On Synthesizing Test Cases in Symbolic Real-time Testing

Ahmed Khoumsi

University of Sherbrooke, Dep. GEGI, Sherbrooke J1K2R1, CANADA
Ahmed.Khoumsi@USherbrooke.ca

Abstract

*Test synthesis (or test generation) can be described as follows: from a formal specification of an implementation under test (***IUT***), and from a test purpose describing behaviors to be tested, the aim is to synthesize test cases to be executed in order to check whether the* **IUT** *conforms to its formal specification, while trying to control the* **IUT** *so that it satisfies the test purpose. In this paper, we study the synthesis of test cases for symbolic real-time systems. By* symbolic*, we mean that the specification of the* **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 testing 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

Testing is an essential step in the design of software systems, and *conformance testing* [1] is one of the most rigorous testing techniques. The objective of conformance testing is to determine whether the **IUT** *respects* a formal specification of the desired behavior of the **IUT**. The notion of *conformance relation* is used in order to define rigorously what we mean by "respects". In the sequel, the term *testing* means *conformance testing*. The main test activities consist of: *synthesizing* (or generating) test cases from the specification, and *executing* them on the **IUT**. We study both activities by proposing: a synthesis method, as well as an architecture for the execution of the synthesized test cases. Among existing work on testing, we are essentially interested by the following two complementary works:

Real-time testing (or *test of real-time systems*): the specification of the **IUT** contains *order* as well as *timing* constraints of the interactions between the **IUT** and its environment. This is the case for example of many safety-critical applications, such as patient monitoring systems and air traffic control systems. Several real-time testing methods have been developed in the last years [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12].

Symbolic testing (or *test of symbolic systems*): the specification of the **IUT** contains variables and parameters. This is the case for example of most industrial softwares. A few symbolic testing methods have been developed [13, 14, 15]. These methods aim at avoiding the synthesis 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 [16] and diagnosis [17].

This paper is motivated by the fact that each of the above two types of testing is unsatisfactory when the **IUT** is both real-time and symbolic. And our objective is indeed to propose a test synthesis method which combines the two types of testing. That is, the method to be developed can be used to synthesize test cases for real-time systems without instantiating their variables (i.e., without enumerating all the possible values of variables). We first define the model of *timed input output symbolic automata* ($\mathrm{T_{iosa}}$), that adds time to the IOSTS model of [13] and is used to model the specification of the **IUT**. We use a two-step synthesis method:

Step 1: we express the test problem into a non-real-time form, by transforming a $\mathrm{T_{iosa}}$ into an automaton called Set-Exp-IOSA ($\mathrm{SE_{iosa}}$). $SetExp$ denotes the transformation, and $SetExp(A)$ is the $\mathrm{SE_{iosa}}$ obtained by applying $SetExp$ to a $\mathrm{T_{iosa}}$ A.

Step 2: we adapt the non-real-time symbolic test method of [13].

An advantage of our method is its simplicity, due to the fact that the main treatment of the real-time aspect is concentrated into the first step. A short and incomplete version of this paper has been published in [18]. In Sect. 7 we will indicate the contributions of the present paper w.r.t. [18].

The rest of the paper is structured as follows. Sect. 2 describes the $\mathrm{T_{iosa}}$ 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 $\mathrm{SE_{iosa}}$ model and the transformation "$SetExp : \mathrm{T_{iosa}} \mapsto \mathrm{SE_{iosa}}$". In Sect. 5, we propose a test architecture. Sect. 6 presents a method based on $SetExp$ that solves the test problem. In Sect. 7, we conclude the paper. And finally, follows an appendix containing proofs of all lemmas and propositions.

2. TIMED IOSA ($\mathrm{T_{iosa}}$)

In this section, we present timed input output symbolic automata ($\mathrm{T_{iosa}}$) used to model the **IUT** and its specification. $\mathrm{T_{iosa}}$ is a combination of timed automata of [11] and input output symbolic transition systems (IOSTS) of [13].

2.1. CLOCKS AND RELATED CONCEPTS

A clock c_i is a real-valued variable that can be reset (to 0) when an action occurs 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 (possibly empty) subset of $\mathcal{H}$, and $2^{\mathcal{H}}$ is the set of clock resets.

2.2. DATA AND RELATED CONCEPTS

A variable is a data whose value can be set when an action occurs. 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 (or more briefly, a *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 $\Gamma_{\mathcal{D}}$ be the set of data guards (we consider that $True \in \Gamma_{\mathcal{D}}$).

A Variable Assignment (VA) is a (possibly empty) 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.

Let also $Type(x)$ denote the domain of definition of $x \in \mathcal{D}$.

2.3. SYNTAX OF $\mathrm{T_{iosa}}$

A $\mathrm{T_{iosa}}$ 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. There are three kinds of actions: the reception of an input, the sending of an output, and the occurrence of an internal action. In the sequel, these three kinds of actions will be abbreviated by "input", "output" and "internal action", respectively. To each input or output $a \in \Sigma$ is associated a (possibly empty) tuple $(p_1, \cdots, p_k)$ of parameters denoted θ_a. Signature of a is denoted $Sig(a)$ and defined as follows:

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

We will use the following notation for actions: an input i containing a tuple θ_i is written $?i(\theta_i)$, an output o containing a tuple θ_o is written $!o(\theta_o)$, and an internal action a (without tuple) is written ϵ_a. θ_i and θ_o are omitted when empty. Inputs and outputs are *observable*, whereas internal actions are *unobservable*.

A transition of $\mathrm{T_{iosa}}$ 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 $\mathcal{V} \cup \mathcal{C} \cup \theta_\sigma$.[1] The index σ in Z_σ means that the clock reset of a transition depends only on its action, that is, all transitions with the same event will also have the same clock reset. This restriction guarantees determinizability of $\mathrm{T_{iosa}}$ [11].

Fig. 1 illustrates the definition of $\mathrm{T_{iosa}}$ 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 "-". x, p, m are integers, $\Sigma = \{\phi, \alpha, \beta, \rho\}$, $\mathcal{H} = \{c_1\}$, $\mathcal{V} = \{x\}$, $\mathcal{C} = \{p\}$, and $\mathcal{P} = \{m\}$. ϕ cannot be an internal action because it contains parameter m, and the other actions can be of any type.

[1] 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 $\mathcal{D} = \mathcal{V} \cup \mathcal{C} \cup \mathcal{P}$

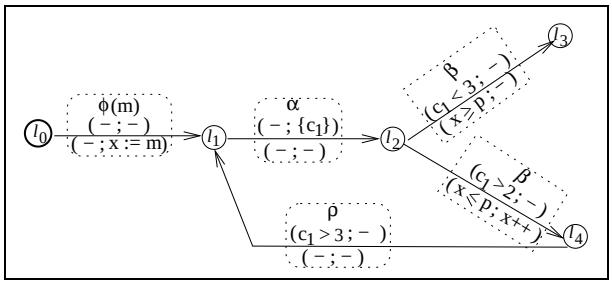

Figure 1. Example of $\mathrm{T}_{\mathrm{iosa}}$

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 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[2]; 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.

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

- The $\mathrm{T}_{\mathrm{iosa}}$ 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 $\mathrm{T}_{\mathrm{iosa}}$ reaches l_2 at the occurrence of α.
- From l_2, the $\mathrm{T}_{\mathrm{iosa}}$ 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 $\mathrm{T}_{\mathrm{iosa}}$ executes nothing.
- From l_4, the $\mathrm{T}_{\mathrm{iosa}}$ reaches l_1 at the occurrence of ρ. We have $\delta_{\alpha,\rho} > 3$.

The semantics of a $\mathrm{T}_{\mathrm{iosa}}$ 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*.

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

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* ${\lambda^t}_0$ is the empty sequence 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.

We can now introduce the notion of timed observable language of a $\mathrm{T}_{\mathrm{iosa}}$:

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

The class of $\mathrm{T}_{\mathrm{iosa}}$ that we will consider obeys to the following hypothesis:

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

Remark 2.1 *Unlike [19], with our model, consecutive actions cannot occur at the same time. We think that 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 clarify the test problem to be solved, we need 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 $\mathrm{T}_{\mathrm{iosa}}$

Let I and S denote two $\mathrm{T}_{\mathrm{iosa}}$s over the same alphabet Σ. We define the following conformance relation $I\ \mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}\ S$, where λ is a timed trace, "." stands for concatenation, o is an output action of Σ and τ is its occurrence time:

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

The intuition of "$I\ \mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}\ S$" is that after an execution of the **IUT** (modeled by I), 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 $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$, we will first define the *input-completion* of $\mathrm{T}_{\mathrm{iosa}}$. Let $\Sigma_?$ be the set of inputs of the alphabet Σ, and $Univ$ be the "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 [20, 21].

