Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 130 (2005) 79–100
www.elsevier.com/locate/entcs

Complete Test Graph Synthesis For Symbolic Real-time Systems

Ahmed Khoumsi [1,2]

Department of Electrical and Computer Engineering
University of Sherbrooke
Sherbrooke J1K2R1, CANADA

Abstract

We study the synthesis of test cases for symbolic real-time systems. By *symbolic*, we mean that the specification of the implementation under test (**IUT**) contains variables and parameters. And by *real-time*, we mean that the specification of the **IUT** contains timing constraints. Our method combines and generalizes two test methods presented in previous work, namely : 1) a method for synthesizing test cases for (non-symbolic) real-time systems, and 2) a method for synthesizing test cases for (non-real-time) symbolic systems.

Keywords: Test cases synthesis, real-time test, symbolic test, timed input output symbolic automata, test architecture.

1 Introduction

Conformance testing (or more briefly, *testing*) aims at checking whether an *implementation under test* (**IUT**) conforms to a formal specification of the desired behavior of the **IUT**. Test activity consists of: *synthesizing* (or *generating*) test cases from the specification, and *executing* them on the **IUT**. We study the *synthesis phase*, but we also propose a test architecture for the *execution phase*. Among existing work on testing, we are interested by the following two complementary works:

[1] Work started when I was visiting at IRISA, Rennes, France, in Sept. 2002-Aug. 2003.
[2] Email: Ahmed.Khoumsi@USherbrooke.ca

doi:10.1016/j.entcs.2005.03.006

Test of real-time systems (or *real-time test*): the specification of the **IUT** contains *order* as well as *timing* constraints of the interactions between the **IUT** and its environment. Several real-time test methods have been developed in the last years [7,24,4,21,9,20,5,6,16,19].

Test of symbolic systems (or *symbolic test*): the specification of the **IUT** contains variables and parameters. A few symbolic test methods have been developed [23,22,8]. These methods aim at avoiding the generation of test cases where all variables are instantiated. Note that symbolic techniques have also been developed in other areas than testing, e.g., model-checking [3] and diagnosis [26].

We propose a test synthesis method which combines, and thus extends, real-time testing and symbolic testing. We are motivated by the desire to synthesize test cases for real-time systems that do not require instantiation of variables (i.e., do not necessitate enumeration of their possible values). We first define the model of *timed input output symbolic automata* (T_{iosa}), that adds time to the IOSTS model of [23] and is used to model the specification of the **IUT**. We use a two-step approach:

Step 1: we express the test problem into a non-real-time form, by using a transformation of a T_{iosa} into an automaton called Set-Exp-IOSA (SE_{iosa}). *SetExp* denotes such a transformation, and *SetExp*(A) denotes the SE_{iosa} obtained by transformation of a T_{iosa} A. *SetExp* basically adds to the structure of a T_{iosa} two additional types of actions: *Set* and *Exp* that model the setting and expiring of clocks, respectively.

Step 1: we adapt the non-real-time symbolic test method of [23].

As we will see, an advantage of our method is its simplicity because the main treatment of the real-time aspect is concentrated into one step.

The remainder of the paper is structured as follows. Sect. 2 describes the T_{iosa} model used to describe the specification of the **IUT**. In Sect. 3, we define formally the test problem to be solved. Sect. 4 introduces the SE_{iosa} model and the transformation "*SetExp* : $T_{iosa} \mapsto SE_{iosa}$". In Sect. 5, we propose a test architecture and present a theorem related to *SetExp*. Sect. 6 presents a method based on *SetExp* that solves the test problem. And in Sect. 7, we conclude the paper.

2 Timed IOSA (T_{iosa})

In this section, we present timed input output symbolic automata (T_{iosa}) used to describe the **IUT** and its specification. T_{iosa} is a combination of *timed automata* of [16] and *input output symbolic transition systems* (IOSTS) of [23].

2.1 Clocks and related concepts

A clock c_i is a real variable whose value can be reset (to 0) with the occurrence of an action and such that, between two resets, its derivative (w.r.t. time) is equal to 1. Let $\mathcal{H} = \{c_1, \cdots, c_{N_c}\}$ be a set of clocks.

A Clock Guard (CG) is a conjunction of formula(s) in the form "$c_i \sim k$", where $c_i \in \mathcal{H}$, $\sim \in \{<, >, \leq, \geq, =\}$, and k is a nonnegative integer. A CG can be the constant *True* (empty conjunction). Let $\Phi_{\mathcal{H}}$ be the set of CGs using clocks of $\mathcal{H}$.

A clock reset is a subset of $\mathcal{H}$, and $2^{\mathcal{H}}$ denotes the set of clock resets.

2.2 Data and related concepts

A variable is a data whose value can be set with the occurrence of an action. Let $\mathcal{V}$ be a set of variables.

A constant is a data whose value is set once at initial time. Let $\mathcal{C}$ be a set of constants.

A (communication) parameter is a data which is transmitted as a parameter of an action. Let $\mathcal{P}$ be a set of parameters.

A Data Guard (DG) is a boolean expression using data of $\mathcal{D} = \mathcal{V} \cup \mathcal{C} \cup \mathcal{P}$. Let $\Gamma_{\mathcal{D}}$ denote the set of data guards (we consider that $True \in \Gamma_{\mathcal{D}}$).

A Variable Assignment (VA) is a set of assignments $v := E$, where $v \in \mathcal{V}$ and E is an expression depending on $\mathcal{D}$. Let $\Lambda_{\mathcal{D}}$ be the set of VAs.

The domain of definition of every $x \in \mathcal{D}$ is noted $Type(x)$.

2.3 Syntax of $\mathrm{T}_{\mathrm{iosa}}$

A $\mathrm{T}_{\mathrm{iosa}}$ is defined by $(\mathcal{L}, l_0, \mathcal{H}, \mathcal{D}, \mathcal{I}, \Sigma, \mathcal{T})$, where: $\mathcal{L}$ is a finite set of locations, l_0 is the initial location, $\mathcal{H}$ is a finite set of clocks, $\mathcal{D} = \mathcal{V} \cup \mathcal{C} \cup \mathcal{P}$ is a finite set of data, $\mathcal{I}$ is a boolean expression depending of $\mathcal{V} \cup \mathcal{C}$ called initial condition, Σ is a finite set of actions, and $\mathcal{T}$ is a transition relation. To each $a \in \Sigma$ is associated a tuple $(p_1, \cdots, p_k)$ of parameters (possibly empty) denoted θ_a. Signature of a is denoted $Sig(a)$ and defined as follows:

$$Sig(a) = \begin{cases} \langle Type(p_1) \cdots Type(p_k) \rangle & \text{if } a \text{ is an input or output} \\ \text{empty tuple} & \text{if } a \text{ is an internal action} \end{cases}$$

There are three kinds of actions: the reception of an *input* i containing the tuple θ_i, written $?i(\theta_i)$; the sending of an *output* o containing the tuple θ_o, written $!o(\theta_o)$; and the occurrence of an *internal action* a, written ϵ_a. θ_i and

θ_o are omitted when empty. Receptions of inputs and sending of outputs are *observable* actions, and occurrences of internal actions are *unobservable* actions. A transition of $\mathrm{T}_{\mathrm{iosa}}$ is defined by $\mathrm{Tr} = \langle q; r; \sigma; \theta_\sigma; CG; Z_\sigma; DG; VA\rangle$, where: q and r are origin and destination locations; σ is an action in the form $?i$, $!o$ or ϵ_a; θ_σ is the (possibly empty) tuple of parameters associated to σ; CG and Z_σ are a clock guard and a clock reset; and DG and VA are a data guard and a variable assignment defined in $\mathcal{V} \cup \mathcal{C} \cup \theta_\sigma$.[3]

The index σ in Z_σ means that the clock reset of a transition depends only on its action. This restriction guarantees determinizability of $\mathrm{T}_{\mathrm{iosa}}$ [16].

Fig. 1 illustrates the definition of $\mathrm{T}_{\mathrm{iosa}}$ through an example. Locations are represented by nodes, and a transition $\mathrm{Tr} = \langle q; r; \sigma; \theta_\sigma; CG; Z_\sigma; DG; VA\rangle$ is represented by an arrow linking q to r and labeled in 3 lines by: $\sigma(\theta_\sigma)$, $(CG; Z_\sigma)$ and $(DG; VA)$. The CG and DG *True* and the absence of Z_σ or VA are indicated by "-". $\Sigma = \{\phi, \alpha, \beta, \rho\}$, $\mathcal{H} = \{c_1\}$, $\mathcal{V} = \{x\}$, $\mathcal{C} = \{p\}$, $\mathcal{P} = \{m\}$, and x, p, m are integers. ϕ cannot be an internal action because it contains parameter m, and the other actions can be of any type.

