Definition 2 *(Canonical enabling condition, CEC, Enabling condition, EC, $\mathcal{EC}_\mathcal{C}$)* A Canonical Enabling Condition (CEC) is any formula of the form "$c_i \sim k$", where $\sim \in \{<, >, \leq, \geq, =\}$ and k is a nonnegative integer constant. An Enabling Condition (EC) is any CEC or conjunction of CECs. Let then $\mathcal{EC}_\mathcal{C}$ denote the set of ECs depending of clocks of $\mathcal{C}$ (we consider that $True \in \mathcal{EC}_\mathcal{C}$).

Definition 3 *(Reset, $\mathcal{P}_\mathcal{C}$)* A Reset is any subset of $\mathcal{C}$. Let then $\mathcal{P}_\mathcal{C}$ denote the set of resets.

Definition 4 *(Timed Automaton, TA)* A Timed Automaton (TA) [1,11] is defined by $(\mathcal{L}, \mathcal{E}, \mathcal{C}, \mathcal{T}r, l_0)$ where: $\mathcal{L}$ is a finite set of locations, l_0 is the initial location, $\mathcal{E}$ is a finite set of events, $\mathcal{C}$ is a finite set of clocks, and $\mathcal{T}r \subseteq \mathcal{L} \times \mathcal{E} \times \mathcal{L} \times \mathcal{EC}_\mathcal{C} \times \mathcal{P}_\mathcal{C}$ is a transition relation. There are two types of events: the reception of an input u (written $?u$) and the sending of an output u (written $!u$).

A transition is therefore defined by $\mathrm{Tr} = \langle q; \sigma; r; EC; Z\rangle$ where: q and r are origin and destination locations, σ is the event of the transition, Tr can occur only if $EC = true$, and after the occurrence of Tr, the clocks in Z are reset. Z is called *reset* of Tr.

The clocks used in a TA that describes a given system are just a way to express timing constraints of the system, they do not correspond necessarily to real clocks that are used by the system. For example, to specify that the delay d between two transitions Tr1 and Tr2 is such that $d \in [1,3]$, we may use a clock c_1 as follows: the Z of Tr1 is $\{c_1\}$, and the EC of Tr2 is $(c_1 \geq 1) \wedge (c_1 \leq 3)$.

Example 1 *(Timed Automaton, TA)* We consider a system modeled by the TA of Fig. 1, where locations are represented by nodes, and a transition $\mathrm{Tr} = \langle q; \sigma; r; EC; \{c_i, c_j, \cdots\}\rangle$ is represented by an arrow linking q to r and labelled by $(\sigma; EC; c_i := 0, c_j := 0, \cdots)$. The absence of EC or of clocks to reset is indicated by "-". As described in Fig. 1:

- the system is initially in l_0 and reaches l_1 at the reception of σ;
- from l_1, the system reaches l_2 by sending μ;
- from l_2, the system reaches l_0 either by receiving ϕ or by sending ρ.

The timing constraints are as follows, where $\delta_{e_1 e_2}$ denotes the delay separating e_1 and e_2: $\delta_{?\sigma!\mu} \leq 3$, $\delta_{?\sigma?\phi} < 2$, $\delta_{?\sigma!\rho} \geq 2$, $\delta_{!\mu?\phi} \geq 1$, and $\delta_{!\mu!\rho} \geq 1$.

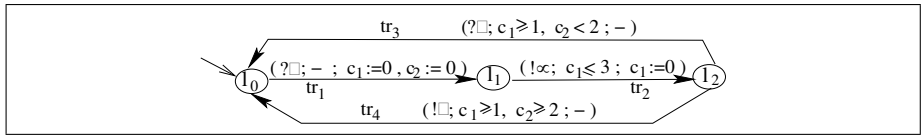

Fig. 1. Example of TA

2.2 Fault Model

Faults that may arise in an implementation of a timed system can be categorized by: 1) faults independent of timing constraints and 2) timing faults [22]. For the first category, we consider the following two types of faults determined in [23]: (i) an implementation has an *output fault* if it does not respond with the expected output in one of its states, and (ii) an implementation has a *transfer fault* if, after the reception of an input or the sending of an ouput, it enters a different state than the expected one. A fault of the second category occurs when $\mathcal{IUT}$ does not respect timing constraints described in its specification. We will consider all the types of timing constraints that can be modelled in a TA. More precisely, in a given sequence of executed events $e_1 e_2 \cdots e_n$, between every pair (e_i, e_j) where $i < j$, we may have a timing constraint in the form $\delta_{e_i e_j} \sim k$, where $\delta_{e_i e_j}$ is the delay between e_i and e_j, $\sim \in \{<, >, \leq, \geq, =\}$, and k is a nonnegative integer constant. During the execution of a sequence, the tester respects timing constraints of inputs and checks whether timing constraints of outputs are satisfied. The aim of our test method is to detect faults of category 1 and category 2. Note that after the transformation procedure mentioned in Sect. 1.3 and presented in Sect. 3, timing faults will be represented in the form of faults of category 1.

3 Transformation Procedure of Timed Automata

Let us recall that the aim of this article is not the development of the transformation (which is done in [15]), but the *use* of the transformation for the development of a method for testing real-time systems. The transformation method transforms a TA into an *equivalent* and *minimal* FSA. The latter is denoted se-FSA and uses two special types of events: *Set* and *Exp*, in addition to the events of the TA. The TA and the corresponding se-FSA are *equivalent* in the sense that they specify the same order and timing constraints (of events other than *Set* and *Exp*). More clarifications and details about this equivalence are given in Section 3.5. The se-FSA is *minimal* in the sense that it contains much less states than any other known FSA that is equivalent to the corresponding TA. Let us, for example, compare the state spaces of the grid automaton (GA) [8,9] and the se-FSA obtained from the same TA. The state space of the GA increases with the number of clocks and the magnitudes of the constants used in the transitions' timing constraints. The state space of the se-FSA is smaller because it increases only with the number of clocks.

3.1 *Set* and *Exp* Events

$Set(c_i, k)$ means: clock c_i is set to zero and will expire when its value is equal to k. More generally, $Set(c_i, k_1, k_2, \cdots, k_p)$ means that c_i is set to zero and will expire several times, when its value is equal to $k_1, k_2, \cdots, k_p$, respectively. We consider without loss of generality that $k_1 < k_2 < \cdots < k_p$.

$Exp(c_i, k)$ means: clock c_i expires and its current value is k. During the period which separates a $Set(c_i, k)$ and the corresponding $Exp(c_i, k)$, the latter is said is *foreseen*.

Therefore, $Set(c_i, k)$ is followed (after a delay k) by $Exp(c_i, k)$, and $Set(c_i, k_1, k_2, \cdots, k_p)$ is followed (after delays $k_1, \cdots, k_p$) by $Exp(c_i, k_1)$, $Exp(c_i, k_2), \cdots, Exp(c_i, k_p)$. When a $Set(c_i, m)$ occurs, every $Exp(c_i, *)$ which was foreseen before this $Set(c_i, m)$ is cancelled.

As we will see in Sect. 4, *Set* and *Exp* events are concrete events. In fact, *Set*s are sent by Test-Controller and received by Clock-Handler, and *Exp*s are sent by Clock-Handler and received by Test-Controller. If we conceptually include Clock-Handler in the system under test, then *Set*s (resp. *Exp*s) can be considered as inputs (resp. outputs) and, thus, will be prefixed by "?" (resp. "!").