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 3.1 [3] *$I\ \mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}\ S \Leftrightarrow I\ \mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}\ InpComp(S)$.*

Lemma 3.2 [4] *If S is input-complete then: $I\ \mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}\ S \Leftrightarrow TOL_I^{\mathrm{T}_{\mathrm{iosa}}} \subseteq TOL_S^{\mathrm{T}_{\mathrm{iosa}}}$.*

Lemma 3.1 implies that we can replace a $\mathrm{T}_{\mathrm{iosa}}$ S by its input-completion before checking if a $\mathrm{T}_{\mathrm{iosa}}$ I conforms to it, w.r.t. $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$. Lemma 3.2 means that if S is input-complete, then $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$ is simplified into an inclusion of timed observable languages of $\mathrm{T}_{\mathrm{iosa}}$. Based on these two lemmas, an interesting approach would be to check $I\ \mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}\ InpComp(S)$ instead of $I\ \mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}\ S$. 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 "$I\ \mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}\ S$", we assume S input-complete.*

[3] Proof in Section A.1
[4] Proof in Section A.2

Note that Lemma 3.1 and Hyp. 3.1 are inspired from their non-real-time and non-symbolic (i.e., without clocks and data) version in [20].

Remark 3.1 *In the simple case where S has no internal action and is deterministic, its input-completion can be simply computed as follows:*

1. *A trap TL is added to S; by* trap *we mean a location such that for every action σ, TL has a self-loop transition $\langle TL; TL; \sigma; \theta_\sigma; True; \emptyset; True; \emptyset\rangle$. That is, when a trap is reached, then it is never left and every action is executable from it at any time.*
2. *For every location l and every input i of S, a non-specified transition $\langle l; TL; i; \theta_i; CG; \emptyset; DG; \emptyset\rangle$ is added to S; by* non-specified *we mean that the guards CG and DG define the domain in which i is* not *enabled in l of S.*

Therefore, Hyp. 3.1 is not restrictive when we are in the case of Remark 3.1. There exist many real examples in this case. But we agree that there are also many real examples containing internal actions. For these examples, we can try to input-complete the specification manually by using our intuition, but we have no guarantee of success. This issue is being studied presently.

3.2. TEST PURPOSE, AND TEST HYPOTHESIS

In order to define *test purpose*, let us first define the notion of *completeness*:

Definition 3.3 *A* $\mathrm{T}_{\mathrm{iosa}}$ $A = (\mathcal{L}, l_0, \mathcal{H}, \mathcal{D}, \mathcal{I}, \Sigma, \mathcal{T})$ *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* $\mathrm{T}_{\mathrm{iosa}}$ *accepts every (input, output or internal) action at any time.*

Definition 3.4 *A* test purpose *is a* $\mathrm{T}_{\mathrm{iosa}}$ *TP used to select the behaviors to be tested. By analogy with [22, 13, 11], TP is complete, deterministic, and equipped with two sets of trap*[5] *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, whereas timed sequences to be ignored are those terminating in or traversing a location R.*

In the above Def. 3.4, by *complete*, we mean that $TestPurp$ accepts every (input, output, and internal) action at any time. A test purpose should be 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 (e.g., scenarios, properties) of the specification. Ideally, a test purpose should correspond exactly to what the user has in mind to test. Generally, this intention is defined by scenarios (i.e., executions) or by properties (i.e.

[5] The notion of *trap* has been defined in Remark 3.1

formulas, e.g. in temporal logic). We have selected $\mathrm{T_{iosa}}$ to describe test purposes; this model gives enough expressiveness for describing test purposes defined by scenarios with timing constraints and variables. For test purposes defined by a property, we will need to construct the $\mathrm{T_{iosa}}$ that allows to check the given property. This process is in general iterative: a first $\mathrm{T_{iosa}}$ is constructed grossly and is refined repeatedly.

We will also use the following *test hypothesis* inspired from [23]:

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

We think that Hyp. 3.2 is realistic because the model of $\mathrm{T_{iosa}}$ is sufficiently rich for modeling many real-time discrete event systems using parameters.

3.3. Clarification of the test problem

We can now state our objective: Given two $\mathrm{T_{iosa}}$s $Spec$ and TP over the same alphabet, modeling the specification and the test purpose respectively, the aim is to synthesize an automaton CTG (Complete Test Graph) and then to extract test cases from CTG.

The test cases are intended to be executed on the **IUT** in order to check 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 $Spec$ leading to locations A of TP.

The test system takes into account TP by *ignoring* every execution λ of the **IUT** accepted by $Spec$ (i.e., $\lambda \in TOL^{\mathrm{T_{iosa}}}_{\mathcal{IUT}} \cap TOL^{\mathrm{T_{iosa}}}_{Spec}$) 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 $\mathrm{SE_{iosa}}$

Our test problem will be solved in Sect. 6 by using a transformation, called $SetExp$, that is described in detail in [24] and applied in [10, 11, 25, 26, 27]. In these references, $SetExp$ 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. Their semantics is taken into account when using (interpreting, processing, ...) the automaton called $\mathrm{SE_{iosa}}$ that results from $SetExp$. In this Section, we present the $\mathrm{SE_{iosa}}$ model and illustrate $SetExp$ by an example. Let A be a $\mathrm{T_{iosa}}$ over an alphabet Σ and $SetExp(A)$ be the $\mathrm{SE_{iosa}}$ obtained by applying $SetExp$ to A.

4.1. Actions Set and Exp

$Set(c_i, k)$ means: clock c_i is reset (to 0) and will expire when c_i evaluates to k. And
$Set(c_i, k_1, k_2, \cdots, k_p)$, $k_1 < k_2 < \cdots < k_p$, 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.

$Exp(c_i, k)$ means: clock c_i evaluates to k and thus expires.

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 that c evaluates to k. When applied to a $\mathrm{T_{iosa}}$ A, $SetExp$ is realized in two steps as follows:

Step 1 : To replace each clock reset in A by the appropriate Set action.

Step 2 : To construct a finite state automaton, denoted $SetExp(A)$, that accepts sequences containing actions of A and Set actions obtained in Step 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 illustrate $SetExp$ by a trivial example, 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 so that it occurs after two time units, and the task must be terminated before the alarm. Clearly, these two specifications define the same timing constraint. Intuitively, $SetExp$ generates 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 $\mathrm{SE_{iosa}}$

We have seen in Sect. 2 that a transition of $\mathrm{T_{iosa}}$ is defined by $\langle q; r; \sigma; \theta_\sigma; CG; Z_\sigma; DG; VA\rangle$ and is 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 be an action of

the alphabet Σ of the $\mathrm{T}_{\mathrm{iosa}}$ A with its parameters, $\mathcal{S}$ (resp. $\mathcal{E}$) be 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. $\mathcal{E}$). Transitions of the $\mathrm{SE}_{\mathrm{iosa}}$ $SetExp(A)$ can be categorized into three types as follows:

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

Type 2 : a 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 the $\mathrm{SE}_{\mathrm{iosa}}$ $SetExp(A)$, 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 the $\mathrm{SE}_{\mathrm{iosa}}$ $SetExp(A)$ 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 4.1 *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. TWO EXAMPLES OF APPLICATION OF $SetExp$: $\mathrm{T}_{\mathrm{iosa}} \mapsto \mathrm{SE}_{\mathrm{iosa}}$

4.4.1. Example 1: We illustrate here $SetExp$ by an example without data. We consider the specification: $1 \leq \delta_{a,b} < 3$, where $\delta_{a,b}$ is the delay between actions a and b. In a $\mathrm{T}_{\mathrm{iosa}}$, such a constraint is expressed by: 1) using two transitions Tr1 and Tr2 that represent the occurrences of a and b, respectively; 2) resetting a clock c at the occurrence of Tr1; and 3) associating to Tr2 the clock guard (CG): $((c \geq 1) \wedge (c < 3))$. This timing constraint can be expressed differently as follows: i) the reset "$c := 0$" of Tr1 is replaced by a $Set(c, 1, 3)$ (which will be followed by $Exp(c, 1)$ and $Exp(c, 3)$), and ii) the CG "$((c \geq 1) \wedge (c < 3))$" of Tr2 becomes "Tr2 occurs after or simultaneously to $Exp(c, 1)$ and before $Exp(c, 3)$". This timing constraint will be represented in a $\mathrm{SE}_{\mathrm{iosa}}$ by the following two sequences, where consecutive actions are separated by "$\cdot$" and simultaneous actions are grouped in "$\langle\rangle$".):

- "$\langle a, Set(c, 1, 3)\rangle \cdot Exp(c, 1) \cdot b \cdot Exp(c, 3)$", i.e., Tr2 occurs *after* $Exp(c, 1)$.
- "$\langle a, Set(c, 1, 3)\rangle \cdot \langle Exp(c, 1), b\rangle \cdot Exp(c, 3)$", i.e., Tr2 occurs *simultaneously to* $Exp(c, 1)$.