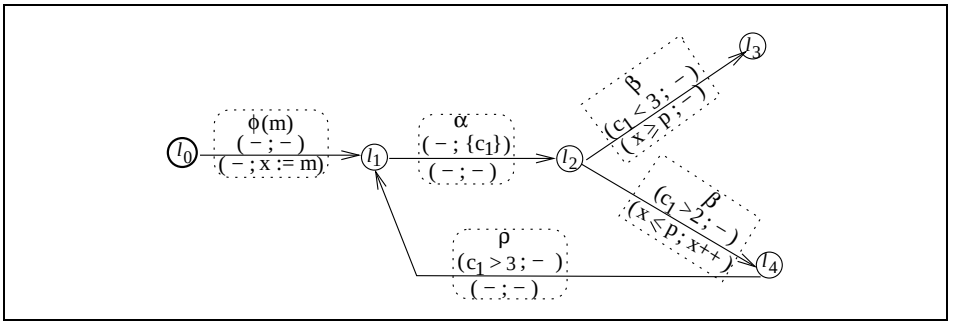

Fig. 1. Example of $\mathrm{T}_{\mathrm{iosa}}$

2.4 Semantics of $\mathrm{T}_{\mathrm{iosa}}$

At time $\tau_0 = 0$, the $\mathrm{T}_{\mathrm{iosa}}$ $A = (\mathcal{L}, l_0, \mathcal{H}, \mathcal{D}, \mathcal{I}, \Sigma, \mathcal{T})$ is at location l_0 with all clocks equal to 0, and variables and constants taking values such that $\mathcal{I}$ evaluates to *True*. A transition $\mathrm{Tr} = \langle q; r; \sigma; \theta_\sigma; CG; Z_\sigma; DG; VA\rangle$ of A is *enabled* when q is the current location and both CG and DG evaluate to *True*; otherwise, Tr is *disabled*. From this location q, the action σ (containing parameters of θ_σ) can be executed only when Tr is enabled[4]; and after the execution of σ: location r is reached, the clocks in Z_σ (if any) are reset, and the assignments in VA (if any) are applied.

[3] Note that DG and VA of a transition $\mathrm{Tr} = \langle q; r; \sigma; \theta_\sigma; CG; Z_\sigma; DG; VA\rangle$ are defined in $\mathcal{V} \cup \mathcal{C} \cup \theta_\sigma$ and *not* in the whole $\mathcal{D} = \mathcal{V} \cup \mathcal{C} \cup \mathcal{P}$

[4] But when Tr is enabled, σ is not necessarily executed.

For the example of Fig. 1, let $\delta_{u,v}$ be the delay between actions u and v:

- The T_{iosa} is initially in location l_0. At the occurrence of $\phi(m)$, location l_1 is reached and variable x is assigned with the value of m.
- From l_1, the T_{iosa} reaches l_2 at the occurrence of α.
- From l_2, the T_{iosa} reaches l_3 or l_4 at the occurrence of β. l_3 is reached only if $\delta_{\alpha,\beta} < 3$ and $x \geq p$, and l_4 is reached only if $\delta_{\alpha,\beta} > 2$ and $x \leq p$.
  We see that there is a nondeterminism when $2 < \delta_{\alpha,\beta} < 3$ and $x = p$.
  x is incremented when l_4 is reached.
- From l_3, the T_{iosa} executes nothing.
- From l_4, the T_{iosa} reaches l_1 at the occurrence of ρ. We have $\delta_{\alpha,\rho} > 3$.

The semantics of a T_{iosa} A can also be defined by the set of timed traces accepted by A. Here are a few necessary definitions:

A timed action is a pair (e, τ) where e is an action and τ is the instant of time when e occurs. When e is an input (resp. output, internal) action, then (e, τ) is called *timed input* (resp. *timed output*, *timed internal*) *action*.

A timed sequence is a (finite or infinite) sequence of timed actions "$(e_1, \tau_1) \cdots (e_i, \tau_i) \cdots$", where $0 < \tau_1 < \cdots < \tau_i < \cdots$.

A timed trace is obtained from a timed sequence by removing all its timed internal actions.

Acceptance of a timed sequence $\lambda^t = (e_1, \tau_1)(e_2, \tau_2) \cdots$, for $e_1, e_2, \cdots \in \Sigma$. Let n be the length of λ^t (n can be infinite), and ${\lambda^t}_i = (e_1, \tau_1) \cdots (e_i, \tau_i)$ be the prefix of λ^t of length i, for $0 \leq i \leq n$ (i is finite). λ^t *is accepted by* A *iff* λ^t is the empty sequence ${\lambda^t}_0$ or A has a sequence of length n of consecutive transitions $Tr_1\, Tr_2 \cdots$ starting at l_0 and such that $\forall i = 1, 2, \cdots, n$: the action of Tr_i is e_i and, after the execution of ${\lambda^t}_{i-1}$, Tr_i is enabled at time τ_i. Intuitively, λ^t corresponds to an execution of A.

Acceptance of a timed trace : Let $\mu^t = (e_1, \tau_1)(e_2, \tau_2) \cdots$ be a timed trace. μ^t is accepted by A *iff* μ^t is obtained by removing all the timed internal actions of a timed sequence accepted by A. Intuitively, μ^t corresponds to the observation of an execution of A.

Definition 2.1 The *Timed observable language of a* T_{iosa} A ($TOL_A^{T_{iosa}}$) is the set of timed traces accepted by A. That is, $TOL_A^{T_{iosa}}$ models the observable behavior of A.

The class of T_{iosa} that we will consider satisfies the following hypothesis:

Hypothesis 2.1 *Infinite timed sequences accepted by a* T_{iosa} A *are* non-zeno*, i.e., an infinite number of actions cannot be executed into a finite time interval.*

Remark 2.2 Unlike [1], with our model consecutive actions cannot occur at the same time. Actually this is not a restriction, because we consider that if an action e is followed an action f, then e and f are not simultaneous.

3 Test problem to be solved

In order to define rigorously the test problem to be solved, it is necessary to define formally a conformance relation between $\mathrm{T}_{\mathrm{iosa}}$ and the notion of test purpose. A test hypothesis is also necessary.

3.1 Conformance relation between $\mathrm{T}_{\mathrm{iosa}}$*, and some related lemmas*

In the following: I and S denote two $\mathrm{T}_{\mathrm{iosa}}$s over the same alphabet Σ, and o is an output action of Σ. We define the following conformance relation:

Definition 3.1 I $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$ S means, $\forall \lambda \in TOL_S^{\mathrm{T}_{\mathrm{iosa}}}$:
$(\lambda \cdot (o,\tau) \in TOL_I^{\mathrm{T}_{\mathrm{iosa}}}) \Rightarrow (\lambda \cdot (o,\tau) \in TOL_S^{\mathrm{T}_{\mathrm{iosa}}})$.

If the **IUT** is modeled by I, "I $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$ S" means that after an execution of the **IUT** accepted by S, the **IUT** can generate an output o at time τ only if S accepts o at time τ.

In order to give a simpler definition of $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$, let us first define the *input-completion* of $\mathrm{T}_{\mathrm{iosa}}$. Let $\Sigma_?$ be the set of inputs of the alphabet Σ, and *Univ* be a "universal" $\mathrm{T}_{\mathrm{iosa}}$ accepting *all* the timed traces over Σ. That is, $TOL_{Univ}^{\mathrm{T}_{\mathrm{iosa}}}$ contains every timed trace over Σ. The following definition is inspired from [11,12].

Definition 3.2 The *input-completion* of a $\mathrm{T}_{\mathrm{iosa}}$ $A = (\mathcal{L}, l_0, \mathcal{H}, \mathcal{D}, \mathcal{I}, \Sigma, \mathcal{T})$ is a $\mathrm{T}_{\mathrm{iosa}}$ $InpComp(A)$ that contains all the timed traces of A, as well as all the timed traces that diverge from the timed traces of A by executing inputs not accepted by A. Formally, $InpComp(A)$ is a $\mathrm{T}_{\mathrm{iosa}}$ such that:

$$TOL_{InpComp(A)}^{\mathrm{T}_{\mathrm{iosa}}} = TOL_A^{\mathrm{T}_{\mathrm{iosa}}} \cup \Big(\bigcup_{w \in TOL_A^{\mathrm{T}_{\mathrm{iosa}}}, a \in \Sigma_?, wa \notin TOL_A^{\mathrm{T}_{\mathrm{iosa}}}, x \in TOL_{Univ}^{\mathrm{T}_{\mathrm{iosa}}}} w \cdot a \cdot x \Big).$$

A is said *input-complete iff* $A = InpComp(A)$. Intuitively, an input-complete $\mathrm{T}_{\mathrm{iosa}}$ accepts every input at any time.