3.2 Principle of Transformation Explained in a Simple Example

The procedure that transforms a TA into a se-FSA is quite complex although its basic principle is simple [15]. Let us first explain this principle in the following example that uses a single clock. We consider the following timing constraint: the delay between two events a and b falls within the interval $[1, 3[$. In a TA, such a constraint is expressed by: 1) using two transitions tr_i and tr_j which represent the occurrences of a and b, 2) resetting a clock c at the occurrence of tr_i, and 3) associating to tr_j the EC $((c \geq 1) \wedge (c < 3))$. This timing constraint can also be expressed as follows: we replace the reset of c by a $Set(c, \mathit{1}, \mathit{3})$ (the latter will be followed by $Exp(c, \mathit{1})$ and $Exp(c, \mathit{3})$) and we specify that tr_j occurs after or simultaneously to $Exp(c, \mathit{1})$ and before $Exp(c, \mathit{3})$. Therefore, the above timing constraint will be represented in a se-FSA by the following two sequences: "$\langle a, Set(c, \mathit{1}, \mathit{3})\rangle \cdot Exp(c, \mathit{1}) \cdot b \cdot Exp(c, \mathit{3})$" and "$\langle a, Set(c, \mathit{1}, \mathit{3})\rangle \cdot \langle Exp(c, \mathit{1}), b\rangle \cdot Exp(c, \mathit{3})$". (Consecutive events are separated by "$\cdot$", and simultaneous events are grouped in "$\langle\rangle$".)

3.3 A Few Details About the Construction of *Set* and *Exp*

The example used in Sect. 3.2 gives the impression that the transformation procedure is trivial. Actually, such a procedure is quite complex because it is applicable to the general case where *several* clocks are used [15]. A simpler version, which is applicable when a single clock is used, is presented in [10]. Let us consider the example of Fig. 1 in order to show a particular aspect which can occur only when several clocks are used. The four transitions are identified by tr_i, $i = 1, 2, 3, 4$. If we apply the principle of Sect. 3.2:

- the reset "$c_1 := 0$" in tr_1 is replaced by $Set(c_1, 3)$ because it is used to express the timing constraint "$c_1 \leq 3$" in tr_2;
- the reset "$c_2 := 0$" in tr_1 is replaced by $Set(c_2, 2)$ because it is used to express the timing constraints "$c_2 < 2$" and "$c_2 \geq 2$", in tr_3 and tr_4 respectively;
- the reset "$c_1 := 0$" in tr_2 is replaced by $Set(c_1, 1)$ because it is used to express the timing constraint "$c_1 \geq 1$" in tr_3 and tr_4.

Let us consider the *Exp*s that can occur from location l_2 and correspond to the ECs of outgoing transitions of l_2, i.e., tr_3 and tr_4. These *Exp*s are $Exp(c_1, 1)$ and $Exp(c_2, 2)$, whose corresponding *Set*s are in tr_2 and tr_1 respectively. In order to determine the possible orders of $Exp(c_1, 1)$ and $Exp(c_2, 2)$ in l_2, let t_1 and t_2 be the instants of transitions tr_1 and tr_2, respectively.
"$!Exp(c_1, 1)$ before $!Exp(c_2, 2)$" implies $t_2 + 1 < t_1 + 2$, i.e., $t_2 - t_1 < 1$,
"$!Exp(c_1, 1)$ simultaneous to $!Exp(c_2, 2)$" implies $t_2 + 1 = t_1 + 2$, i.e., $t_2 - t_1 = 1$,
"$!Exp(c_1, 1)$ after $!Exp(c_2, 2)$" implies $t_2 + 1 > t_1 + 2$, i.e., $t_2 - t_1 > 1$.
Therefore, in l_2 we can determine the order of the two *Exp* events if we know whether the delay between tr_1 and tr_2 (i.e., $t_2 - t_1$) is smaller than, equal to or greater than 1. For this purpose, we add $?Set(c_i, 1)$ to tr_1 (transition occurring at instant t_1). Clock c_i can be c_1 or c_2 because: (i) c_1 and c_2 are set in tr_1, (ii) c_1 and c_2 are not set between tr_1 and tr_2, and (iii) c_i is used to estimate the delay between tr_1 and tr_2. If, for example, we select c_2, tr_1 will contain $?Set(c_2, 1)$ and $?Set(c_2, 2)$ which can be combined into $?Set(c_2, 1, 2)$.

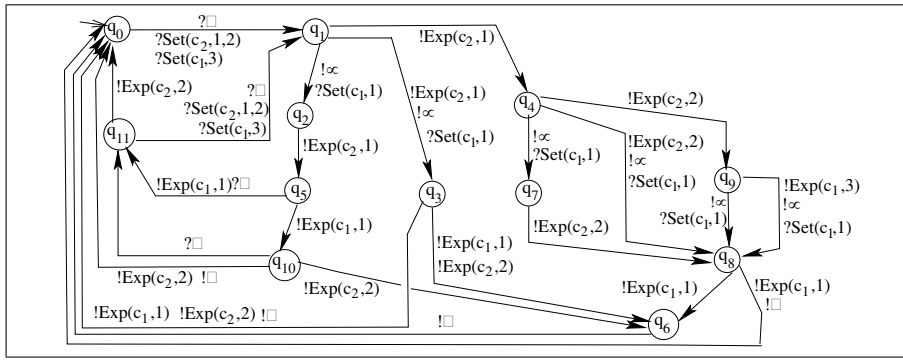

Fig. 2. Transformation of the TA of Fig. 1 into a se-FSA

3.4 An Example

If we apply the transformation to the TA of Fig. 1, we obtain the se-FSA of Fig. 2 which represents all the possible occurrences of: 1) the events of the TA, to which certain *Set* events are associated; and 2) the *Exp* events. Timing constraints specified in the TA have been transformed in the form of order constraints relatively to *Exp* events. Note that an *Exp* event occurs:

- alone in transitions $q_1 \to q_4$, $q_2 \to q_5$, $q_5 \to q_{10}$, $q_{10} \to q_6$, $q_7 \to q_8$, $q_8 \to q_6$, $q_4 \to q_9$ and $q_{11} \to q_0$;
- simultaneously to another *Exp* event in transitions $q_3 \to q_6$ and $q_3 \to q_0$;
- simultaneously to an event of the TA in transitions $q_1 \to q_3$, $q_5 \to q_{11}$, $q_{10} \to q_0$, $q_3 \to q_0$, $q_8 \to q_0$, $q_4 \to q_8$ and $q_9 \to q_8$.

More generally, each transition of a se-FSA may consist of: (a) an input or an output, (b) *Exp* events, and (c) *Set* events. Events in (a,b) are theoretically simultaneous and are immediately followed by events in (c), if any.

3.5 Equivalence between a TA and the Corresponding se-FSA

The transformation procedure guarantees that every TA and the corresponding se-FSA are equivalent [15]. Intuitively, the TA and se-FSA are equivalent in the sense that their order and timing constraints are equivalent. To clarify the notion of equivalence, let us first define the timed language of TA and the timed language of se-FSA.

Timed Language of TA. In a sequence of transitions $Tr_1\, Tr_2 \cdots$ of a TA A, there exists a timing constraint $\delta_{i,j} \sim k$ on the delay $\delta_{i,j}$ between Tr_i and Tr_j, for $i < j$, iff Tr_i resets a clock c, Tr_j has a timing constraint $c \sim k$, and every Tr_m such that $i < m < j$ (if any) does not reset c. In this study, we assume that from every location, the TA has not several outgoing transitions labelled by the same event and whose ECs can be satisfied simultaneously. This assumption allows to simplify the transformation procedure and will be replaced in Section 5.2 by a more restrictive assumption which is useful for the test generation process.