4.4.2. Example 2: 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 aspect is irrelevant for the comprehension of $SetExp$.

Remark 4.2 *Clocks are real-valued variables although they are compared to (nonnegative) integers, the latter being considered just as particular reals.* $SetExp$ *remains applicable if clocks are compared to reals.*

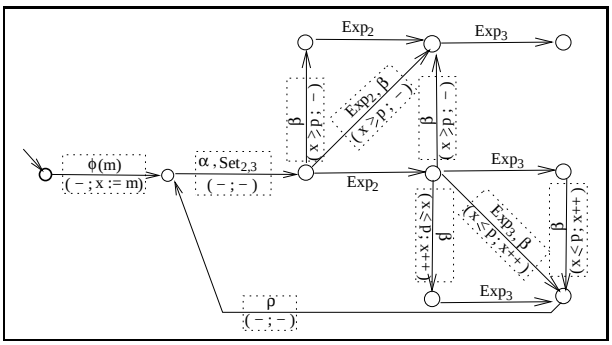

Figure 2. $\mathrm{SE}_{\mathrm{iosa}}$ $SetExp(A)$ obtained from the $\mathrm{T}_{\mathrm{iosa}}$ A of Fig. 1

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 $B = SetExp(A)$ 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 (see Sect. 2.3). A transition of B is syntactically defined by TR = $\langle q; r; \mu; DG; VA\rangle$, where: q and r are origin and destination states; μ consists of 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 $\mathrm{SE}_{\mathrm{iosa}}$

Initially, the $\mathrm{SE}_{\mathrm{iosa}}$ $B = (\mathcal{Q}, q_0, \mathcal{D}, \mathcal{I}, \Lambda, \Psi)$ is at state q_0 with all clocks of $\mathcal{H}$ equal to 0, and variables and constants taking values such that $\mathcal{I}$[6] evaluates to $true$. A transition 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 *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. The semantics of a $\mathrm{SE}_{\mathrm{iosa}}$ $B = (\mathcal{Q}, q_0, \mathcal{D}, \mathcal{I}, \Lambda, \Psi)$ can also be defined 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*:

- either 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 4.2 *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.*

Note that $OL_B^{\mathrm{SE}_{\mathrm{iosa}}}$ implicitly respects the following **Consistency condition***: every $Set(c,k)$ and its corresponding $Exp(c,k)$ are effectively separated by time k.*

We define the following conformance relation $\mathbf{conf}_{\mathrm{SE}_{\mathrm{iosa}}}$ relating two $\mathrm{SE}_{\mathrm{iosa}}$s:

Definition 4.3 *Let I' and S' be two* $\mathrm{SE}_{\mathrm{iosa}}$*s over the same alphabet:* $I'\,\mathbf{conf}_{\mathrm{SE}_{\mathrm{iosa}}}\,S' \equiv (OL_{I'}^{\mathrm{SE}_{\mathrm{iosa}}} \subseteq OL_{S'}^{\mathrm{SE}_{\mathrm{iosa}}})$.

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 proposition of equivalence:

Proposition 4.1 [7]
$RmvSetExp(AddTime(OL_{SetExp(A)}^{\mathrm{SE}_{\mathrm{iosa}}})) = TOL_A^{\mathrm{T}_{\mathrm{iosa}}}$.

Intuitively, Proposition 4.1 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 PROPOSITION

Given two $\mathrm{T}_{\mathrm{iosa}}$s $Spec$ and TP over the same alphabet, we have clarified in Sect. 3.3 that our objective is to synthesize an automaton CTG (Complete Test Graph) from which test cases are extracted. The latter are intended to be executed in order to study the conformance of the **IUT** to the part of $Spec$ corresponding to TP. CTG will not be directly computed on the $\mathrm{T}_{\mathrm{iosa}}$s $Spec$ and TP, but rather on a $\mathrm{SE}_{\mathrm{iosa}}$ computed from the two $\mathrm{T}_{\mathrm{iosa}}$s. In order to make the link between CTG and the **IUT**, we use the test architecture represented in Fig. 3 and proposed in [11]. It comprises the **IUT**, a Tester, and a Clock-Handler that mimics the timing aspect of the **IUT**. More precisely, we have:

Clock-Handler receives Set actions from the Tester and sends Exp actions to the Tester. It respects the consistency condition (see end of Sect. 4.6). It can be seen as a Timer module that upon the reception of a Set action, activates a timer and sends back to the Tester the corresponding Exp action when the timer elapses. Note that Clock-Handler guarantees the consistency condition, i.e., $Set(c,k)$ and the corresponding $Exp(c,k)$ are separated by time k.

[6] $\mathcal{H}$ is the set of clocks of the $\mathrm{T}_{\mathrm{iosa}}$ A such that $B = SetExp(A)$.

[7] Proof in Section C

Tester executes test cases that are derived from a $\mathrm{SE}_{\mathrm{iosa}}$ and is tagged with the Set and Exp actions of this $\mathrm{SE}_{\mathrm{iosa}}$. It sends the inputs and receives the outputs of the **IUT**, it also sends Set actions to Clock-Handler and receives Exp actions from Clock-Handler. The timing constraints that the Tester has to deal with are performed via its interaction with the Clock-Handler module.

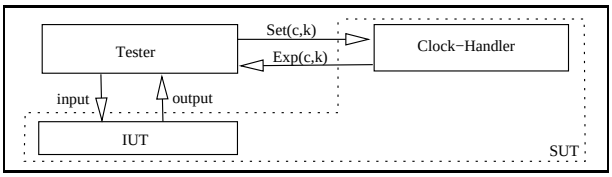

Figure 3. Test architecture

Here are a few necessary notations:

Notation 5.1 *If L is a language, then $\overline{L}$ denotes the prefix of L. That is, $\overline{L}$ contains every* finite *sequence that is a prefix of a sequence contained in L.*

Notation 5.2 *$Tester \preceq K$ means that during a test execution, the Tester generates only Set actions that are accepted by the $\mathrm{SE}_{\mathrm{iosa}}$ K. More formally, it means:*
$\forall \lambda \in \overline{OL_K^{\mathrm{SE}_{\mathrm{iosa}}}}$, $\forall U$ action of **IUT**, *$\forall \mathcal{S}$ set of Set actions: after the execution of λ, the Tester generates $\mathcal{S}$ simultaneously to U (if any)* iff $\lambda \cdot (U, \mathcal{S}) \in \overline{OL_K^{\mathrm{SE}_{\mathrm{iosa}}}}$.

We can now state the next proposition which makes the link between $\mathbf{conf}_{\mathrm{SE}_{\mathrm{iosa}}}$ (relating two $\mathrm{SE}_{\mathrm{iosa}}$s) and the real-time conformance relation $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$ (relating two $\mathrm{T}_{\mathrm{iosa}}$s), where **SUT** (System Under Test) consists of **IUT** and Clock-Handler, $\mathcal{IUT}$ is the $\mathrm{T}_{\mathrm{iosa}}$ modeling **IUT**, $\mathcal{SUT}$ is the $\mathrm{SE}_{\mathrm{iosa}}$ modeling **SUT**, and S is a $\mathrm{T}_{\mathrm{iosa}}$:

Proposition 5.1 *$Tester \preceq SetExp(S)$ implies:*
$\mathcal{IUT}$ $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$ $S \Leftrightarrow$ ($\exists$ $\mathrm{SE}_{\mathrm{iosa}}$ $\mathcal{SUT}$ accepting behavior of **SUT** *s.t. $\mathcal{SUT}$ $\mathbf{conf}_{\mathrm{SE}_{\mathrm{iosa}}}$ $SetExp(S)$.*

The above proposition 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) adapt a non-real-time method of Symbolic Test Generation (STG) [13].