Lemma 3.3 $(I$ $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$ $S) \Leftrightarrow (I$ $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$ $InpComp(S))$.

Lemma 3.3 implies that we can replace a $\mathrm{T}_{\mathrm{iosa}}$ S by its input-completion before checking whether a $\mathrm{T}_{\mathrm{iosa}}$ I conforms to it, w.r.t. $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$. However, Def. 3.2 is not constructive and we do not know how to compute $InpComp(S))$ from a $\mathrm{T}_{\mathrm{iosa}}$ S in the general case. Hence, we will use the following hypothesis:

Hypothesis 3.1 *In "I* $\mathbf{conf}_{\mathrm{T_{iosa}}}$ *S", we assume S input-complete.*

Note that Lemma 3.3 and Hyp. 3.1 are inspired from their non-real-time and non-symbolic (i.e., without clocks and data) version in [11].

Lemma 3.4 *With Hypothesis 3.1:* $I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ S \Leftrightarrow TOL_I^{\mathrm{T_{iosa}}} \subseteq TOL_S^{\mathrm{T_{iosa}}}$.

Lemma 3.4 means that with Hypothesis 3.1, $\mathbf{conf}_{\mathrm{T_{iosa}}}$ is simplified into an inclusion of timed observable languages of $\mathrm{T_{iosa}}$.

3.2 Test purpose, and test hypothesis

In order to define *test purpose*, let us first define the notions of *completion* and *trap*:

Definition 3.5 A $\mathrm{T_{iosa}}$ $A = (\mathcal{L}, l_0, \mathcal{H}, \mathcal{D}, \mathcal{I}, \Sigma, \mathcal{T})$ is said to be *complete iff*: $\forall l \in \mathcal{L}$ and $\forall e \in \Sigma$, e is enabled in l for every possible clock value and data value. Intuitively, a complete $\mathrm{T_{iosa}}$ accepts every (input, output or internal) action at any time.

Definition 3.6 A *trap location* q is a location in which, for each event $\sigma \in \Sigma$, there is a selfloop transition $\mathrm{Tr} = \langle q; q; \sigma; \theta_\sigma; True; \emptyset; True; \emptyset \rangle$. That is, when a trap is reached then it is never left and every action is executable from it at any time

Definition 3.7 A *test purpose* is a $\mathrm{T_{iosa}}$ *TP* used to select the behaviors to be tested. By analogy with [10,23,16], *TP* is complete, deterministic, and equipped with two sets of trap locations A and R (for *Accept* and *Refuse*). Timed Sequences to be considered in testing activity are those terminating in and not traversing a location A, while timed sequences to be ignored are those traversing or terminating in a location R.

A test purpose is used to select a part of the specification (and thus, to ignore the remaining part) before applying a test generation method.

The following *test hypothesis* is inspired from [25]:

Hypothesis 3.2 IUT *can be described by a (possibly unknown) input-complete* $\mathrm{T_{iosa}}$ $\mathcal{IUT}$.

3.3 Formalization of the test problem

Given two $\mathrm{T_{iosa}}$s *Spec* and *TP* over the same alphabet, describing the specification and the test purpose, respectively, the objective is to synthesize an automaton *CTG* (Complete Test Graph), from which test cases can be extracted and executed in order to determine whether: $\mathcal{IUT}$ $\mathbf{conf}_{\mathrm{T_{iosa}}}$ *Spec*.

We assume *Spec* input-complete (see Hyp. 3.1). *CTG* is an interesting automaton because it contains all test cases of *Spec* corresponding to *TP*.

The test system takes into account *TP* by *ignoring* every execution λ of the **IUT** accepted by *Spec* (i.e., $\lambda \in TOL^{\mathrm{T_{iosa}}}_{IUT} \cap TOL^{\mathrm{T_{iosa}}}_{SpecTP}$) and such that: a location R of *TP* may be reached by λ, or no location A of *TP* is reachable after λ by *Spec*.

4 Transformation of $\mathrm{T_{iosa}}$ into $\mathrm{SE_{iosa}}$

A transformation, called *SetExp*, is presented in [18] and applied in [13,16,17,14,15]. *SetExp* basically transforms a timed automaton (TA) into a finite state automaton by adding to the structure of the TA two additional types of actions: *Set* and *Exp*, that capture the temporal aspect of the TA. In the present article, we apply *SetExp* to $\mathrm{T_{iosa}}$ instead of TA. When applying *SetExp* to $\mathrm{T_{iosa}}$, the semantics of data and their DG and VA is ignored, that is, they are processed just like action labels. The latter is taken into account when using (interpreting, processing, ...) the automaton called $\mathrm{SE_{iosa}}$ that results from *SetExp*. Our test problem will be solved in Sect. 6 by using *SetExp*.

In this Section, we present the $\mathrm{SE_{iosa}}$ model and illustrate *SetExp* by an example (a detailed description of *SetExp* can be found in [18]). Let A be a $\mathrm{T_{iosa}}$ over an alphabet Σ and $SetExp(A)$ be the $\mathrm{SE_{iosa}}$ obtained by transformation.

4.1 Actions Set and Exp

$Set(c_i, k)$ means: clock c_i is reset (to 0) 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 reset and will expire several times, when its value is equal to $k_1, k_2, \cdots, k_p$, resp. We assume 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.

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, then all $Exp(c_i, *)$ which were expected before this $Set(c_i, m)$ are canceled.

4.2 Basic principle of SetExp

In a $\mathrm{T_{iosa}}$ A, a clock c is reset with the objective to compare later its value to (at least) one constant, say k. The action $Set(c, k)$ is very convenient for that purpose, because it resets c and programs $Exp(c, k)$ which is a notification

when $c = k$. The principle of *SetExp* when applied to a $\mathrm{T}_{\mathrm{iosa}}$ A is:

(i) To replace each clock reset in A by the appropriate *Set* action; and then

(ii) To construct a finite state automaton, denoted *SetExp*(A), that accepts sequences containing actions of A and *Set* actions obtained in (1) and the corresponding *Exp* actions, and such that the order of actions in each accepted sequence respects order and timing constraints of A.

In order to give a trivial example that illustrates *SetExp*, let us consider the following two specifications. Specification 1: a task must be realized in less than two units of time. Specification 2: at the beginning of the task an alarm is programmed for occurring after two time units, and the task must be terminated before the alarm. It is clear that these two specifications define the same timing constraint. In this example, *SetExp* can be used to obtain the second specification from the first one. The programming of the alarm corresponds to a *Set* action, and the occurrence of the alarm corresponds to an *Exp* action.

4.3 Transitions of $\mathrm{SE}_{\mathrm{iosa}}$

Recall that a transition of $\mathrm{T}_{\mathrm{iosa}}$ is defined by $\langle q; r; \sigma; \theta_\sigma; CG; Z_\sigma; DG; VA\rangle$ and represented in a figure by an arrow linking q to r and labeled by: $\sigma(\theta_\sigma)$, $(CG; Z_\sigma)$ and $(DG; VA)$. Let: η denote an action of the alphabet Σ of the $\mathrm{T}_{\mathrm{iosa}}$ A with its parameters, $\mathcal{S}$ (resp. $\mathcal{E}$) denote a set of *Set* (resp. *Exp*) actions, and *occurrence of* $\mathcal{S}$ (resp. $\mathcal{E}$) mean the simultaneous occurrences of all the actions in $\mathcal{S}$ (resp. $\mathcal{E}$). We categorize transitions of $\mathrm{SE}_{\mathrm{iosa}}$ *SetExp*(A) into three types:

Type 1 : transition labeled by $(\mathcal{E})$ represents the occurrence of $\mathcal{E}$.

Type 2 : transition labeled by (η) or $(\eta, \mathcal{S})$, and by a *DG* and a *VA*. (η) represents the occurrence of η, $(\eta, \mathcal{S})$ represents the simultaneous occurrences of η and $\mathcal{S}$, and *DG* and *VA* have the same semantics as in $\mathrm{T}_{\mathrm{iosa}}$. A transition TR of Type 2 in *SetExp*(A), corresponds to a transition Tr of A such that: Tr and TR have the same η and *DG* and *VA*, and Tr resets the clocks in the $\mathcal{S}$ (if any) of TR.

Type 3 : transition labeled by $(\mathcal{E}, \eta)$ or $(\mathcal{E}, \eta, \mathcal{S})$, and by a *DG* and a *VA*. $(\mathcal{E}, \eta)$ represents the simultaneous occurrences of $\mathcal{E}$ and η, and $(\mathcal{E}, \eta, \mathcal{S})$ represents the simultaneous occurrences of $\mathcal{E}$, η and $\mathcal{S}$. A transition TR of Type 3 in *SetExp*(A) corresponds to simultaneous executions of $\mathcal{E}$ and a transition Tr of A such that: Tr and TR have the same η and *DG* and *VA*, and Tr resets the clocks in the $\mathcal{S}$ (if any) of TR.