Definition 5 *(Timed event, timed execution)* A *timed event* is formally represented by a pair (e, τ) where e is an event and τ is the instant of occurrence of e. A *timed execution* is any sequence of timed events "$(e_1, \tau_1)(e_2, \tau_2) \cdots (e_i, \tau_i) \cdots$", where $\tau_1 < \tau_2 < \cdots < \tau_i < \cdots$. Such a sequence represents an execution of the sequence $e_1 e_2 \cdots e_i \cdots$ such that every e_i occurs at instant τ_i.

Remark 1 In Def. 5, we have used symbol $<$ instead of $\leq$. This means that we do not consider simultaneous transitions in a TA. This assumption has been used in order to simplify the transformation procedure. We think that this is not a restrictive assumption because simultaneousness is very unlikely.

Definition 6 *(Acceptance of timed execution by TA, timed language of TA)* Let A be a TA and $Seq = (e_1, \tau_1)(e_2, \tau_2) \cdots$ be a timed execution where $e_1, e_2, \cdots$ are members of the alphabet of A. *Seq* is accepted by A iff there exists a sequence of transitions $Tr_1\, Tr_2 \cdots$ of A such that, for $i, j = 1, 2, \cdots$ and $i \neq j$: the event of Tr_i is e_i, and for every timing constraint $\delta_{i,j} \sim k$ on the delay between Tr_i and Tr_j, we have $\tau_j - \tau_i \sim k$. The timed language of A, denoted TL_A, is the set of timed executions accepted by A.

Timed Language of se-FSA

Definition 7 *(Timed event of se-FSA, timed execution of se-FSA)* A timed event is formally represented by a pair (E, τ), where E consists of one or several (simultaneous) events and τ is the instant of occurrence of E. A *timed execution* is any sequence of timed events "$(E_1, \tau_1)(E_2, \tau_2) \cdots (E_i, \tau_i) \cdots$", where $\tau_1 < \tau_2 < \cdots < \tau_i < \cdots$. Such a sequence represents an execution of the event sequence $E_1 E_2 \cdots E_i \cdots$ such that every E_i occurs at instant τ_i.

Definition 8 *(Consistant timed execution)* A timed execution $SEQ = (E_1, \tau_1)(E_2, \tau_2) \cdots$ is consistent iff, for every E_i, E_j, $i < j$:
If E_i contains a $Set(c, k)$, E_j contains a $Exp(c, k)$, and no E_m (where $i < m < j$) contains a $Set(c, *)$, *Then* $\tau_j = \tau_i + k$.

Definition 9 *(Acceptance of timed execution by se-FSA, timed language of se-FSA)* Let B be a se-FSA and $SEQ = (E_1, \tau_1)(E_2, \tau_2) \cdots$ be a timed execution where $E_1, E_2, \cdots$ are members of the alphabet of B, i.e., each E_i labels a transition of B. SEQ is accepted by B iff : SEQ is consistent and there exists a sequence of transitions $tr_1 tr_2 \cdots$ of B such that, for $i = 1, 2, \cdots$, tr_i is labelled by E_i. The timed language of A, denoted TL_B, is the set of timed executions accepted by B.

Definition 10 *(Equivalence between TA and se-FSA)* A TA A is equivalent to a se-FSA B iff TL_A is obtained from TL_B by removing all the *Set* and *Exp* events.

4 Conformance Relation and Test Architecture

4.1 Conformance Relation

A conformance relation defines the meaning of "an implementation conforms to a specification". We make the following usual hypothesis.

Hypothesis 1 The environment (e.g., the test system $\mathcal{TS}$) sends the inputs by respecting the specification. This assumption can be more precisely defined as follows, where S is a TA describing the specification and TL_S is the timed language of S (see Sect. 3.5): after the execution by $\mathcal{IUT}$ of any (empty or non-empty) timed execution Seq of TL_S, $\mathcal{TS}$ can send an input i to $\mathcal{IUT}$ at an instant τ iff $Seq \cdot (i, \tau) \in TL_S$.

Definition 11 *(Conformance)* Assuming Hypothesis 1, $\mathcal{IUT}$ conforms to S iff: after the execution by $\mathcal{IUT}$ of any (empty or non-empty) timed sequence Seq of TL_S, $\mathcal{IUT}$ can generate an output o at an instant τ iff $Seq \cdot (o, \tau) \in TL_S$.

In Hyp. 1 and Def. 11, the empty timed execution is considered as a member of TL_S, and Symbol "$\cdot$" represents the concatenation operator.

4.2 Test System ($\mathcal{TS}$) and System under Test ($\mathcal{SUT}$)

We propose a test system ($\mathcal{TS}$) which consists of two modules called Test-Controller and Clock-Handler, respectively. The advantage of such a $\mathcal{TS}$ is that it guarantees the following equivalence.

TESTING EQUIVALENCE: Let S1 be a TA and S2 be the se-FSA obtained from S1 using the transformation procedure of Sect. 3. The following two points are equivalent:
1. $\mathcal{IUT}$ conforms to S1.
2. $\mathcal{SUT}$ consisting of $\mathcal{IUT}$ and Clock-Handler conforms to S2.

The test system ($\mathcal{TS}$) consists of Clock-Handler and Test-Controller (see Fig. 3).

Clock-Handler receives *Set* events and sends *Exp* events. Clock-Handler creates a process $PC_i(k)$ with the reception of $Set(c_i, k)$. After a delay k since its creation, $PC_i(k)$ sends $Exp(c_i, k)$ and then terminates. Clock-Handler creates a process $PC_i(k_1, k_2, \cdots, k_p)$ with the reception of $Set(c_i, k_1, k_2, \cdots, k_p)$. After each delay k_i since its creation, $PC_i(k_1, k_2, \cdots, k_p)$ sends $Exp(c_i, k_i)$. $PC_i(k_1, k_2, \cdots, k_p)$ terminates after the sending of $Exp(c_i, k_p)$. If a $Set(c_i, m)$ is received before the termination of $PC_i(*)$, then the latter is killed and a new process $PC_i(m)$ is created.

Test-Controller controls the execution of any test sequence generated from S2. More concretely, it sends inputs to $\mathcal{IUT}$, receives outputs from $\mathcal{IUT}$, sends *Set* events to Clock-Handler, and receives *Exp* events from Clock-Handler.

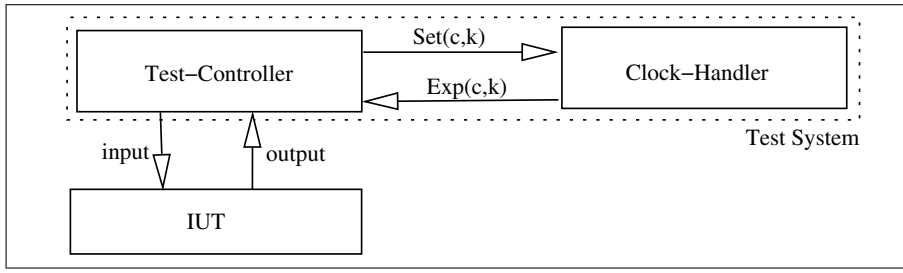

Fig. 3. Structure of the test system

Our test system ($\mathcal{TS}$) has the structure represented in Fig. 3. Let **S1** be a TA describing the specification of $\mathcal{IUT}$, **S2** be the se-FSA derived from **S1** using the transformation procedure of Sect. 3, and **Seq** be the set of test sequences generated from **S2** using a method which will be presented in Sect. 5. We consider therefore that a system conforms to **S2** if it conforms to every sequence of **Seq**.