Here is a simple example that gives the intuition of Prop. 5.1. S specifies that a task $\mathcal{T}$ is realized in less than two units of time. $SetExp(S)$ specifies that: i) at the beginning of $\mathcal{T}$ an alarm is programmed so that it occurs after two units of time, and ii) $\mathcal{T}$ is terminated before the alarm. The programming (resp. occurrence) of the alarm corresponds to a Set (resp. Exp) action. Tester orders the **IUT** to start $\mathcal{T}$ and, simultaneously, programs the alarm by sending a $Set(c, 2)$ to Clock-Handler. Tester deduces that **IUT** has conformed to S *iff* it receives $Exp(c, 2)$ from Clock-Handler *after* it receives from the **IUT** the indication that $\mathcal{T}$ is terminated.

The proposed architecture is applicable only if transitions executing internal (i.e., unobservable) actions do not reset clocks. In fact, in order to generate Set actions, the Tester needs to observe every action to which is associated a clock reset. Hence the following hypothesis meaning that there is no timing constraint relatively to unobservable actions:

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

We argue that there exist many real examples respecting Hyp. 5.1, because in many cases, timing constraints that interest the user of **IUT** are defined between actions that (s)he observes.

6. METHOD OF TEST GENERATION

Let us propose a test method that can be used to synthesize test cases for real-time systems without enumerating all the possible values of their variables. The proposed method combines, and thus extends, two complementary test methods: 1) a test method applicable to (non-symbolic) real-time systems [11], and 2) a test method applicable to (non-real-time) symbolic systems [13]. It consists of five steps outlined in Fig. 4 and described in subsections 6.1 to 6.5. Its inputs are $Spec$ (input-complete, from Lemma 3.1 and Hyp. 3.1) and TP (complete, from Def. 3.4). In a first step, we compute a $\mathrm{T}_{\mathrm{iosa}}$ $SpecTP$ that 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 test cases are extracted in Step 5. Test cases are intended to be executed on the **IUT** in order to check 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. 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) [13].[8] Step 5 is inspired from [11].

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

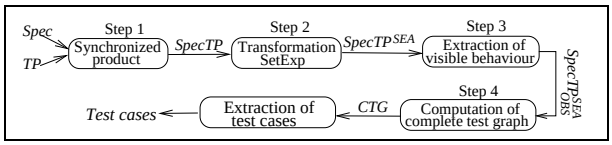

Figure 4. Steps of the test method

$Spec$ and TP of Figure 5 will be used to illustrate the five steps of the test method. These two $\mathrm{T}_{\mathrm{iosa}}$ are defined over the alphabet $\Sigma = \{?\phi, ?\sigma, !\rho, \epsilon_a, \epsilon_b\}$. 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. Recall that input-completion of $Spec$ is justified by Lemma 3.1, and that we do not know how to compute it in the general case (Def. 3.2 is not constructive). In the particular example of Fig. 5, input-completion of $Spec$ can be computed using Remark 3.1, although $Spec$ contains internal actions. 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 intend to test executions of $Spec$ terminating by the first occurrence of $!\rho$ in $Spec$ (i.e. without traversing Location TL). This example of TP is taken very simple (with one parameter and *no* timing constraint) in order to clarify the operations of the different steps. Recall that generally, 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 (e.g., scenarios, properties) of the specification. A simple test purpose defined by scenarios can be easily modeled by $\mathrm{T}_{\mathrm{iosa}}$. In the presence of a test purpose defined by a property P, we need to transform P into a $\mathrm{T}_{\mathrm{iosa}}$ in an iterative way: a first $\mathrm{T}_{\mathrm{iosa}}$ is constructed grossly and is refined repeatedly.

6.1. Step 1 : Compute the synchronous product of $Spec$ and TP

We compute a $\mathrm{T}_{\mathrm{iosa}}$ $SpecTP$ that is observationally equivalent to $Spec$ (i.e., $TOL_{Spec}^{\mathrm{T}_{\mathrm{iosa}}} = TOL_{SpecTP}^{\mathrm{T}_{\mathrm{iosa}}}$), but $SpecTP$ contains locations indicated by A (resp. R) that correspond to locations A (resp. R) of TP. For that purpose, we need to define the synchronized product of two $\mathrm{T}_{\mathrm{iosa}}$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 $\mathrm{T}_{\mathrm{iosa}}$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 [28] and the synchronized product of IOSTS [13]. $A_1 \otimes A_2$ is defined *iff* the following four conditions are satisfied:

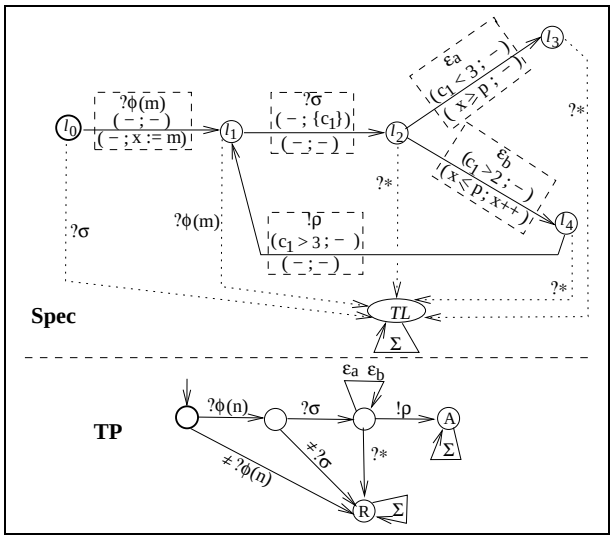

Figure 5. Example for illustrating the test method

1. $\Sigma^1 = \Sigma^2$. The common alphabet will then be denoted Σ. This condition can be easily relaxed [13], but we will keep it for simplicity.
2. $\mathcal{H}^1 \cap \mathcal{H}^2 = \emptyset$ [28].
3. $(\mathcal{V}^1 \cup \mathcal{P}^1) \cap (\mathcal{V}^2 \cup \mathcal{P}^2) = \emptyset$, $\mathcal{C}^1 \cap \mathcal{P}^2 = \emptyset$, and $\mathcal{C}^2 \cap \mathcal{P}^1 = \emptyset$ [13].
4. Each action $a \in \Sigma$ has the same signature in A^1 and A^2 [13].

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 exists 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 exists a transition $\langle (q^1; q^2); (r^1; r^2); \sigma; \theta_\sigma^1; CG^1 \wedge CG^2; Z_\sigma^1 \cup Z_\sigma^2; DG^1 \wedge DG^{1,2}; VA^1 \cup VA^{1,2}\rangle \in \mathcal{T}$.

Note that we can also proceed symmetrically by defining $DG^{2,1}$ and $VA^{2,1}$, instead of $DG^{1,2}$ and $VA^{1,2}$.

This procedure is inspired from [13].

In Step 1, we compute $SpecTP = Spec \otimes TP$, from which we remove the (unreachable) locations without incoming transitions.

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 $Spec$. The symmetrical approach consists of removing m, instead of n.