Remark 4.1 A transition of type 3 corresponds to the simultaneity of two

transitions of type 1 and 2, respectively.

Definition 4.2 An *Exp-Trans* of $SetExp(A)$ is a transition of type 1 or 3, i.e., whose label contains one or several *Exp* actions.

4.4 Example of transformation SetExp : $\mathrm{T}_{\mathrm{iosa}} \mapsto \mathrm{SE}_{\mathrm{iosa}}$

For the $\mathrm{T}_{\mathrm{iosa}}$ A of Fig. 1, we obtain the $\mathrm{SE}_{\mathrm{iosa}}$ $SetExp(A)$ of Fig. 2, where $Set_{2,3}$ is an abbreviation of $?Set(c_1, 2, 3)$, Exp_i is an abbreviation of $!Exp(c_1, i)$ for $i = 2, 3$, $x{+}{+}$ means "x is incremented by 1", and the constant *DG True* and the absence of *VA* are indicated by "-". Transitions of Type 1 are those labeled Exp_i. Transitions of Types 2 and 3 are labeled in two lines, where Line 2 consists of $(DG; VA)$. Transitions of Type 2 are those labeled $\phi(m)$, $(\alpha, Set_{2,3})$, β or ρ in Line 1. Transitions of Type 3 are those labeled (Exp_i, β) in Line 1, and correspond to the simultaneous executions of Exp_i and β. We do not indicate whether each action $\phi(m)$, α, β or ρ is an input, an output or an internal action, because this is irrelevant for the comprehension of *SetExp*.

Note that this example does not respect Hyp. 3.1 (input-completeness), because the aim here is to illustrate *SetExp* while input-completeness is required by the test method and not by *SetExp*. Note that clocks are real variables although they are compared to integers (2, 3), the latter being considered just as particular reals. For example, before the occurrence of $!Exp(c_1, 2)$, c_1 takes all the nonnull real values smaller than 2.

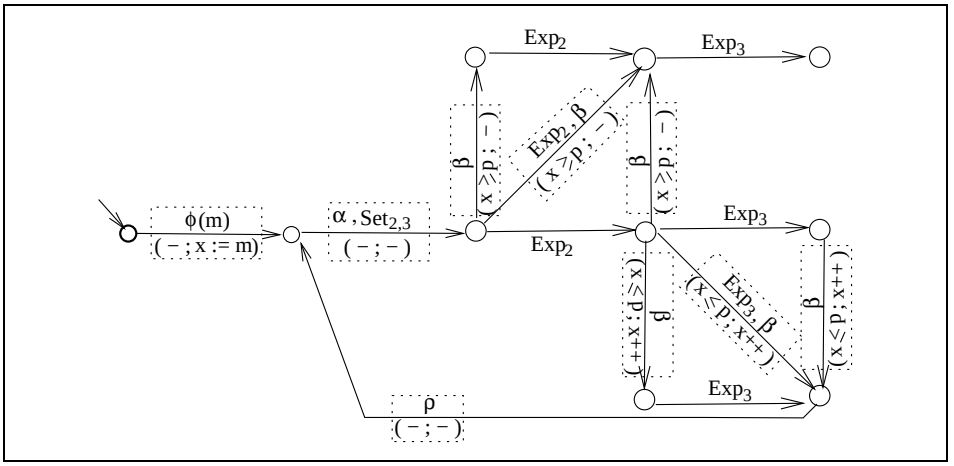

Fig. 2. $\mathrm{SE}_{\mathrm{iosa}}$ $SetExp(A)$ obtained from the $\mathrm{T}_{\mathrm{iosa}}$ A of Fig. 1

4.5 Syntax of $\mathrm{SE}_{\mathrm{iosa}}$

Let $A = (\mathcal{L}, l_0, \mathcal{H}, \mathcal{D}, \mathcal{I}, \Sigma, \mathcal{T})$ be a $\mathrm{T}_{\mathrm{iosa}}$ and $B = \mathit{SetExp}(A)$ be the corresponding $\mathrm{SE}_{\mathrm{iosa}}$. The *syntax* of B can be defined by $B = (\mathcal{Q}, q_0, \mathcal{D}, \mathcal{I}, \Lambda, \Psi)$, where: $\mathcal{Q}$ is a finite set of states, q_0 is the initial state, Λ is a finite alphabet that labels the transitions of B, Ψ is a transition relation, and $\mathcal{D}$ and $\mathcal{I}$ are the same as those used in the definition of A. A transition of B is syntactically defined by $\mathrm{TR} = \langle q; r; \mu; DG; VA\rangle$, where: q and r are origin and destination states; μ is the action(s) of TR; and DG and VA are a data guard and a variable assignment. DG and VA are always empty for transitions of Type 1 (see Sects. 4.3 and 4.4). Λ is an alphabet consisting of labels of transitions of types 1, 2 and 3 (see Sect. 4.3).

4.6 Semantics of $\mathrm{SE}_{\mathrm{iosa}}$

Let us define the *semantics* of a $\mathrm{SE}_{\mathrm{iosa}}$ $B = (\mathcal{Q}, q_0, \mathcal{D}, \mathcal{I}, \Lambda, \Psi)$. Initially, B is at state q_0 with all clocks of $\mathcal{H}$ equal to 0, and variables and constants taking values such that $\mathcal{I}$ evaluates to *true*. The transition $\mathrm{TR} = \langle q; r; \mu; DG; VA\rangle$ is *enabled* when q is the current state and DG (if any) evaluates to *true*; otherwise, TR is *disabled*. From this state q, μ (consisting of one or more actions) is executed only when TR is enabled; and after the execution of μ: State r is reached, and the assignments in VA (if any) are applied.

Let a *sequence of* $\mathrm{SE}_{\mathrm{iosa}}$ denote a sequence "$E_1E_2\cdots$", where $E_1, E_2, \cdots, \in \Lambda$; and let a *trace of* $\mathrm{SE}_{\mathrm{iosa}}$ be obtained from a sequence of $\mathrm{SE}_{\mathrm{iosa}}$ by removing all its internal actions. Let us define the semantics of a $\mathrm{SE}_{\mathrm{iosa}}$ $B = (\mathcal{Q}, q_0, \mathcal{D}, \mathcal{I}, \Lambda, \Psi)$ by the set of sequences and traces accepted by B:

Acceptance of a (finite or infinite) sequence $\lambda = E_1E_2\cdots$, for $E_1, E_2, \cdots \in \Lambda$. Let n be the length of λ (n can be infinite), and $\lambda_i = E_1E_2\cdots E_i$ be the prefix of λ of length i, for $0 \leq i \leq n$ (i is finite). λ *is accepted by* B *iff* λ is the empty sequence λ_0 or:
there exists a sequence of transitions $Tr_1\, Tr_2\cdots$ of B of length n such that $\forall i = 1, 2, \cdots, n$: Tr_i is labeled by E_i and, after the execution of λ_{i-1}, Tr_i is enabled. Intuitively, λ corresponds to an execution of B.

Acceptance of a trace μ : μ is accepted by B *iff* μ is obtained by removing the internal actions of a sequence accepted by B. Intuitively, μ corresponds to the observation of an execution of B.

We can now introduce the notion of Observable Language of a $\mathrm{SE}_{\mathrm{iosa}}$:

Definition 4.3 The observable language of a $\mathrm{SE}_{\mathrm{iosa}}$ B ($OL_B^{\mathrm{SE}_{\mathrm{iosa}}}$) is the set of traces accepted by B. That is, $OL_B^{\mathrm{SE}_{\mathrm{iosa}}}$ models the observable behavior of B.

Given a SE_{iosa} B, one can remark that $OL_B^{\text{SE}_{\text{iosa}}}$ implicitly respects the following condition, called the *Consistency condition*: every $Set(c,k)$ and its corresponding $Exp(c,k)$ are effectively separated by time k.

We terminate this section by presenting a fundamental property of *SetExp*. Let $TL = AddTime(L)$ be a timed language obtained from a language L by associating a time to each action such that the consistency condition is respected. Let $RmvSetExp(TL)$ be obtained from a timed language TL by removing all the *Set* and *Exp* actions, if any. We have the following theorem of equivalence:

Theorem 4.4 $RmvSetExp(AddTime(OL_{SetExp(A)}^{\text{SE}_{\text{iosa}}})) = TOL_A^{\text{T}_{\text{iosa}}}$.