The proposed architecture allows to check whether $\mathcal{SUT}$ (consisting of $\mathcal{IUT}$ and Clock-Handler) conforms to each sequence of **Seq** and therefore to **S2**. From the testing equivalence, we deduce that this architecture allows to check whether $\mathcal{IUT}$ conforms to **S1**. Recall that every *Set* event follows *immediately* (i.e., is

practically simultaneous to) the sending of an input or the reception of an output of $\mathcal{IUT}$.

Here is a very simple example that illustrates the test equivalence. S1 specifies that a task T must be realized in less than two units of time. S2, obtained from S1 by the transformation procedure, specifies that: (i) at the beginning of the task an alarm is programmed for occurring after two units of time, and (ii) the task must be terminated before the alarm. The programming of the alarm corresponds to a *Set* event and the occurrence of the alarm corresponds to an *Exp* event. Test-Controller orders $\mathcal{IUT}$ to start the task T and, simultaneously, programs the alarm by sending a $Set(c, 2)$ to Clock-Handler. Test-Controller deduces that $\mathcal{IUT}$ conforms to S1 iff it receives $Exp(c, 2)$ from Clock-Handler *after* it receives from $\mathcal{IUT}$ the indication that the task is terminated.

5 Test Cases Generation

5.1 Transforming se-FSA into io-FSA

Our aim is to use a test generation method applicable to FSAs whose transitions are labelled in the form input/output. Such FSAs are commonly called Mealy-automata, and we will denote them io-FSA. For that purpose, the se-FSA obtained by the transformation procedure of Sect. 3 needs to be relabelled in the form of a io-FSA. In a "classical" io-FSA, the transitions are generally labelled in the form (X/Y), where X is an input and Y is the corresponding output, and an empty X or Y is indicated by $-$. In our case, as we will see in the following, each transition of the obtained io-FSA is labelled in one of the forms: $(-/Y)$, $(X/-)$ or $(-/Y):(X/-)$. In the latter form, the symbol ":" means that the output Y is *immediately* followed by the input X. Note that in our case, X (resp. Y) may consist of one or *several* inputs (resp. outputs). Let us explain how the relabelling is realized and the semantics of the obtained io-FSA. For that purpose, let us consider the different types of transitions in a se-FSA, where $\mathcal{E}$ (resp. $\mathcal{S}$) denotes any set of *Exp* (resp. *Set*) events:

1. A transition labelled by an input i of $\mathcal{IUT}$, and possibly by $\mathcal{E}$ and $\mathcal{S}$, is relabelled $(-/\mathcal{E}):(i, \mathcal{S}/-)$. If $\mathcal{E}$ is empty, the transition is labelled $(i, \mathcal{S}/-)$. If $\mathcal{S}$ is empty, the transition is labelled $(-/\mathcal{E}):(i/-)$. If $\mathcal{E}$ and $\mathcal{S}$ are empty, the transition is labelled $(i/-)$. Intuitively, Test-Controller receives $\mathcal{E}$ from Clock-Handler and then sends i to $\mathcal{IUT}$ and $\mathcal{S}$ to Clock-Handler. $\mathcal{E}$ and i are theoretically simultaneous and are immediately followed by $\mathcal{S}$.
2. A transition labelled by an output o of $\mathcal{IUT}$, and possibly by $\mathcal{E}$ and $\mathcal{S}$, is relabelled $(-/(o, \mathcal{E})):(\mathcal{S}/-)$. If $\mathcal{E}$ is empty, the transition is labelled $(-/o):(\mathcal{S}/-)$. If $\mathcal{S}$ is empty, the transition is labelled $(-/(o, \mathcal{E})$. If $\mathcal{E}$ and $\mathcal{S}$ are empty, the transition is labelled $(-/o)$. Intuitively, Test-Controller receives o from $\mathcal{IUT}$ and $\mathcal{E}$ from Clock-Handler, and then sends $\mathcal{S}$ to Clock-Handler. o and $\mathcal{E}$ are theoretically simultaneous and are immediately followed by $\mathcal{S}$.
3. A transition labelled by $\mathcal{E}$ is relabelled $(-/\mathcal{E})$. Intuitively, Test-Controller receives $\mathcal{E}$ from Clock-Handler. When $\mathcal{E}$ consists of several *Exp*s, the latter are theoretically simultaneous.

This relabelling does not change the semantics of transitions and is just a rewriting of labels with another syntax. Therefore, the se-FSA and the corresponding io-FSA model exactly the same thing. This relabelling is not absolutely necessary. Its advantage is that we obtain an automaton with labels in the form a/b. For example, the se-FSA of Fig. 2 is relabelled into the io-FSA of Fig. 4 where:

Transitions related to the input σ: $Tr_1 = Tr_{22} = (\sigma, Set(c_1, 3), Set(c_2, 1, 2)/-)$.
Transitions related to the input ϕ: $Tr_6 = (-/Exp(c_1, 1)):(\phi/-)$, $Tr_{13} = (\phi/-)$.
Transitions related to the output μ: $Tr_2 = Tr_{10} = Tr_{19} = (-/\mu):(Set(c_1, 1)/-)$,
$Tr_3 = (-/\mu, Exp(c_2, 1)):(Set(c_1, 1)/-)$,
$Tr_{11} = (-/\mu, Exp(c_2, 2)):(Set(c_1, 1)/-)$,
$Tr_{20} = (-/\mu, Exp(c_1, 3)):(Set(c_1, 1)/-)$.
Transitions related to the output ρ: $Tr_8 = (-/Exp(c_1, 1), Exp(c_2, 2), \rho)$,
$Tr_{14} = (-/\rho, Exp(c_2, 2))$, $Tr_{18} = (-/\rho, Exp(c_1, 1))$, $Tr_{21} = (-/\rho)$.
Transitions related no neither input nor output of $\mathcal{IUT}$:
$Tr_4 = Tr_5 = (-/Exp(c_2, 1))$,
$Tr_7 = Tr_{17} = (-/Exp(c_1, 1))$, $Tr_9 = (-/Exp(c_1, 1), Exp(c_2, 2))$,
$Tr_{12} = Tr_{15} = Tr_{16} = Tr_{23} = (-/Exp(c_2, 2))$.

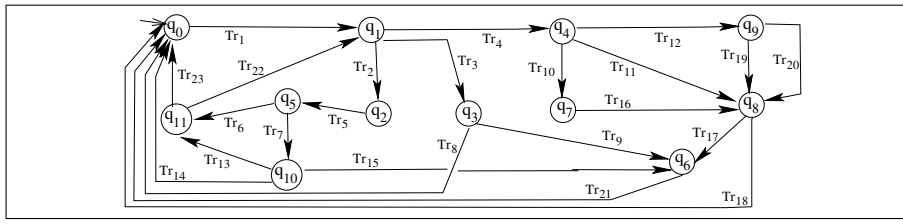

Fig. 4. io-FSA obtained after the relabeling of the se-FSA of Fig. 2. $Tr_0, Tr_1, \cdots, Tr_{23}$ are defined above.

5.2 Discussions and Hypotheses Related to Nondeterminism

In a TA, a transition labelled by an input (resp. output) of $\mathcal{IUT}$ is called input (resp. output) transition. We assume that the TA describing the specification is deterministic, i.e., from any location of the TA, we cannot have:

- several outgoing input transitions labelled by the same input and whose ECs can be satisfied simultaneously;
- an outgoing output transition and another outgoing (input or output) transition whose ECs can be satisfied simultaneously.