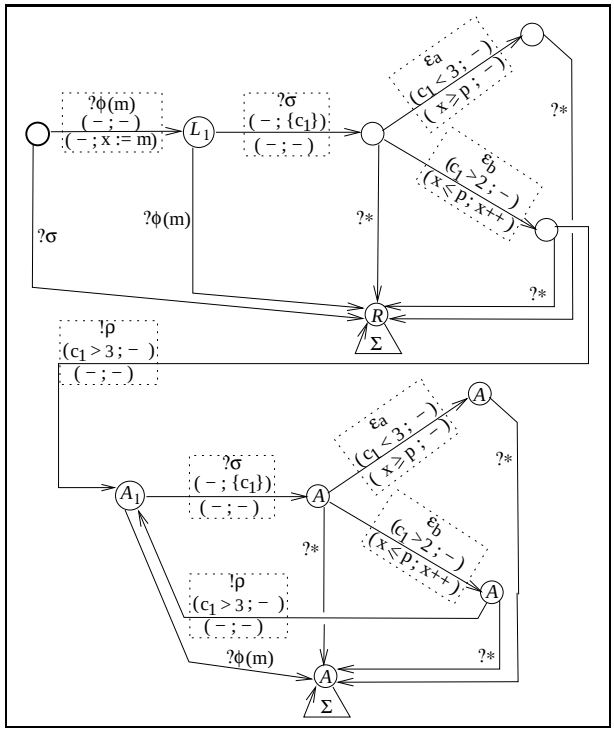

Figure 6. Step 1: $SpecTP$ obtained from $Spec$ and TP of Fig. 5

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: $?*$ denotes any input (i.e., $?\phi(m)$ or $?\sigma$); Σ means any action $x \in \Sigma = \{?\phi(m), ?\sigma, !\rho, \epsilon_a, \epsilon_b\}$; $\mathrm{Set_{2,3}}$ denotes $?Set(c_1, 2, 3)$; $\mathrm{Exp_i}$ denotes $!Exp(c_1, i)$ for $i = 2, 3$; $(\mathrm{Exp_i}, \Sigma)$ means the simultaneous occurrence of $\mathrm{Exp_i}$ and any $x \in \Sigma$; nodes linked by a dotted line correspond to the same location[9]; and states that correspond to location A (resp. R) of $SpecTP$ are indicated by A (resp. R). State A_1 is equivalent to State S_1 with the difference that S_1 does not correspond to a location A of TP. We have not represented the states reachable from A_1 because the sequences to be tested are those terminating in and not traversing a state A. In Fig. 7 and subsequent figures, if DG evaluates to $true$ and VA is empty in a transition (of Type 2 or 3), then $(DG; VA)$ is not represented.

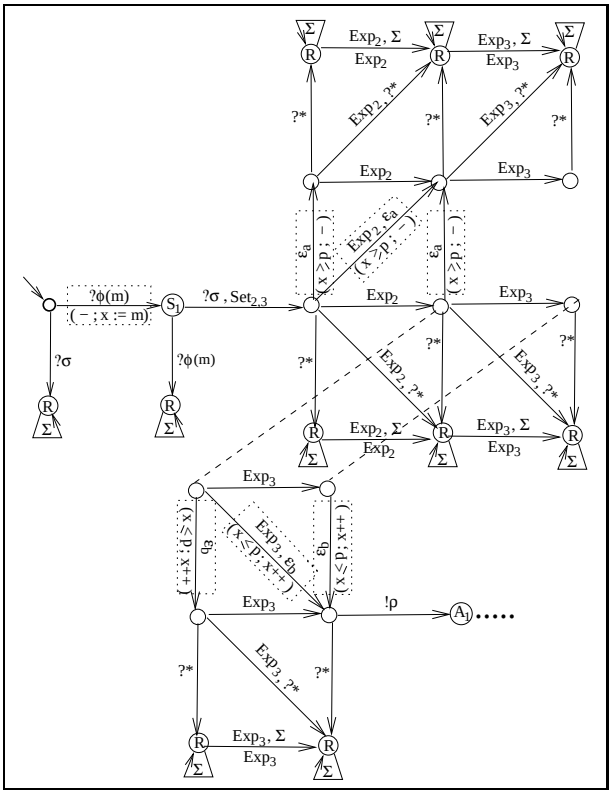

Figure 7. Step 2: $SpecTP^{\mathrm{SE_{iosa}}}$ obtained from $SpecTP$ of Fig. 6

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

We construct the observable 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 [13]. The result is denoted $Obs(SpecTP^{\mathrm{SE_{iosa}}})$. The adaptation consists of a preliminary step where internal actions in transitions of Type 3 are simply erased. After that,

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

we can use the procedure of [13] because the remaining internal actions are "alone" in their transitions (of type 2). (Recall that we consider only the case where internal actions do not reset clocks.)

Substep 3b : $Obs(SpecTP^{\mathrm{SE_{iosa}}})$ is determinized by using a heuristic proposed in [13]. The result is denoted $Det(Obs(SpecTP^{\mathrm{SE_{iosa}}}))$.

Substep 3c : Note that every state of $Det(Obs(SpecTP^{\mathrm{SE_{iosa}}}))$ corresponds to one or several states of $SpecTP^{\mathrm{SE_{iosa}}}$. States R and A of $Det(Obs(SpecTP^{\mathrm{SE_{iosa}}}))$ 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 call A every state corresponding exclusively 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 $SpecTP^{\mathrm{SE_{iosa}}}_{OBS}$.

For the $SpecTP^{\mathrm{SE_{iosa}}}$ of Fig. 7, after Substep 3a, we obtain $Obs(SpecTP^{\mathrm{SE_{iosa}}})$ of Fig. 8 where Σ_o means any observable action $x \in \{?\sigma, ?\phi(m), !\rho\}$; and after Substep 3c, we obtain $SpecTP^{\mathrm{SE_{iosa}}}_{OBS}$ of Fig. 9.

6.4. STEP 4 : COMPUTING A COMPLETE TEST GRAPH (CTG)

Recall that a transition of $\mathrm{SE_{iosa}}$ can be labeled as follows: $(\mathcal{E})$, (σ), $(\sigma, \mathcal{S})$, $(\mathcal{E}, \sigma)$, or $(\mathcal{E}, \sigma, \mathcal{S})$, (in addition to $(DG;\ VA)$). Let:

output transition be any transition labeled in one of the five forms and such that σ is an output of the **IUT**;

input transition be any transition labeled (σ) or $(\sigma, \mathcal{S})$ and such that σ is an input of the **IUT**;

mixed transition be any transition labeled $(\mathcal{E}, \sigma)$ or $(\mathcal{E}, \sigma, \mathcal{S})$ and such that σ is an input of the **IUT**.

We construct a Complete Test Graph (CTG) in a way inspired (but different) from [22, 11, 13] as follows:

- Let $L2A$ be the set of states of $SpecTP^{\mathrm{SE_{iosa}}}_{OBS}$ that are co-accessible to a location A, i.e., from which a state A is accessible.
- Let $\mathbf{Pass}$ be the set of states A of $SpecTP^{\mathrm{SE_{iosa}}}_{OBS}$.
- Let $\mathbf{Fail} = \{\mathbf{fail}\}$ consist of a new state that is reached by every non-specified output transition of $SpecTP^{\mathrm{SE_{iosa}}}_{OBS}$ executable from $L2A$.

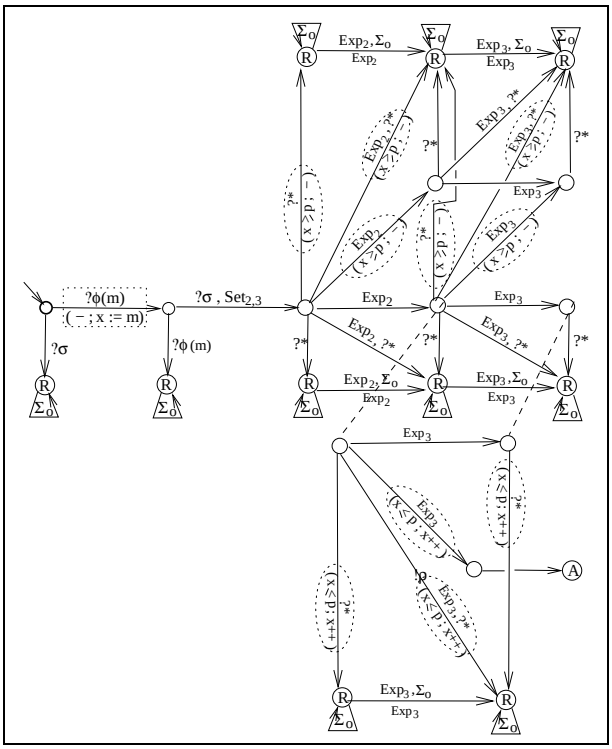

Figure 8. Step 3a: After elimination of internal actions from $SpecTP^{\mathrm{SE_{iosa}}}$ of Fig. 7

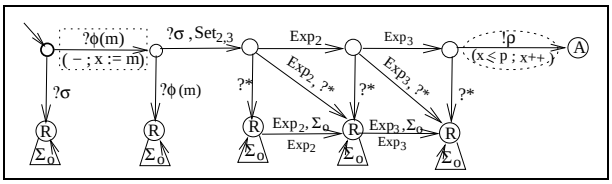

Figure 9. Step 3: $SpecTP^{\mathrm{SE_{iosa}}}_{OBS}$ obtained from $SpecTP^{\mathrm{SE_{iosa}}}$ of Fig. 7

- Let $\mathbf{Inconc}$ be the set of states of $SpecTP^{\mathrm{SE_{iosa}}}_{OBS}$ that are not in $L2A \cup \mathbf{Pass}$ and are accessible from $L2A$ by a single output transition of $SpecTP^{\mathrm{SE_{iosa}}}_{OBS}$.
- We then obtain CTG from $SpecTP^{\mathrm{SE_{iosa}}}_{OBS}$ by:
  - adding (implicitly) state $\mathbf{Fail}$ and its incoming (non-specified output) transitions,
  - removing every state $\notin L2A \cup \mathbf{Pass} \cup \mathbf{Inconc} \cup \mathbf{Fail}$, and
  - removing outgoing transitions of every state $\in \mathbf{Pass} \cup \mathbf{Inconc}$.