Intuitively, Theorem 4.4 states that from a behavioral point of view, there is no difference between A and $SetExp(A)$ for an observer who does not see (or ignores) *Set* and *Exp* actions. In a sense, $SetExp(A)$ does nothing but add some new actions (*Set* and *Exp*) to A that capture the relevant temporal aspect of A. As we will see in the next section, in our test method, these *Set* and *Exp* are physical actions that are produced by the test system.

5 Test architecture, and a theorem

Given two T_{iosa}s *Spec* and *TP* over the same alphabet, recall that the objective is to synthesize an automaton *CTG*, from which test cases can be extracted and executed in order to determine whether the **IUT** conforms to the part of *Spec* corresponding to *TP*. *CTG* will not be directly computed on the T_{iosa}s *Spec* and *TP*, but rather on a SE_{iosa} computed from the two T_{iosa}s. In order to make the link between *CTG* and the **IUT**, we use a particular test architecture [16] that we now present. It is represented in Fig. 3:

Clock-Handler receives *Set* events and sends *Exp* actions. (It respects the consistency condition.)

Test-Controller sends inputs to the **IUT**, receives outputs from the **IUT**, sends *Set* actions to Clock-Handler, and receives *Exp* actions from Clock-Handler.

We define the following conformance relation $\mathbf{conf}_{\text{SE}_{\text{iosa}}}$ which is simply an inclusion of observable languages of SE_{iosa}:

Definition 5.1 Let I' and S' be two SE_{iosa}s over the same alphabet:
$(I' \ \mathbf{conf}_{\text{SE}_{\text{iosa}}} \ S') \equiv (OL_{I'}^{\text{SE}_{\text{iosa}}} \subseteq OL_{S'}^{\text{SE}_{\text{iosa}}})$.

We have the following theorem, where **SUT** (System Under Test) consists of the **IUT** and Clock-Handler:

Theorem 5.2 *Let S be an input-complete* $\mathrm{T}_{\mathrm{iosa}}$*. If Test-Controller generates Set actions only when they are accepted by SetExp(S), then:*
$(\mathcal{IUT}\ \mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}\ S) \Leftrightarrow$
$(\exists\ \mathrm{SE}_{\mathrm{iosa}}\ \mathcal{SUT}$ *accepting behavior of* **SUT** *such that* $\mathcal{SUT}\ \mathbf{conf}_{\mathrm{SE}_{\mathrm{iosa}}}\ SetExp(S))$.

The above theorem implies that we can check "$\mathcal{SUT}\ \mathbf{conf}_{\mathrm{SE}_{\mathrm{iosa}}}\ SetExp(S)$" instead of "$\mathcal{IUT}\ \mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}\ S$". We have transformed the test of a real-time symbolic system into a non-real-time form, and thus, we can (and will) use and adapt a non-real-time method of Symbolic Test Generation (STG) [23].

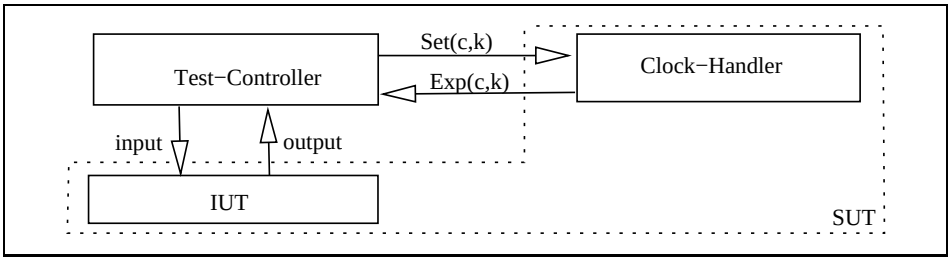

Fig. 3. Test architecture

This architecture is applicable only if transitions executing internal (i.e., unobservable) actions do not reset clocks. In fact, in order to generate *Set* actions, Test-Controller needs to observe every action to which is associated a clock reset. Hence the following hypothesis:

Hypothesis 5.1 *Transitions executing internal actions do not reset clocks.*

Hyp. 5.1 means that there is no timing constraint relatively to internal actions.

6 Method of test generation

Our test method combines, and thus extends, two complementary test methods presented in [23] and [16], respectively. It consists of four steps outlined in Fig. 4 and described in subsections 6.1 to 6.4. Its inputs are *Spec* (input-complete, from Lemma 3.3 and Hyp. 3.1) and *TP* (complete, from Def. 3.7). In a first step, we compute a $\mathrm{T}_{\mathrm{iosa}}$ *SpecTP* which accepts (all and only) the timed sequences of *Spec* and indicates the locations corresponding to the locations A and R of *TP*. Then, we synthesize in three steps (2 to 4) a complete test graph (*CTG*), from which a set of test cases can be extracted and executed on the **IUT** in order to determine whether: $\mathcal{IUT}\ \mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}\ SpecTP$. The indication A and R is used to ignore every execution of the **IUT** that leads to a location R or from which no location A is reachable. (See Sects. 3.2 and

3.3 for more detail.) The fact that *TP* is deterministic and complete implies that *Spec* is input-complete *iff* *SpecTP* is input-complete.

An advantage of our method is its simplicity because the main treatment of the real-time aspect is concentrated in Step 2. Steps 1, 3 and 4 constitute a slight adaptation of the (non-real-time) symbolic test generator (STG) [23]. [5]

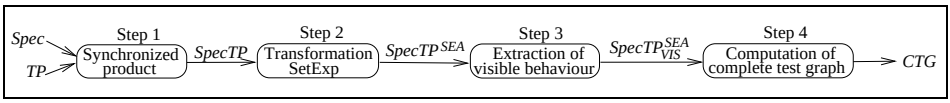

Fig. 4. Steps of the test method

Figure 5 represents *Spec* and *TP* over the alphabet $\Sigma = \{?\phi, ?\sigma, !\rho, \epsilon_a, \epsilon_b\}$ used to illustrate the steps of the test method. Data of *Spec* are $\mathcal{H}^1 = \{c_1\}$, $\mathcal{V}^1 = \{x\}$, $\mathcal{C}^1 = \{p\}$, $\mathcal{P}^1 = \{m\}$, where x, p, m are integers. Data of *TP* are $\mathcal{H}^2 = \mathcal{V}^2 = \mathcal{C}^2 = \emptyset$, $\mathcal{P}^2 = \{n\}$, where n is integer. $\neq x$ means any action of Σ different from x, and $?*$ means any input $\in \Sigma$ (i.e., $?\phi$ or $?\sigma$). *Spec* was not initially input-complete and we represent by dotted arrows the part that has been added to make *Spec* input-complete (see Hyp. 3.1). Transitions labeled only by an action mean that: their (clock and data) guards are equal to the constant *True*, and they do not reset clocks and do not have variable assignments. The *TP* of this example means that: we are interested to test executions of *Spec* terminating by the first occurrence of $!\rho$ without traversing Location *TL*.

This example of *TP* is taken very simple (with *no* timing constraint) in order to clarify the operations of the different steps. Even in a concrete case, *TP* should be relatively simple because the objective of its use is to select a relatively small part of the specification in order to concentrate only in certain aspects of the specification.

[5] Actually, STG is a software tool. But here, STG denotes the theoretical method that underlies the tool.

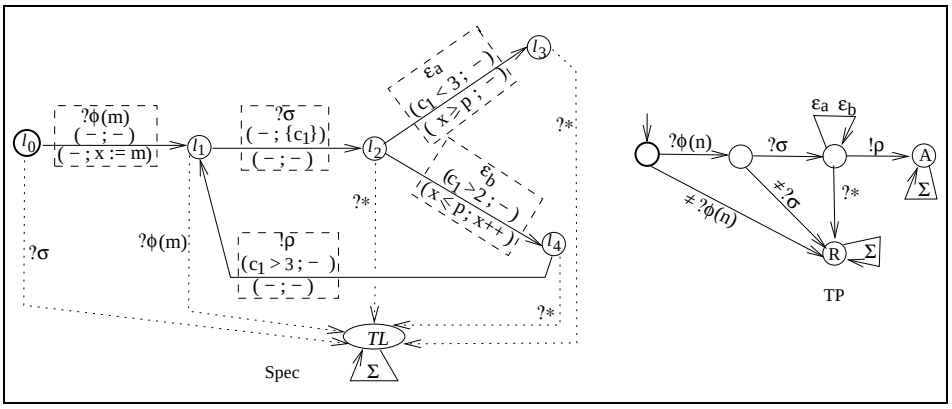

Fig. 5. Example for illustrating the test method

6.1 Step 1 : Synchronous product of Spec and TP

The aim is to compute a T_{iosa} *SpecTP* equivalent to *Spec* such that locations of *SpecTP* that correspond to locations A (resp. R) of TP are also denoted A (resp. R). For that purpose, we need to define the synchronized product of two T_{iosa}s.