Note that the above assumption is stronger than the one just above Def. 5. The latter assumption is used to simplify the transformation procedure while the above assumption is used to simplify the test generation process.

Even with the assumption that the TA describing the specification is deterministic, the corresponding io-FSA (and se-FSA) may be nondeterministic,

because of the use of *Exp* events. More precisely, there are two types of nondeterministic states in a io-FSA. In the following: $\mathcal{E}$ and $\mathcal{E}'$ denote two any distinct sets of *Exp* events (each of them may consist of one or several *Exp* events), i and o denote an input and an output of $\mathcal{IUT}$ respectively, and "*" means "anything" (which may be empty, i.e., *nothing*).

Type 1: We consider states corresponding to a situation where an output o of $\mathcal{IUT}$ and a set $\mathcal{E}$ of *Exp* event(s) are both possible ($\mathcal{E}$ can be a singleton). When a state of this type is reached, we cannot foresee the order between o and $\mathcal{E}$, because we cannot foresee the instant of o. We deduce that a state of type 1 is *nondeterministic*.

For example, in State q_1 of the io-FSA of Fig. 4, the three outgoing transitions Tr_2, Tr_3, Tr_4 correspond to the three cases where the output μ is *before*, *simultaneous to*, and *after* $Exp(c_2, 1)$, respectively. We cannot foresee which of the three transitions will occur. $q_3, q_4, q_8, q_9, q_{10}$ are other states of type 1.

Type 2: We consider states corresponding to a situation where two different sets of *Exp* events, $\mathcal{E}$ and $\mathcal{E}'$, are possible ($\mathcal{E}$ and $\mathcal{E}'$ can be a singleton). Actually, such a situation does not correspond necessarily to a nondeterminism. When a state of this type is reached, we can foresee which of $\mathcal{E}$ or $\mathcal{E}'$ will occur, because we know the instants when the corresponding *Set* events have occurred.

Let us consider the case where we cannot control the instants of the *Set* events corresponding to $\mathcal{E}$ and $\mathcal{E}'$. This case may occur, for example, when these *Set* events are associated to (i.e., immediately follow) outputs of $\mathcal{IUT}$, whose instants cannot be controlled. In such a case, before the occurrence of the *Set* events, we can neither select nor foresee which of $\mathcal{E}$ or $\mathcal{E}'$ will occur. In such a case, a state of type 2 is *nondeterministic*.

To summarize about a state s of type 2, which one of $\mathcal{E}$ and $\mathcal{E}'$ will occur: (i) can always be foreseen *after* the occurrences of the corresponding *Set* events, and (ii) in certain cases, can be neither selected nor foreseen *before* the occurrences of the corresponding *Set* events. When there exist cases corresponding to (ii), state s is nondeterministic.

As an example, State 3 of Figure 5 is a state of type 2. We have Case (i) because, when state 3 is reached (i.e., after the occurrences of $Set(c_1, 2)$ and $Set(c_2, 1)$), we determine that State 4 (resp. State 6, resp. State 5) is reached if the delay between $Set(c_1, 2)$ and $Set(c_2, 1)$ is smaller than (resp. greater than, resp. equal to) 1. And we have Case (ii) because, before state 3 is reached, we do not know the delay which will separate $Set(c_1, 2)$ and $Set(c_2, 1)$ and, thus, we cannot determine which of the three states 4, 5, or 6 will be reached. ($?\sigma$ and $!\mu$ are an input and an output of $\mathcal{IUT}$, respectively.)

Since we intend to generate test sequences from a io-FSA and since the latter can be nondeterministic, we must use a test cases generation method that is applicable to nondeterministic FSAs. In this study, we have opted for the Generalized Wp-Method (GWp-Method) [24]. In order to test nondeterministic

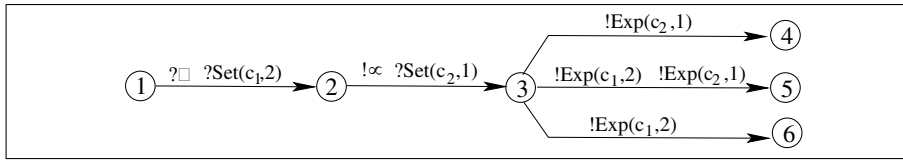

Fig. 5. Nondeterministic state of type 2.

implementations, we need to make a so-called *complete-testing assumption* which can be expressed as follows: it is possible, by applying a given input sequence t to a given $\mathcal{IUT}$ a finite number of times, to exercise all possible execution paths of the $\mathcal{IUT}$ that are traversed by t [24,25]. In our context, this assumption can also be expressed as follows: for every nondeterministic state s of type 1 or 2, it is possible, after a finite number of executions that lead to s, to execute all the outgoing transitions of s. Without such a complete-testing assumption, no test suites can guarantee full fault coverage for nondeterministic implementations.

A io-FSA (obtained from a deterministic TA), although nondeterministic, is *observable* in the sense that a state and a transition label uniquely determine the next state. In other terms, in a io-FSA, we cannot have two outgoing transitions of the same state that have the same label. *Observability* is interesting because GWp-Method is applicable for testing *observable* nondeterministic systems. In [24], an algorithm is proposed for transforming an arbitrary nondeterministic FSA into an observable nondeterministic FSA. Since our io-FSA is observable, we avoid the step of such a transformation.

5.3 Discussions and Hypotheses Related to Complete Specification

The GWp-Method we intend to use is applicable only if the io-FSA describing the specification is completely specified over its set of inputs, otherwise we must complete it. For clarity, let us first consider a specification described by a "classical" io-FSA, whose transitions are labelled in the form "i/o", where i and o denote an input and an output respectively. GWp-Method can be applied for generating test sequences from this kind of io-FSA, only if the latter is *completely specified* over the set of inputs. When such a io-FSA is not completely specified, here are two methods for completing it. For every state s and for every input i which is not specified in s:

First completion method: we add a selfloop labelled "$i/-$". Here, we assume that non-specified inputs are accepted and ignored by $\mathcal{IUT}$.

Second completion method: we add a transition labelled "$i/error$" which leads to an *Error* state, from which all inputs are ignored. Here, we assume that when $\mathcal{IUT}$ receives a non-specified input, then it sends an *error* message and enters an *Error* state from which all inputs are ignored.

Let us now consider a specification described by the type of io-FSA of Sect. 5.1; henceforth "io-FSA" denotes this type of io-FSA. With our test architecture,

where *Set* events are *inputs* (see Sect. 4), the set of inputs of the io-FSA is the set of all the elements i, $(i, \mathcal{S})$ and $\mathcal{S}$, where i is an input of $\mathcal{IUT}$, $\mathcal{S}$ consists of one of several *Set* events, and:

- each element i is an input of the $\mathcal{IUT}$ and corresponds to a transition of the io-FSA which is labelled in one of the two forms "$(-/\mathcal{E}):(i/-)$" or "$(i/-)$",
- each element $(i, \mathcal{S})$ corresponds to a transition of the io-FSA which is labelled in one of the two forms "$(-/\mathcal{E}):(i, \mathcal{S}/-)$" or "$(i, \mathcal{S}/-)$",
- each element $\mathcal{S}$ corresponds to a transition of the io-FSA which is labelled in one of the two forms "$(-/(o, \mathcal{E})):(\mathcal{S}/-)$" or "$(-/o):(\mathcal{S}/-)$".