To synthesize test cases executable in acceptable time (that is, to avoid that Tester waits for an output of the **IUT** 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 $\mathbf{inconc}_\delta \in \mathbf{Inconc}$, and
- to every state $\notin \mathbf{Pass} \cup \mathbf{Inconc} \cup \mathbf{Fail}$ in which only output transitions of type 2 can be executed, 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 $SpecTP_{OBS}^{\mathrm{SE}_{\mathrm{iosa}}}$ of Fig. 9, we obtain the CTG of Fig. 10. 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.

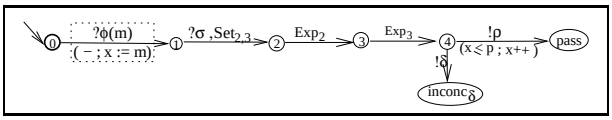

Figure 10. Step 4: CTG obtained from $SpecTP_{OBS}^{\mathrm{SE}_{\mathrm{iosa}}}$ of Fig. 9

Correctness of our construction of CTG is stated by the following three lemmas:

Lemma 6.1 [10] *When the Tester observes a trace λ of* **SUT** *that leads to a state $p \in \mathbf{Pass}$, then the* **IUT** *has executed a timed trace μ that conforms to $Spec$ w.r.t. $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$ (i.e., $\mu \in TOL_{Spec}^{\mathrm{T}_{\mathrm{iosa}}}$) and that leads to a location A of TP.*

Lemma 6.2 [11] *When the Tester observes a trace λ of* **SUT** *that leads to the state $fail$, then the* **IUT** *has executed a timed trace μ that does not conform to $Spec$ w.r.t. $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$ (i.e., $\mu \notin TOL_{Spec}^{\mathrm{T}_{\mathrm{iosa}}}$).*

Lemma 6.3 [12] *When the Tester observes a trace λ of* **SUT** *that leads to the state $x \in \mathbf{Inconc}$, then the* **IUT** *has executed a timed trace μ that conforms to $Spec$ w.r.t. $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$ but no location A of TP can be reached after μ.*

6.5. Step 5 : Extracting Test Cases from CTG

The objective of Step 5 is to extract so-called *controllable subgraphs* of CTG. For that purpose, let us use the following hypothesis:

[10] Proof in Section D.1
[11] Proof in Section D.2
[12] Proof in Section D.3

Hypothesis 6.1 *When desired, the Tester is capable of reacting more promptly than the* **SUT** *in all situations where both are allowed to send an action to each other.*

Hyp 6.1 is reasonable when the **SUT** is a system with very high computing ressources and a very high clock frequency. Assuming this hypothesis, controllable subgraphs can be extracted from CTG by executing one the following three options for each state of CTG:

- One input transition is kept and all other (input, output, and mixed) transitions are pruned. That is, the Tester sends a given input to the **SUT**, before the latter has the time to generate an output.
- All output transitions are kept, and all other (input and mixed) transitions are pruned. That is, the Tester sends no input and waits for the reception of any possible output from the **SUT**.
- One mixed transition T is kept with all the outputs transitions that have not the same $\mathcal{E}$ as T, and all other transitions are pruned. That is, the Tester waits for the reception of a given set of expirations $\mathcal{E}_1$ with the objective to send a given input to the **SUT** simultaneously to $\mathcal{E}_1$. The input is not sent if Tester receives an output or another $\mathcal{E}$ from the **SUT**.

Note that this procedure is more complex than procedures in [22, 11] that have inspired us. For the CTG of Fig. 10, we obtain a single controllable subgraph: CTG itself.

7. Contribution and Future Work

We have proposed a test method that combines two types of testing: real-time testing that consists of testing systems with timing constraints; and symbolic test that consists of testing systems without enumerating values of their data. More precisely, our method combines and extends in a rigorous way the method STG of symbolic testing of [13] and the method of real-time testing of [11]. An advantage of our method is its simplicity because the main treatment of the real-time aspect is concentrated into one step. Since the test method in [11] is a rigorous generalization of TGV [22] to the real-time case, we can say that our method is a rigorous generalization of STG and TGV[13] 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.

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

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.

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.

A previous version of his article has been published in [18]. Here are the main contributions of the present paper w.r.t. [18]:

1. All lemmas and propositions are rigorously proved (in the Appendix).
2. New lemmas (6.1, 6.2, 6.3), that state correctness of the test method, are added and proved.
3. The test method contains an additional (fifth) step that extracts test cases from the synthesized Complete Test Graph.
4. Proposition 5.1 is expressed more formally and proved; this proposition is the basis for transforming the test problem into a non-real-time form.
5. The notion of *test purpose* is presented and explained in more detail.
6. A few errors have been corrected.

Here are some future work directions:

- Our method (as well as STG in [13]) 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. 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 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 have explained how to compute $InpComp(S)$ when S has no internal action and is deterministic. We intend to determine a more general class of $\mathrm{T}_{\mathrm{iosa}}$s for which we can construct their input-completion.
- We intend to implement a prototype of the test method in order to study it with real world examples.

References

[1] ISO/IEC. *International Standard 9646-1/2/3, OSI-Open Systems Interconnection, Information Technology - Open Systems Interconnection Conformance Testing Methodology and Framework*, 1992.

[2] D. Clarke. *Testing real-time constraints.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, USA, 1996.

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

[4] V. Braberman, M. Felder, and M. Massé. Testing timing behaviors of real time software. In *Proc. Quality Week 1997*, pages 143–155, San Francisco, USA, April-May 1997.

[5] J. Peleska, 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.

[6] 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 Sympos. (RTSS)*, Madrid, Spain, December 1998.

[7] B. Nielsen. *Specification and test of real-time systems*. PhD thesis, Dept of Comput. Science, Faculty of Engin. and Sc., Aalborg University, Aalborg, Denmark, 2000.

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

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

[10] A. Khoumsi. 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, September 2002.

[11] A. Khoumsi, T. Jéron, and H. Marchand. Test cases generation for nondeterministic real-time systems.

In *Proc. Formal Approaches to TEsting of Software (FATES), LNCS 2931*, Montreal, Canada, October 2003. Springer.

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

[13] V. Rusu, L. du Bousquet, and T. Jéron. An approach to symbolic test generation. In *Int. Conf. on Integrating Formal Methods (IFM)*, pages 338–357, Dagstuhl, Germany, 2000. LNCS 1945.

[14] V. Rusu. Verification using test generation techniques. In *Formal Methods Europe (FME)*, pages 252–271. LNCS 2391, 2002.

[15] D. Clarke, Thierry Jéron, V. Rusu, and E. Zinovieva. STG: a symbolic test generation tool. In *Tools and Algor. for the Const. and Anal. of Syst. (TACAS)*, pages 470–475. LNCS 2280, 2002.

[16] G. Behrmann, 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)*, pages 3–22. Springer-Verlag, Sept. 2002.

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

[18] A. Khoumsi. Complete test graph generation for symbolic real-time systems. In *Proc. Brazilian Symposium of Formal Methods (SBMF)*, Recife, Brazil, November 2004. Best Paper Award.

[19] R. Alur. Timed automata. In *Proc. 11th Intern. Conf. on Comp. Aided Verif. (CAV)*, pages 8–22. Springer-Verlag LNCS 1633, 1999.

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

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

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

[23] J. Tretmans. *A Formal Approach to Conformance Testing*. PhD thesis, University of Twente, The Netherlands, December 1992.

[24] A. Khoumsi and L. Ouedraogo. A new method for transforming timed automata. In *Proc. Brazilian Symposium of Formal Methods (SBMF)*, Recife, Brazil, November 2004.

[25] A. Khoumsi 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, March 2002.