Let $A^i = (\mathcal{L}^i, l_0^i, \mathcal{H}^i, \mathcal{D}^i, \mathcal{I}^i, \Sigma^i, \mathcal{T}^i)$ where $\mathcal{D}^i = \mathcal{V}^i \cup \mathcal{C}^i \cup \mathcal{P}^i$, for $i = 1, 2$, be two T_{iosa}s. The synchronized product of A^1 and A^2, written $A^1 \otimes A^2$, is inspired (but different) from the synchronized product of TA [2] and the synchronized product of IOSTS [23]. $A_1 \otimes A_2$ is defined *iff* the following four conditions are satisfied:

(i) $\Sigma^1 = \Sigma^2$. The common alphabet will then be denoted Σ. This condition can be easily relaxed [23], but we will keep it for simplicity.

(ii) $\mathcal{H}^1 \cap \mathcal{H}^2 = \emptyset$ [2].

(iii) $(\mathcal{V}^1 \cup \mathcal{P}^1) \cap (\mathcal{V}^2 \cup \mathcal{P}^2) = \emptyset$, $(\mathcal{C}^1 \cup \mathcal{P}^1) \cap \mathcal{P}^2 = \emptyset$, and $(\mathcal{C}^2 \cup \mathcal{P}^2) \cap \mathcal{P}^1 = \emptyset$ [23].

(iv) Each action $a \in \Sigma$ has the same signature in A^1 and A^2 [23].

Assuming the above four conditions satisfied, $A^1 \otimes A^2$ is defined by $(\mathcal{L}, l_0, \mathcal{H}, \mathcal{D}, \mathcal{I}, \Sigma, \mathcal{T})$ such that: $\mathcal{L} = \mathcal{L}^1 \times \mathcal{L}^2$, $l_0 = (l_0^1, l_0^2)$, $\mathcal{H} = \mathcal{H}^1 \cup \mathcal{H}^2$, $\mathcal{D} = \mathcal{V} \cup \mathcal{C} \cup \mathcal{P}$, $\mathcal{V} = \mathcal{V}^1 \cup \mathcal{V}^2$, $\mathcal{C} = (\mathcal{C}^1 \cup \mathcal{C}^2) \setminus \mathcal{V}$, $\mathcal{P} = \mathcal{P}^1 \cup \mathcal{P}^2$, $\mathcal{I} = (\mathcal{I}^1 \wedge \mathcal{I}^2)$, and the set of transitions $\mathcal{T}$ is defined as follows:
For each pair of transitions $(\langle q^i; r^i; \sigma; \theta_\sigma^i; CG^i; Z_\sigma^i; DG^i; VA^i\rangle \in \mathcal{T}^i$, $i = 1, 2$:

If θ_σ^1 and θ_σ^2 are the empty tuple ϵ : then there is a transition $\langle (q^1; q^2); (r^1; r^2); \sigma; \epsilon; CG^1 \wedge CG^2; Z_\sigma^1 \cup Z_\sigma^2; DG^1 \wedge DG^2; VA^1 \cup VA^2\rangle \in \mathcal{T}$.

If θ_σ^1 and θ_σ^2 are not empty : let $DG^{1,2}$ (resp. $VA^{1,2}$) denote the expression obtained by replacing in DG^2 (resp. VA^2) each parameter from θ_σ^2

by the corresponding, same-position parameter from θ_σ^1; then there is a transition $\langle (q^1;q^2);(r^1;r^2);\sigma;\theta_\sigma^1;CG^1 \wedge CG^2;Z_\sigma^1 \cup Z_\sigma^2;DG^1 \wedge DG^{1,2};VA^1 \cup VA^{1,2}\rangle \in \mathcal{T}$. This procedure is inspired from [23].

In Step 1, we compute $SpecTP = Spec \otimes TP$, whose locations that correspond to locations A (resp. R) of TP are indicated by A (resp. R).

Completeness of TP implies that $Spec$ and $SpecTP$ are observationally equivalent (i.e., $TOL_{Spec}^{\mathrm{T_{iosa}}} = TOL_{SpecTP}^{\mathrm{T_{iosa}}}$). Completeness of TP and input-completeness of $Spec$ imply that $SpecTP$ is input-complete. The effect of $Spec \otimes TP$ is to determine in $Spec$ all the executions that correspond to locations A and R, respectively.

For $Spec$ and TP of Fig. 5, we obtain the $SpecTP$ of Fig. 6. Locations L_1 and A_1 are equivalent in the sense that the same behavior can be produced from them. The difference between these two locations is that only A_1 corresponds to Location A of TP. Note that, in accordance with the definition of synchronized product, parameter n of TP has been removed by replacing it by parameter m of $Spec$.

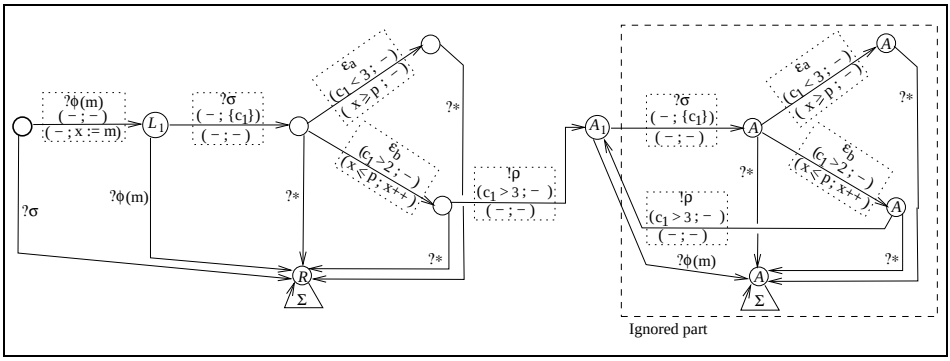

Fig. 6. Step 1: $SpecTP$ obtained from $Spec$ and TP of Fig. 5

6.2 Step 2 : Transforming the $\mathrm{T_{iosa}}$ SpecTP into $\mathrm{SE_{iosa}}$

We transform the problem into a non-real-time form by computing $SpecTP^{\mathrm{SE_{iosa}}} = SetExp(SpecTP)$. For the $SpecTP$ of Fig. 6, we obtain the $SpecTP^{\mathrm{SE_{iosa}}}$ of Fig. 7. $Set_{2,3}$ denotes $?Set(c_1, 2, 3)$, and Exp_i denotes $!Exp(c_1, i)$. States of $SpecTP^{\mathrm{SE_{iosa}}}$ that correspond to location A (resp. R) of $SpecTP$ are indicated by A (resp. R), $(\mathrm{Exp_i}, \Sigma)$ means the simultaneous occurrences of $\mathrm{Exp_i}$ and any $x \in \Sigma$, and nodes linked by a broken line indicate the same state[6]. State A_1 of Fig. 7 is equivalent to State S_1 with the difference that S_1 does not correspond to a location A of TP.

[6] They are duplicated for the sake of clarity.

In Fig. 7 and subsequent figures, if DG evaluates to *true* and VA is empty in a transition, then $(DG; VA)$ is not represented.

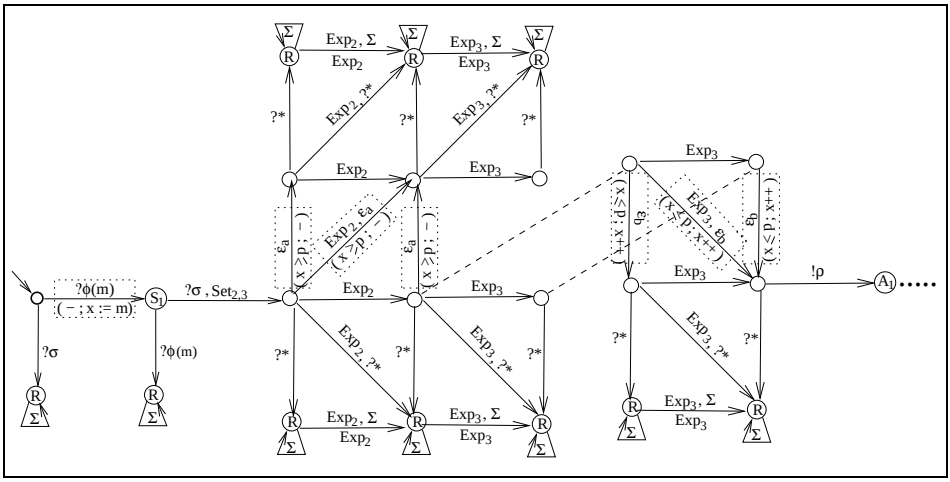

Fig. 7. Step 2: $SpecTP^{\mathrm{SE_{iosa}}}$ obtained from *SpecTP* of Fig. 6

6.3 Step 3 : Extracting the visible behavior of $SpecTP^{\mathrm{SE_{iosa}}}$

We construct the visible behavior of $SpecTP^{\mathrm{SE_{iosa}}}$ in three substeps:

Substep 3a : Internal actions are eliminated by projection into the observable alphabet. For that purpose, we can *adapt* a procedure proposed in [23]. The result is denoted $Vis(SpecTP^{\mathrm{SE_{iosa}}})$. The adaptation consists of a preliminary step where internal actions in transitions of Type 3 are simply erased. After that, we can use the procedure of [23] because the remaining internal actions are "alone" in their transitions. (Recall that we consider only the case where internal actions do not reset clocks.)