Completing such a io-FSA cannot be done as simply as for a traditional io-FSA. For example:

- If we apply the first completion method, the addition of a selfloop labelled "$(i, \mathcal{S})/-$" or "$\mathcal{S}/-$" in a state s means that *Set* events have no influence on $\mathcal{SUT}$. This assumption is unrealistic because setting a clock has an influence on Clock-Handler.
- If we apply the second completion method, the addition of a transition labelled "$(i, \mathcal{S})/error$" from a state s to state *Error*, means that $\mathcal{SUT}$ (consisting of $\mathcal{IUT}$ and Clock-Handler) does not accept $(i, \mathcal{S})$. Since Clock-Handler never refuses a *Set* event, this implies that the $\mathcal{IUT}$ refuses i. Therefore, this addition is unrealistic if i is specified in s, i.e., if s has an outgoing transition using i.

More generally, the problem for completing a io-FSA is that the states reached by the added transitions cannot always be determined. In fact, since clocks are set in order to specify timing constraints, adding transitions which sets clocks that are currently in use, distorts the specification. (A clock c is said *in use* when: c has been previously reset in order to check the timing constraint of a transition which has not still occurred.) In order to circumvent such a problem, we can make the following assumption:

Hypothesis 2 In the TA describing the specification, the transitions associated to the same event (input or ouput) of $\mathcal{IUT}$ reset the *same* set of clocks.

When Hypothesis 2 is not satisfied, let $Tr_1, Tr_2, \cdots, Tr_k$ be any set of transitions associated to the same event e of $\mathcal{IUT}$ and which reset the clocks sets $Z_1, Z_2, \cdots, Z_k$ respectively, and let $Z = Z_1 \cup Z_2 \cup \cdots \cup Z_k$. Let $l_1, l_2, \cdots l_k$ be the original locations of $Tr_1, Tr_2, \cdots, Tr_k$, respectively. In order to respect Hypothesis 2, we need to replace each Z_i by Z. But this replacement is correct (i.e., does not distort the specification) if and only if no clock in $Z \setminus Z_i$ is in use in l_i, for $i = 1, \cdots, k$. ($Z \setminus Z_i = \{x | x \in Z, x \notin Z_i\}$.) Henceforth, we assume Hypothesis 2. The latter seems to be a strong assumption and we intend to investigate the development of a test method where Hypothesis 2 is not necessary.

Assuming Hypothesis 2, the transformation procedure of [15] (see Sect. 3) needs to be modified such that the transitions associated to the same event

(input or ouput) of $\mathcal{IUT}$ contain the same *Set* events. Let us, for example, consider two transitions Tr1 and Tr2 associated to the same event e of $\mathcal{IUT}$, which reset the same set of clocks $\{c_1, c_2\}$. The *Set* events generated by the transformation procedure of [15] can for example be: $(Set(c_1, u_1), Set(c_2, u_2))$ for Tr1, and $(Set(c_1, v_1), Set(c_2, v_2))$ for Tr2. The transformation procedure of [15] needs to be modified such that the two transitions Tr1 and Tr2 are associated to the same *Set* events $(Set(c_1, u_1, v_1), Set(c_1, u_2, v_2))$. With Hypothesis 2, such a modification can be easily integrated in the transformation algorithm of [15].

With hypothesis 2 and the above-mentioned modification of the transformation procedure of [15], we obtain a completely specified io-FSA if before applying the transformation procedure, the original TA is transformed as follows. For every location l and for every input i, let Z be the set of clocks that are reset by every transition associated to i, and EC (a boolean function depending on clock values) define the domain where i is *not* specified from l. In the general case, such a EC can be defined in the form EC_1 OR EC_2 OR $\cdots$ OR EC_n, where each EC_i is a CEC or a conjunction of CECs. In this case, we add to l several outgoing transitions $\langle ?i; -; EC_1; Z\rangle, \langle ?i; -; EC_2; Z\rangle, \cdots, \langle ?i; -; EC_n; Z\rangle$ that lead to a same location from which an outgoing transition $\langle !error; E; -\rangle$ leads to an *Error* state, from which all inputs are ignored. E is an enabling condition that bounds the delay after which $\mathcal{IUT}$ sends the *error* message. Here, we assume that when $\mathcal{IUT}$ receives a non-specified input, then it sends an *error* message and enters an *Error* state from which all inputs are ignored. Let us note that $EC = true$ in the particular case where l has no outgoing transition with i.

For example, in the TA of Fig. 1, we add: $\langle ?\sigma; -; c_1 := 0, c_2 := 0\rangle$ to locations l_1 and l_2, $\langle ?\phi; -; -\rangle$ to locations l_0 and l_1, $\langle ?\phi; c_1 < 1; -\rangle$ and $\langle ?\phi; c_2 \geq 2; -\rangle$ to location l_2. These added transitions are followed by $\langle !error; E; -\rangle$ that leads to an *Error* state. If we apply the transformation procedure of [15] to this new TA, we obtain a se-FSA which completes the se-FSA of Fig. 2 by adding ($\mathcal{E}$ denotes one or several *Exp* events):

- a transition labelled $(\sigma, ?Set(c_1, 3), ?Set(c_2, 1, 2))$ to all states, except for q_0 and q_{11};
- a transition labelled $(\mathcal{E}, ?\sigma, ?Set(c_1, 3), ?Set(c_2, 1, 2))$ to every state with an outgoing transition labelled $\mathcal{E}$;
- a transition labelled $?\phi$ to all states, except for q_{10};
- a transition labelled $(\mathcal{E}, ?\phi)$ to every state with an outgoing transition labelled $\mathcal{E}$, except for q_5.

These added transitions are followed by $!error$ which leads to an *Error* state. If we transform this new se-FSA into a io-FSA, we obtain a io-FSA which completes the io-FSA of Fig. 4 by adding:

- a transition labelled $(\sigma, Set(c_1, 3), Set(c_2, 1, 2)/-)$ to all states, except for q_0 and q_{11};
- a transition labelled $(-/\mathcal{E}) : (\sigma, Set(c_1, 3), Set(c_2, 1, 2)/-)$ to every state with an outgoing transition labelled $(-/\mathcal{E})$;
- a transition labelled $(\phi/-)$ to all states, except for q_{10};

– a transition labelled $(-/\mathcal{E}):(\phi/-)$ to every state with an outgoing transition labelled $(-/\mathcal{E})$, except for q_5.

These added transitions are followed by $(-/error)$ which leads to an *Error* state.

Hypothesis 2 and the modification of the transformation procedure of [15] can be justified by the fact that they allow us to avoid the following situation: having to add, in a io-FSA, a transition Tr1 to a state s which has an outgoing transition Tr2, such that Tr1 and Tr2 have the same event e (input or output) of $\mathcal{IUT}$ and different *Set* events. Such a situation is problematic because the added Tr1 can: neither lead to *Error* state because Tr2 implies that e is accepted by $\mathcal{IUT}$, nor be a selfloop because *Set* events of Tr1 have an effect on Clock-Handler.

5.4 Other Usual Necessary Hypotheses for Using GWp-Method

Hypothesis 3 There exists a *reset* action which brings $\mathcal{IUT}$ to its initial state.

Hypothesis 4 The specification (in the form of io-FSA) from which test sequences are generated is minimal, in the sense that it does not contain equivalent states. This hypothesis is guaranteed by the transformation procedure of [15].

Hypothesis 5 We assume that the behaviour of $\mathcal{SUT}$ (see Section 4.2) can be described by a io-FSA. This implies that $\mathcal{IUT}$ can be described by a FSA. Note that this hypothesis is equivalent to the test hypothesis in [26].