[26] A. Khoumsi. Supervisory control of dense real-time discrete-event systems with partial observation. In *Proc. 6th Intern. Workshop on Discrete Event Systems (WODES)*, Zaragoza, Spain, October 2002.

[27] A. Khoumsi. Supervisory control for the conformance of real-time discrete-event systems. In *Proc. 7th Intern. Workshop on Discrete Event Systems (WODES)*, Reims, France, September 2004.

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

A. PROOFS OF LEMMAS 3.1 AND 3.2

A.1. PROOF OF LEMMA 3.1:

$(I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ S) \Leftrightarrow (I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ InpComp(S))$

A.1.1. Proof of:

$(I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ S) \Rightarrow (I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ InpComp(S))$**:**

1. We assume $(I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ S)$, that is (from Def. 3.1), for any output o and $\forall \lambda \in TOL_S^{\mathrm{T_{iosa}}}$: $(\lambda{\cdot}(o,\tau) \in TOL_I^{\mathrm{T_{iosa}}}) \Rightarrow (\lambda{\cdot}(o,\tau) \in TOL_S^{\mathrm{T_{iosa}}})$.

2. $\lambda \in TOL_S^{\mathrm{T_{iosa}}}$ implies: $\lambda{\cdot}(o,\tau) \in TOL_S^{\mathrm{T_{iosa}}} \Leftrightarrow \lambda{\cdot}(o,\tau) \in TOL_{InpComp(S)}^{\mathrm{T_{iosa}}}$, because only inputs (and not outputs) are added to locations of S when $InpComp$ operator is applied to S.

3. Items 1 and 2 imply that $\forall \lambda \in TOL_S^{\mathrm{T_{iosa}}}$: $\lambda{\cdot}(o,\tau) \in TOL_I^{\mathrm{T_{iosa}}} \Rightarrow \lambda{\cdot}(o,\tau) \in TOL_{InpComp(S)}^{\mathrm{T_{iosa}}}$.

4. Let τ_λ be the time of the last (observable) event of λ. If $\lambda \in TOL_{InpComp(S)}^{\mathrm{T_{iosa}}} \setminus TOL_S^{\mathrm{T_{iosa}}}$ then, from Def. 3.2, $\lambda = w \cdot a \cdot x$ such that: $a \in \Sigma_?, w \cdot a \notin TOL_A^{\mathrm{T_{iosa}}}$, $x \in TOL_{Univ}^{\mathrm{T_{iosa}}}$. Therefore: $(\tau > \tau_\lambda) \Leftrightarrow$ $(\lambda{\cdot}(o,\tau) = w{\cdot}a{\cdot}y \in TOL_{InpComp(S)}^{\mathrm{T_{iosa}}} \setminus TOL_S^{\mathrm{T_{iosa}}})$, where $y = x{\cdot}(o,\tau) \in TOL_{Univ}^{\mathrm{T_{iosa}}}$.

5. $(\lambda{\cdot}(o,\tau) \in TOL_I^{\mathrm{T_{iosa}}}) \Rightarrow (\tau > \tau_\lambda)$.
6. Items 4 and 5 imply that $\forall \lambda \in TOL_{InpComp(S)}^{\mathrm{T_{iosa}}} \setminus TOL_S^{\mathrm{T_{iosa}}}$: $(\lambda{\cdot}(o,\tau) \in TOL_I^{\mathrm{T_{iosa}}}) \Rightarrow (\lambda{\cdot}(o,\tau) \in TOL_{InpComp(S)}^{\mathrm{T_{iosa}}})$.
7. Items 3 and 6 imply that $\forall \lambda \in TOL_{InpComp(S)}^{\mathrm{T_{iosa}}}$: $(\lambda{\cdot}(o,\tau) \in TOL_I^{\mathrm{T_{iosa}}}) \Rightarrow (\lambda{\cdot}(o,\tau) \in TOL_{InpComp(S)}^{\mathrm{T_{iosa}}})$.
8. Item 7 means $(I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ InpComp(S))$. **QED**

A.1.2. Proof of:
$(I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ InpComp(S)) \Rightarrow (I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ S)$**:**

1. We assume $(I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ InpComp(S))$, that is (from Def. 3.1), $\forall \lambda \in TOL_{InpComp(S)}^{\mathrm{T_{iosa}}}$: $(\lambda{\cdot}(o,\tau) \in TOL_I^{\mathrm{T_{iosa}}}) \Rightarrow (\lambda{\cdot}(o,\tau) \in TOL_{InpComp(S)}^{\mathrm{T_{iosa}}})$.
2. $TOL_S^{\mathrm{T_{iosa}}} \subseteq TOL_{InpComp(S)}^{\mathrm{T_{iosa}}}$.
3. Items 1 and 2 imply that $\forall \lambda \in TOL_S^{\mathrm{T_{iosa}}}$: $(\lambda{\cdot}(o,\tau) \in TOL_I^{\mathrm{T_{iosa}}}) \Rightarrow (\lambda{\cdot}(o,\tau) \in TOL_{InpComp(S)}^{\mathrm{T_{iosa}}})$.
4. Item 2 of Sect. A.1.1 and the above Item 3 imply that $\forall \lambda \in TOL_S^{\mathrm{T_{iosa}}}$: $(\lambda{\cdot}(o,\tau) \in TOL_I^{\mathrm{T_{iosa}}}) \Rightarrow (\lambda{\cdot}(o,\tau) \in TOL_S^{\mathrm{T_{iosa}}})$.
5. Item 4 means $(I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ S)$. **QED**

A.2. Proof of Lemma 3.2:
$I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ S \Leftrightarrow TOL_I^{\mathrm{T_{iosa}}} \subseteq TOL_S^{\mathrm{T_{iosa}}}$

A.2.1. Proof of:
$I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ S \Rightarrow TOL_I^{\mathrm{T_{iosa}}} \subseteq TOL_S^{\mathrm{T_{iosa}}}$**:** In the following, τ_λ denotes the time of the last (observable) action of λ.

1. We assume $S = InpComp(S))$.
2. We assume $(I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ S)$, that is (from Def. 3.1), for any output o and $\forall \lambda \in TOL_S^{\mathrm{T_{iosa}}}$: $(\lambda{\cdot}(o,\tau) \in TOL_I^{\mathrm{T_{iosa}}}) \Rightarrow (\lambda{\cdot}(o,\tau) \in TOL_S^{\mathrm{T_{iosa}}})$.
3. $\varepsilon \in TOL_S^{\mathrm{T_{iosa}}}$ and $\varepsilon \in TOL_I^{\mathrm{T_{iosa}}}$, that is, $TOL_S^{\mathrm{T_{iosa}}}$ and $TOL_I^{\mathrm{T_{iosa}}}$ contain the empty sequence.
4. Item 1 implies: $(\lambda \in TOL_S^{\mathrm{T_{iosa}}}) \Rightarrow (\forall \tau > \tau_\lambda, \forall i, \lambda{\cdot}(i,\tau) \in TOL_S^{\mathrm{T_{iosa}}})$.
5. $\lambda{\cdot}(e,\tau) \in TOL_I^{\mathrm{T_{iosa}}} \Rightarrow \tau > \tau_\lambda$.
6. Items 2, 4 and and 5 imply that $\forall \lambda \in TOL_S^{\mathrm{T_{iosa}}}$: $(\lambda{\cdot}(e,\tau) \in TOL_I^{\mathrm{T_{iosa}}}) \Rightarrow (\lambda{\cdot}(e,\tau) \in TOL_S^{\mathrm{T_{iosa}}})$.
7. Items 3 and 6 imply: $(\lambda \in TOL_I^{\mathrm{T_{iosa}}}) \Rightarrow (\lambda \in TOL_S^{\mathrm{T_{iosa}}})$. This implication can be easily proved by induction.
8. Item 7 means: $TOL_I^{\mathrm{T_{iosa}}} \subseteq TOL_S^{\mathrm{T_{iosa}}}$. **QED**

A.2.2. Proof of:
$TOL_I^{\mathrm{T_{iosa}}} \subseteq TOL_S^{\mathrm{T_{iosa}}} \Rightarrow I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ S$**:**

1. We assume $TOL_I^{\mathrm{T_{iosa}}} \subseteq TOL_S^{\mathrm{T_{iosa}}}$.
2. Item 1 implies $(\lambda{\cdot}(o,\tau) \in TOL_I^{\mathrm{T_{iosa}}}) \Rightarrow (\lambda{\cdot}(o,\tau) \in TOL_S^{\mathrm{T_{iosa}}})$.
3. Item 2 implies: $(I\ \mathbf{conf}_{\mathrm{T_{iosa}}}\ S)$. **QED**

B. Proof of Proposition 4.1

We first need to define symbolic languages of $\mathrm{T_{iosa}}$ and $\mathrm{SE_{iosa}}$.