Substep 3b : $Vis(SpecTP^{\mathrm{SE_{iosa}}})$ is determinized by using a heuristic proposed in [23]. The result is denoted $Det(Vis(SpecTP^{\mathrm{SE_{iosa}}}))$.

Substep 3c : Note that every state of $Det(Vis(SpecTP^{\mathrm{SE_{iosa}}}))$ corresponds to one or several states of $SpecTP^{\mathrm{SE_{iosa}}}$. States R and A of $Det(Vis(SpecTP^{\mathrm{SE_{iosa}}}))$ are selected as follows:

- We call R every state corresponding to at least one state R of $SpecTP^{\mathrm{SE_{iosa}}}$. Intuitively, we ignore every execution which can correspond to a sequence not to be tested.
- We can call A every state corresponding to no state R and at least one state A of $SpecTP^{\mathrm{SE_{iosa}}}$. Intuitively, we accept every execution which: 1) cannot correspond to a sequence not to be tested, and 2) can correspond to a sequence to be tested. Another way, which seems more realistic,

is to call A every state corresponding only to states A of $SpecTP^{\mathrm{SE_{iosa}}}$. Intuitively, we accept an execution only when we are sure that it corresponds to a sequence to be tested.

The result is denoted $SpecTP^{\mathrm{SE_{iosa}}}_{VIS}$.

For the $SpecTP^{\mathrm{SE_{iosa}}}$ of Fig. 7, we obtain $SpecTP^{\mathrm{SE_{iosa}}}_{VIS}$ of Fig. 8.

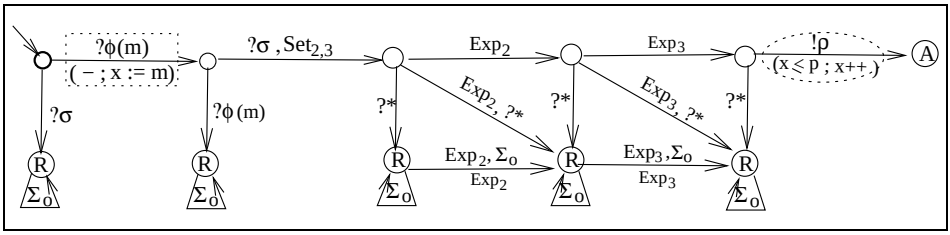

Fig. 8. Step 3: $SpecTP^{\mathrm{SE_{iosa}}}_{VIS}$ obtained from $SpecTP^{\mathrm{SE_{iosa}}}$ of Fig. 7

6.4 Step 4 : Computing a Complete Test Graph (CTG)

Recall that a transition of $\mathrm{SE_{iosa}}$ is labeled in the form $(\mathcal{E})$ or in one of the following four forms: (σ), $(\sigma, \mathcal{S})$, $(\mathcal{E}, \sigma)$, $(\mathcal{E}, \sigma, \mathcal{S})$, in addition to $(DG; VA)$. Let "output transition of $\mathrm{SE_{iosa}}$" denote any transition labeled in one of the four forms such that σ is an output of the **IUT**. By analogy with [10,16,23], we construct a Complete Test Graph (CTG) as follows:

- Let $L2A$ be the set of states of $SpecTP^{\mathrm{SE_{iosa}}}_{VIS}$ from which a A is accessible.
- Let **Pass** denote the set of states A of $SpecTP^{\mathrm{SE_{iosa}}}_{VIS}$.
- Let **Fail** = {**fail**} consist of a new state that is reached by every non-specified output transition of $SpecTP^{\mathrm{SE_{iosa}}}_{VIS}$ executable from $L2A$.
- Let **Inconc** be the set of states of $SpecTP^{\mathrm{SE_{iosa}}}_{VIS}$ that are not in $L2A \cup$ **Pass** and are accessible from $L2A$ by a single output transition of $SpecTP^{\mathrm{SE_{iosa}}}_{VIS}$.
- We then obtain CTG from $SpecTP^{\mathrm{SE_{iosa}}}_{VIS}$ by:
  - adding (implicitly) state **Fail** and its incoming transitions,
  - removing every state $\notin L2A \cup$ **Pass** $\cup$ **Inconc** $\cup$ **Fail**, and
  - removing outgoing transitions of every state $\in$ **Pass** $\cup$ **Inconc**.

To synthesize test sequences executable in acceptable time (that is, to avoid that Test-Controller waits for an input during a very long time), we select a delay T and define a fictitious event $!\delta$ whose occurrence means: no observable action occurs during a period equal to T. We then proceed as follows:

- we define a new state $\mathbf{inconc}_\delta \in$ **Inconc**, and
- to every state $\notin$ **Pass**$\cup$**Inconc**$\cup$**Fail** without outgoing Exp-Trans (of type 1 or 3), we add a transition labeled $!\delta$ and leading to $\mathbf{inconc}_\delta$.

The use of $!\delta$ and $\mathbf{inconc}_\delta$ can be intuitively explained as follows: in a test execution if nothing happens during time T, then the verdict *Inconclusive* is generated. For the $SpecTP^{\mathrm{SE}_{\mathrm{iosa}}}_{VIS}$ of Fig. 8, we obtain the *CTG* of Fig. 9. In *CTG*, every input (resp. output) action must be interpreted as an output (resp. input) action of the tester (i.e., Test-Controller). Transition $!\delta$ in State 0 (resp. 1) is irrelevant, because it can be preempted by the only possible other transition labeled $(?\phi(m); -; x := m)$ (resp. $(?\sigma, Set_{2,3})$) which is under the control of Test-Controller. Transition $!\delta$ in State 4 indicates that nothing has happened during time T, which implies the verdict *Inconclusive*. For simplicity, **Fail** and its incoming transitions are not represented; **Fail** is implicitly reached by every non-specified transition. Note that $!\delta$ can be easily implemented by using $?Set(c_0, T)$ and $!Exp(c_0, T)$, where c_0 is a clock not used for describing timing constraints of *Spec* and *TP*.

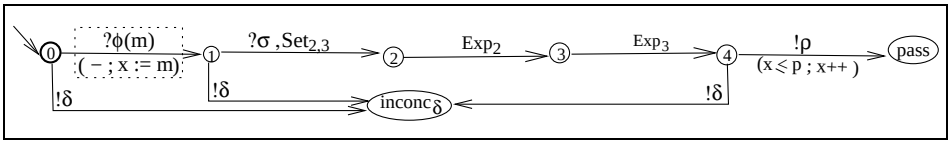

Fig. 9. Step 4: *CTG* obtained from $SpecTP^{\mathrm{SE}_{\mathrm{iosa}}}_{VIS}$ of Fig. 8)

Let $SpecTP_A$ denote the part of *SpecTP* (obtained in Step 1) that leads to a location A, and $SpecTP^{\mathrm{SE}_{\mathrm{iosa}}}_{A}$ denote the part of $SpecTP^{\mathrm{SE}_{\mathrm{iosa}}}$ (obtained in Step 2) that leads to a state A. A verdict *Pass*, *Inconclusive* or *Fail*, generated after the execution by **SUT** of a trace λ, is interpreted as follows:

Pass means that λ conforms (w.r.t. $\mathbf{conf}_{\mathrm{SE}_{\mathrm{iosa}}}$) to $SpecTP^{\mathrm{SE}_{\mathrm{iosa}}}_{A}$. From Theor. 5.2, the **IUT** has executed a timed trace μ that conforms (w.r.t. $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$) to $SpecTP_A$, and thus, to *SpecTP*. Therefore, μ conforms (w.r.t. $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$) to *Spec* (because *Spec* and *SpecTP* are observationally equivalent).

Fail means that λ does not conform (w.r.t. $\mathbf{conf}_{\mathrm{SE}_{\mathrm{iosa}}}$) to $SpecTP^{\mathrm{SE}_{\mathrm{iosa}}}$. From Theor. 5.2, the **IUT** has executed a timed trace μ that does not conform (w.r.t. $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$) to *SpecTP*, and thus, nor to *Spec* (because *Spec* and *SpecTP* are observationally equivalent).