Hypothesis 6 Let n be the number of states of the io-FSA (or se-FSA) that describes the specification, and m be an upper bound of the number of states of the io-FSA that describes $\mathcal{SUT}$. We assume that $m \geq n$.

5.5 Formulation of GWp-Method

Let us first reformulate the *Test Equivalence* of Sect. 4 when "se-FSA" is replaced by "io-FSA". Let S1 be a TA and S2 be the corresponding io-FSA; S2 is obtained from S1 by using the transformations of Sect. 3 and 5.1, consecutively. The following two points are equivalent:

1. $\mathcal{IUT}$ conforms to S1.
2. $\mathcal{SUT}$ (consisting of $\mathcal{IUT}$ and Clock-Handler) conforms to S2.

Recall that our original objective was to check whether a $\mathcal{IUT}$ conforms to a TA. Using the above equivalence, we can transform our objective into: checking whether a $\mathcal{SUT}$ conforms to a io-FSA. And rightly, GWp-Method [24] is a method for generating test sequences which allow to determine whether a system (in our case, $\mathcal{SUT}$) conforms to a given io-FSA.

Note that there is no need to prove the above test equivalence, because it is equivalent to the test equivalence of Sect. 4.2. In fact, the transformation of a se-FSA into a io-FSA is just a syntactic transformation that consists of relabelling the transitions of the se-FSA. The semantics of each transition is not modified.

In the following: $q_0, q_1, q_2, \cdots, q_s$ are the states of the io-FSA (q_0 being the initial state), $Tr_0, Tr_1, \cdots, Tr_t$ are the transitions of the io-FSA, "$\cdot$" represents the concatenation operation, n and m are defined in Hypothesis 6, I is the set of inputs of the considered io-FSA, $I*$ is the corresponding (infinite) set of input sequences, I^k consists of all the input sequences of $I*$ whose length is k, and $I[k] = \{\epsilon\} \cup I \cup I^2 \cup \cdots \cup I^k$. GWp-Method defines and constructs *State Cover Set*, *Transition Cover Set*, *State Identification Set*, and *Test Suite* as follows:

A State Cover Set: $Q = \{Q_{q_0}, Q_{q_1}, \cdots, Q_{q_s}\}$, where Q_{q_i} is an input sequence which brings $\mathcal{SUT}$ from q_0 to q_i.

A Transition Cover Set: $P = P_{Tr_0} \cup P_{Tr_1} \cup \cdots \cup P_{Tr_t}$, where $P_{Tr_k} = \{Q_{q_i},\ Q_{q_i} \cdot Inp(k)\}$, $Inp(k)$ denotes the input(s) of Tr_k, and q_i and q_j are origin and destination states of Tr_k. Note that $Q \subseteq P$.

The Set: $R = P \setminus Q$. Let then $r_0, r_1, \cdots, r_p$ be the input sequences of R.

A State Identification Set $\mathcal{W} = \{W_{q_0}, W_{q_1}, \cdots, W_{q_s}\}$, where W_{q_i} is a set of input sequences which can be used to identify state q_i. We also consider the set $W_{q_0} \cup W_{q_1} \cup \cdots \cup W_{q_s}$ from which we remove every input sequence which is the prefix of another input sequence of the same set. The result is denoted W and called *Charaterization Set*.

A Test Suite $T = T_1 \cup T_2$, where:
$T_1 = reset \cdot Q \cdot I[m-n] \cdot W$ and $T_2 = reset \cdot (R \cdot I^{m-n}) \oplus \mathcal{W}$.
Operator $\oplus$ can be defined as follows: $A \oplus \{W_{q_0}, W_{q_1}, \cdots, W_{q_s}\} = ((a_0 \cdot \mathcal{W}_0) \cup (a_1 \cdot \mathcal{W}_1) \cdots \cup (a_k \cdot \mathcal{W}_k))$, where $A = \{a_0, a_1, \cdots a_k\} \subset I*$, $\mathcal{W}_i = W_{p_{i,1}} \cup W_{p_{i,2}} \cup \cdots \cup W_{p_{i,k_i}}$, and $p_{i,j}$ (for $j = 1, \cdots k_i$) being all the states that can be reached after the execution of the input sequence a_i. The input sequences of T allow to test all the states and transitions of the io-FSA [24].

5.6 Procedure for Generating Test Sequences

The proposed test generation procedure, which receives a io-FSA A as input, consists of the following three steps:

Step 1: In each state of A without any outgoing output transition (i.e., transition labelled $-/*$), add the self-loop $(-/-)$ which models "waiting without receiving any output". For example, in the io-FSA of Fig. 4, we add the selfloop $Tr_0 = -/-$ in state q_0.
Step 1 is useful because there are cases where we can obtain information about the current state just by waiting and observing what happens.

Step 2: Test sequences (which are input sequences) are generated by using GWp-Method.

Step 3: The set of test sequences is minimized by removing redundant test sequences. This is done by removing every test sequence which is the prefix of another test sequence.

5.7 Example

Let us compute $Q, P, R, \mathcal{W}$ and W for the example in Fig. 4. In the following: "$-:i$" represents the input of a transition labelled "$(-/o):(i/-)$" and means "the input i *immediately* follows the output o"; and the (possibly empty) inputs of consecutive transitions are separated by "$\cdot$"

Let ϵ denote the empty input sequence, $-$ denote the empty input which means "waiting for an output", $A = (\sigma, Set(c_1, 3), Set(c_1, 1, 2))$ denote the inputs of Tr_1 and Tr_{22}, $B = (\phi)$ denote the input of Tr_{13}, $C = (-:\phi)$ denote the input of Tr_6, and $D = (-:Set(c_1, 1))$ denote the input of Tr_2, Tr_3, Tr_{10}, Tr_{11}, Tr_{19}, Tr_{20}. We obtain the following Q, P, R, $\mathcal{W}$ and W:

$Q = \{\epsilon, A, A{\cdot}D, A{\cdot}{-}, A{\cdot}D{\cdot}{-}, A{\cdot}{-}{\cdot}D{\cdot}{-}{\cdot}{-}, A{\cdot}{-}{\cdot}D, A{\cdot}{-}{\cdot}D{\cdot}{-}, A{\cdot}{-}{\cdot}{-}, A{\cdot}D{\cdot}{-}{\cdot}{-}, A{\cdot}D{\cdot}{-}C\}$,
$R = \{-, A{\cdot}D{\cdot}{-}{\cdot}{-}{\cdot}B, A{\cdot}D{\cdot}{-}{\cdot}{-}{\cdot}{-}, A{\cdot}D{\cdot}{-}C{\cdot}A, A{\cdot}D{\cdot}{-}C{\cdot}{-}, A{\cdot}{-}{\cdot}{-}{\cdot}D, A{\cdot}{-}{\cdot}D{\cdot}{-}{\cdot}{-}\}$,
$P = Q \cup R$, $\mathcal{W} = \{W_0, W_1, W_2, W_3, W_4, W_5, W_6, W_7, W_8, W_9, W_{10}, W_{11}\}$, where
$W_0 = W_3 = \{-\}$, $W_2 = W_6 = W_8 = W_{10} = W_{11} = \{-{\cdot}{-}\}$, $W_7 = \{-{\cdot}{-}{\cdot}{-}\}$,
$W_1 = \{-{\cdot}{-}, D{\cdot}{-}, -{\cdot}D\}$, $W_4 = \{-{\cdot}D{\cdot}{-}, D{\cdot}{-}{\cdot}{-}{\cdot}{-}\}$, $W_5 = \{-{\cdot}B{\cdot}{-}\}$, $W_9 = \{D{\cdot}{-}\}$.
$W = \{-{\cdot}{-}{\cdot}{-}, D{\cdot}{-}{\cdot}{-}{\cdot}{-}, -{\cdot}D{\cdot}{-}, -{\cdot}B{\cdot}{-}\}$.