Inconclusive means that we cannot determine whether λ conforms (w.r.t. $\mathbf{conf}_{\mathrm{SE}_{\mathrm{iosa}}}$) to $SpecTP^{\mathrm{SE}_{\mathrm{iosa}}}_{A}$. From Theor. 5.2, we cannot determine whether the **IUT** has executed a timed trace μ that conforms (w.r.t. $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$) to $SpecTP_A$.

7 Contribution and future Work

Real-time test consists of testing systems which must guarantee timing constraints. Symbolic test consists of testing systems without enumerating values of their data. The contribution of this work is the combination of those two types of testing. An advantage of our method is its simplicity because the main treatment of the real-time aspect is concentrated into one step. Our method combines in a rigorous way the method STG of symbolic testing of [23] and the method of real-time testing of [16]. Since the test method in [16] is a rigorous generalization of TGV [10] to the real-time case, we can say that our method is a rigorous generalization of STG and TGV[7] to the real-time case. We are optimistic for the applicability of our method because both TGV and STG have led to interesting software tools. But we recognize that such applicability remains to be demonstrated with real world examples.

Theoretically, the method may suffer from state explosion essentially during the synchronized product (Step 1) and the transformation *SetExp* (Step 2). But in practice, the state explosion is attenuated by the following facts:

For Step 1: *TP* is relatively simple (see comment before Fig. 5).

For Step 2: the following two numbers, that influence state explosion, are relatively small:
- the number of clocks,
- the number of values to which each clock is compared in timing constraints.

Here are some future work directions:

- Our method (as well as STG in [23]) does not support the quiescence aspect, that is used for specifying when the **IUT** is permitted to stop its execution. We intend to investigate the possibility to fill this gap.
- Our method (as well as other methods of real-time testing) does not support unobservable clock resets (Hyp. 5.1). We intend to determine conditions under which our method is applicable in the presence of unobservable clock reset.
- We intend to add the notion of invariants to $\mathrm{T}_{\mathrm{iosa}}$, in order to model actions that *must* occur (instead of being only *permitted* to occur) when they are enabled.
- Def. 3.2 is not constructive and we do not know how to compute $InpComp(S)$) from a $\mathrm{T}_{\mathrm{iosa}}$ A in the general case. We intend to determine a class of (nondeterministic) $\mathrm{T}_{\mathrm{iosa}}$s for which we can obtain a constructive

[7] Actually, STG and TGV are software tools for testing. But here, STG and TGV denote the theoretical test methods that underly the tools, respectively.

definition.

- We intend to implement a prototype of the test method in order to apply it and evaluate its complexity with non-trivial examples.

Acknowledgements

The author thanks Thierry Jéron and Vlad Rusu, from IRISA in France, for their valuable comments towards the improvement of a previous version of this manuscript.

References

[1] Alur, R., *Timed automata*, in: *Proc. 11th Intern. Conf. on Comp. Aided Verif. (CAV)* (1999), pp. 8–22.

[2] Alur, R. and D. Dill, *A theory of timed automata*, Theoretical Computer Science **126** (1994), pp. 183–235.

[3] Behrmann, G., J. Bengtsson, A. David, K. G., Larsen, P. Pettersson and W. Yi, *Uppaal implementation secrets*, in: *Proc. Form. Technique in Real-Time and Fault-Toler. Syst. (FTRTFT)* (2002), pp. 3–22.

[4] Braberman, V., M. Felder and M. Massé, *Testing timing behaviors of real time software*, in: *Proc. Quality Week 1997*, San Francisco, USA, 1997, pp. 143–155.

[5] Cardell-Oliver, R., *Conformance testing of real-time systems with timed automata*, Formal Aspects of Computing **12** (2000), pp. 350–371.

[6] Cardell-Oliver, R., *Conformance testing of real-time systems with timed automata*, in: *Nordic Workshop on Programming Theory*, 2000.

[7] Clarke, D., "Testing real-time constraints," Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania, USA (1996).

[8] Clarke, D., T. Jéron, V. Rusu and E. Zinovieva, *STG: a symbolic test generation tool*, in: *Tools and Algor. for the Const. and Anal. of Syst. (TACAS)* (2002), pp. 470–475.

[9] En-Nouaary, A., R. Dssouli, F. Khendek and A. Elqortobi, *Timed test generation based on state characterization technique*, in: *Proc. 19th IEEE Real-Time Systems Sympos. (RTSS)*, Madrid, Spain, 1998.

[10] Jard, C. and T. Jéron, *TGV : theory, principles and algorithms*, in: *Proc. 6th World Conf. on Integ. Design and Process Technol. (IDPT)*, Pasadena, California, USA, 2002.

[11] Jard, C., T. Jéron, L. Tanguy and C. Viho, *Remote testing can be as powerful as local testing*, in: *Proc. PSTV/FORTE*, Beijing, China, 1999.

[12] Jéron, T., H. Marchand, V. Rusu and V. Tschaen, *Ensuring the conformance of reactive discrete-event systems using supervisory control*, in: *42nd CDC*, Hawaii, USA, 2003.

[13] Khoumsi, A., *A method for testing the conformance of real time systems*, in: *Proc. 7th Intern. Sympos. on Formal Techn. in Real-Time and Fault Toler. Systems (FTRTFT)*, Oldenburg, Germany, 2002,
http://www.gel.usherb.ca/khoumsi/Research/Public/FTRTFT02.ps.

[14] Khoumsi, A., *Supervisory control of dense real-time discrete-event systems with partial observation*, in: *Proc. 6th Intern. Workshop on Discrete Event Systems (WODES)*, Zaragoza, Spain, 2002,
http://www.gel.usherb.ca/khoumsi/Research/Public/WODES02.ps.

[15] Khoumsi, A., *Supervisory control for the conformance of real-time discrete-event systems*, in: *Proc. 7th Intern. Workshop on Discrete Event Systems (WODES)*, Reims, France, 2004, http://www.gel.usherb.ca/khoumsi/Research/Public/WODES04.ps.

[16] Khoumsi, A., T. Jéron and H. Marchand, *Test cases generation for nondeterministic real-time systems*, in: *Proc. Formal Approaches to TEsting of Software (FATES), LNCS 2931* (2003), http://www.gel.usherb.ca/khoumsi/Research/Public/FATES03.ps.

[17] Khoumsi, A. and M. Nourelfath, *An efficient method for the supervisory control of dense real-time discrete event systems*, in: *Proc. 8th Intern. Conf. on Real-Time Computing Systems (RTCSA)*, Tokyo, Japan, 2002, http://www.gel.usherb.ca/khoumsi/Research/Public/RTCSA02-Cont.ps.

[18] Khoumsi, A. and L. Ouedraogo, *A new method for transforming timed automata*, in: *Proc. Brazilian Symposium of Formal Methods (SBMF)*, Recife, Brazil, 2004, http://www.gel.usherb.ca/khoumsi/Research/Public/SBMF04-SetExp.pdf.

[19] Krichen, M. and S. Tripakis, *Black-box conformance testing for real-time systems*, in: *Proc. Model Checking Software: 11th Int. SPIN Workshop, LNCS 2989* (2004).

[20] Nielsen, B., "Specification and test of real-time systems," Ph.D. thesis, Dept of Comput. Science, Faculty of Engin. and Sc., Aalborg University, Aalborg, Denmark (2000).

[21] Peleska, J., P. Amthor, S. Dick, O. Meyer, M. Siegel and C. Zahlten, *Testing reactive real-time systems*, in: *Proc. Mater. for the School-5th Intern. School and Sympos. on Form. Technique in Real-Time and Fault-Toler. Syst. (FTRTFT)*, Lyngby, Denmark, 1998.

[22] Rusu, V., *Verification using test generation techniques*, in: *Formal Methods Europe (FME)* (2002), pp. 252–271.

[23] Rusu, V., L. du Bousquet and T. Jéron, *An approach to symbolic test generation*, in: *Int. Conf. on Integrating Formal Methods (IFM)* (2000), pp. 338–357.

[24] Springintveld, J., F. Vaandrager and P. D'Argenio, *Testing timed automata*, Technical Report CTIT97-17, University of Twente, Amsterdam, The Netherlands (1997).

[25] Tretmans, J., "A Formal Approach to Conformance Testing," Ph.D. thesis, University of Twente, The Netherlands (1992).

[26] Tripakis, S., *Fault diagnosis for timed automata*, in: *Proc. Form. Technique in Real-Time and Fault-Toler. Syst. (FTRTFT), LNCS 2469* (2002).