In order to compute T_1 and T_2, we must also know m which is an upper bound of the number of states of $\mathcal{IUT}$. In other to reduce the number and length of test sequences, m must be as small as possible. Since $m \geq n$ (see Hyp. 6), the ideal situation is $m = n$. For lack of space, we do not present a computation of T_1 and T_2 (for example when $m = n$), but note that parameters they depend on (i.e., $Q, P, R, \mathcal{W}, W$) have been computed.

6 Conclusion and Future Work

6.1 Contributions

This article presents a study for testing timed systems. Our main contributions in comparison to [8,9,10,11] are the following:

1. In [8,9], the elapsing of each fraction of unit of time is represented by a transition ν. Therefore, a state explosion problem arises. Our method reduces state explosion by representing only "relevant" time elapsing.
2. The test method in [10] is applicable iff a *single* clock is used. Our method generalizes the method of [10] by being applicable when one or *several* clocks are used.
3. The test method of [11] is applicable iff the set of possible values of clocks in a location of a TA *does not depend* on the path which has been executed to reach this location. This assumption is not used in our method.
4. Unlike [10,11], *nondeterminism* due to the use of *Exp* events and *complete specification* are studied in a rigorous manner.
5. The proposed test architecture is much *simpler* than the one in [10,11].

6.2 Future Work

In the near future, we intend to investigate the improvement of the transformation method and of the test method, such that certain or all of the following assumptions become unnecessary:

- We assumed Hypothesis 2 which states that in the TA describing the specification, the transitions associated to the same event of $\mathcal{IUT}$ reset the *same* set of clocks.
- We assumed that the TA is deterministic.
- We assumed that consecutive transitions of the TA cannot be simultaneous.

We also intend to study the following issues:

- We considered state and transition coverage. It would be interesting to adapt our test approach for data-based coverage.
- We considered a *centralized* $\mathcal{IUT}$. When the latter is *distributed* in several sites: 1) a distributed test architecture consisting of several local testers must be designed and 2) every generated test sequence must be distributed into local test sequences which are executed by the different testers, respectively.

References

1. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
2. A. Khoumsi, G.v. Bochmann, and R. Dssouli. Protocol synthesis for real-time applications. In *Proc. PSTV/FORTE*, Beijing, China, October 1999. Also available in http://www.gel.usherb.ca/khoumsi/Research/Public/PSTVFORTE99.ps.
3. R. Alur, C. Courcoubetis, and D. Dill. Model checking for real-time systems. In *Proc. IEEE Symposium on Logic in Computer Science*, 1990.
4. F. Liu. Test generation based on an FSM model with timers and counters. Master's thesis, University of Montreal, Department IRO, 1993.
5. 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.
6. D. Clarke and I. Lee. Automatic generation of tests for timing constraints from requirements. In *Proc. 3rd International Workshop on Object-Oriented Real-Time Dependable Systems*, Newport Beach, California, February 1997.
7. A. En-Nouaary, R. Dssouli, and A. Elqortobi. Génération de tests temporisés. In *Proc. 6th Colloque Francophone de l'Ingénierie des Protocoles*. HERMES, 1997.
8. J. Springintveld, F. Vaadranger, and P. Dargenio. Testing timed automata. Technical Report CTIT97-17, University of Twente, Amsterdam, The Netherlands, 1997.
9. A. En-Nouaary, R. Dssouli, F. Khendek, and A. Elqortobi. Timed test generation based on state characterization technique. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS)*, Madrid, Spain, December 1998.
10. A. Khoumsi, M. Akalay, R. Dssouli, A. En-Nouaary, and L. Granger. An approach for testing real time protocol entities. In *Proc. 13th Intern. Workshop. on Testing of Communicating Systems (TestCom)*, Ottawa, Canada, Aug.-Sept. 2000. Kluwer Academic Publishers. Also available in http://www.gel.usherb.ca/khoumsi/Research/Public/TESTCOM00.ps.

11. A. Khoumsi, A. En-Nouaary, R. Dssouli, and M. Akalay. A new method for testing real time systems. In *Proc. 7th Intern. Conf. on Real-Time Computing Systems (RTCSA)*, Cheju Island, South Korea, December 2000. Also available in http://www.gel.usherb.ca/khoumsi/Research/Public/RTCSA00.ps.
12. B. Nielsen and A. Skou. Automated test generation timed automata. In *Work in Progress-Session of the 21st IEEE Real-Time Systems Symposium (RTSS)*, Walt Disney World, Orlando, Florida, USA, November 2000.
13. B. Nielsen and A. Skou. Automated test generation timed automata. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 343–357, Genova, Italy, April 2001.
14. B. Nielsen and A. Skou. Test generation for time critical systems: Tool and case study. In *Proc. 13th Euromicro Conf. on Real-Time Systems*, pages 155–162, Delft, The Netherlands, June 2001.
15. A. Khoumsi. A new transformation of timed automata into finite state automata. In *Submitted to the 23rd IEEE Real-Time Systems Symposium (RTSS)*, 2002.
16. R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transitions systems. In *CONCUR*, pages 340–354. Springer-Verlag LNCS 630, 1992.
17. M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. In *Proc. 5th Conf. on Computer Aided Verification*, pages 210–224. Springer-Verlag LNCS 697, 1993.
18. I. Kang and I. Lee. State minimization for concurrent system analysis based on state space exploration. In *Proc. Conf. On Computer Assurance*, pages 123–134, 1994.
19. I. Kang and I. Lee. An efficient state space generation for analysis of real-time systems. In *Proc. Intern. Symposium on Software Testing and Analysis (ISSTA'96)*, 1996.
20. S. Tripakis and S. Yovine. Analysis of timed systems based on time-abstracting bisimulations. In *Proc. 8th Intern. Conf. on Computer Aided Verification*, pages 229–242. Springer-Verlag LNCS 1102, 1995.
21. R. Alur. Timed automata. In *Proc. 11th Intern. Conf. on Computer Aided Verification*, pages 8–22. Springer-Verlag LNCS 1633, 1999.
22. A. En-Nouaary, F. Khendek, and R. Dssouli. Fault coverage in testing real-time systems. In *Proc. 6th Intern. Conf. on Real-Time Computing Systems and Applications (RTCSA)*, Hong-Kong, December 1999.
23. G. Luo, R. Dssouli, G.v. Bochmann, P. Venkataram, and A. Ghedamsi. Test generation with respect to distributed interfaces. *Computer Standards and Interfaces*, 16:119–132, 1994.
24. G. Luo, G. v. Bochmann, and A. Petrenko. Test selection based on communicating nondeterministic finite-state machines using a generalized wp-method. *IEEE Transactions on Software Engineering*, 20(2):149–162, 1994.
25. S. Fujiwara and G. v. Bochmann. Testing nondeterministic finite-state machine with fault-coverage. In *Proc. 4th Intern. Workshop on Protocol Test Systems (WPTS)*, pages 267–280. North-Holland, 1992.
26. J. Tretmans. *A Formal Approach to Conformance Testing.* PhD thesis, University of Twente, The Netherlands, December 1992.