B.1. Symbolic languages of $\mathrm{T_{iosa}}$ and $\mathrm{SE_{iosa}}$

In [24], *SetExp* is used to transform timed automata (TA) into Set-Exp-Automata (SEA), and timed language of a TA A (TL_A^{TA}) and timed language of a SEA B are defined as the set of timed sequences accepted by A and B, respectively. Note that if we ignore the semantics of DG and VA in $\mathrm{T_{iosa}}$ and $\mathrm{SE_{iosa}}$, we obtain the models of TA and SEA, respectively.

By analogy with timed language of TA, we define the symbolic timed language of a $\mathrm{T_{iosa}}$ $A = (\mathcal{L}, l_0, \mathcal{H}, \mathcal{D}, \mathcal{I}, \Sigma, \mathcal{T})$ ($STL_A^{\mathrm{T_{iosa}}}$) as the set of timed sequences accepted by A, where in each transition Tr $=\langle q; r; \sigma; \theta; CG; Z; DG; VA\rangle$ of A: the semantics of DG and VA is ignored, and $(\sigma(\theta); DG; VA)$ is syntactically processed as an action. That is, a sequence $\xi^t = (\alpha_1, \tau_1)(\alpha_2, \tau_2)\cdots(\alpha_i, \tau_i)\cdots \in STL_A^{\mathrm{T_{iosa}}}$ corresponds to a sequence of consecutive transitions $Tr_1\, Tr_2 \cdots Tr_i \cdots$ in A such that:

- Tr_1 is a first transition of A (i.e., executable from l_0);
- α_i consists of $\sigma(\theta)$, DG and VA of Tr_i; and
- after the execution of a prefix $(\alpha_1, \tau_1)\cdots(\alpha_p, \tau_p)$ of ξ^t, the CG of Tr_{p+1} is $True$ at time τ_{p+1}.

In the same way, by analogy with timed language of SEA, the symbolic timed language of a $\mathrm{SE_{iosa}}$ B ($STL_B^{\mathrm{SE_{iosa}}}$) is defined as the set of timed sequences accepted by B, where in each transition Tr $= \langle q; r; \mu; DG; VA\rangle$ of B: the semantics of DG and VA is ignored, and $(\mu; DG; VA)$ is syntactically processed as an action. That is, a sequence $\xi^t = (\alpha_1, \tau_1)(\alpha_2, \tau_2)\cdots(\alpha_i, \tau_i)\cdots \in STL_B^{\mathrm{SE_{iosa}}}$ corresponds to a sequence of consecutive transitions $Tr_1\, Tr_2 \cdots Tr_i \cdots$ in B such that:

- Tr_1 is a first transition of B,
- α_i consists of μ, DG and VA of Tr_i, and
- consistency condition is respected (see Definition in Sect. 4.6).

In [24], it is proved that for a TA A and the corresponding SEA $B = SetExp(A)$, we have: $TL_A^{\mathrm{TA}} = RmvSetExp(TL_B^{\mathrm{SEA}})$. From the above analogy, we deduce that for a $\mathrm{T_{iosa}}$ A and the corresponding $\mathrm{SE_{iosa}}$ $B = SetExp(A)$, we have: $STL_A^{\mathrm{T_{iosa}}} = RmvSetExp(STL_B^{\mathrm{SE_{iosa}}})$.

B.2. Transforming a symbolic timed language into a timed language

To a symbolic timed language (STL) corresponds a timed language (TL) defined as follows:
$\lambda^t = (e_1, \tau_1)(e_2, \tau_2) \cdots \in TL$ *iff*
$\exists \xi^t = (\alpha_1, \tau_1)(\alpha_2, \tau_2) \cdots \in STL$ s.t.
- λ^t and ξ^t have the same length n (n can be infinite)
- $\alpha_i = (e_i, DG_i, VA_i)$ or $\alpha_i = e_i$ (in the latter case, $DG_i = True$ and VA_i is empty),
- $\forall i \leq n$: DG_i evaluates to $True$ after the application of $VA_1, VA_2, \cdots VA_{i-1}$.

Let then $STL2TL$ be the operator that transforms STL into TL, (written $TL = STL2TL(STL)$). Therefore, for a $\mathrm{T_{iosa}}$ A we have $TL_A^{\mathrm{T_{iosa}}} = STL2TL(STL_A^{\mathrm{T_{iosa}}})$, and for a $\mathrm{SE_{iosa}}$ B we have $TL_B^{\mathrm{SE_{iosa}}} = STL2TL(STL_B^{\mathrm{SE_{iosa}}})$.

B.3. Proof that:

$TL_A^{\mathrm{T_{iosa}}} = RmvSetExp(TL_{SetExp(A)}^{\mathrm{SE_{iosa}}})$

1. In Sect. B.1, we have seen that for every $\mathrm{T_{iosa}}$ A: $STL_A^{\mathrm{T_{iosa}}} = RmvSetExp(STL_{SetExp(A)}^{\mathrm{SE_{iosa}}})$.
2. In Sect. B.2, we have seen that for every $\mathrm{T_{iosa}}$ A: $TL_A^{\mathrm{T_{iosa}}} = STL2TL(STL_A^{\mathrm{T_{iosa}}})$.
3. In Sect. B.2, we have seen that for every $\mathrm{SE_{iosa}}$ B: $TL_B^{\mathrm{SE_{iosa}}} = STL2TL(STL_B^{\mathrm{SE_{iosa}}})$.
4. The order in which operators $RmvSetExp$ and $STL2TL$ are applied has no influence on the result.
5. Items 1 and 2 imply: $TL_A^{\mathrm{T_{iosa}}} = STL2TL(RmvSetExp(STL_{SetExp(A)}^{\mathrm{SE_{iosa}}}))$.
6. Items 4 and 5 imply: $TL_A^{\mathrm{T_{iosa}}} = RmvSetExp(STL2TL(STL_{SetExp(A)}^{\mathrm{SE_{iosa}}}))$.
7. Items 3 and 6 imply: $TL_A^{\mathrm{T_{iosa}}} = RmvSetExp(TL_{SetExp(A)}^{\mathrm{SE_{iosa}}})$. **QED**

B.4. Proof that:

$TOL_A^{\mathrm{T_{iosa}}} = RmvSetExp(TOL_{SetExp(A)}^{\mathrm{SE_{iosa}}})$

Let A be a $\mathrm{T_{iosa}}$ and $B = SetExp(A)$ be the corresponding $\mathrm{SE_{iosa}}$. The *timed observable language* of A ($TOL_A^{\mathrm{T_{iosa}}}$) is obtained from $TL_A^{\mathrm{T_{iosa}}}$ by removing all the internal actions. And the *timed observable language* of B ($TOL_B^{\mathrm{SE_{iosa}}}$) is obtained in the same way from $TL_B^{\mathrm{SE_{iosa}}}$. And let $RmvIntern(x)$ be the operation that removes all internal actions from a timed language. Let us prove that: $TOL_A^{\mathrm{T_{iosa}}} = RmvSetExp(TOL_{SetExp(A)}^{\mathrm{SE_{iosa}}})$.

1. $TL_A^{\mathrm{T_{iosa}}} = RmvSetExp(TL_{SetExp(A)}^{\mathrm{SE_{iosa}}})$.
2. $TOL_A^{\mathrm{T_{iosa}}} = RmvIntern(TL_A^{\mathrm{T_{iosa}}})$
3. $TOL_{SetExp(A)}^{\mathrm{SE_{iosa}}} = RmvIntern(TL_{SetExp(A)}^{\mathrm{SE_{iosa}}})$.
4. Items 1 and 2 imply: $TOL_A^{\mathrm{T_{iosa}}} = RmvIntern(RmvSetExp(TL_{SetExp(A)}^{\mathrm{SE_{iosa}}}))$.
5. The order in which Set and Exp actions and internal actions are removed from a timed sequence, has no influence on the result.
6. Items 4 and 5 imply: $TOL_A^{\mathrm{T_{iosa}}} = RmvSetExp(RmvIntern(TL_{SetExp(A)}^{\mathrm{SE_{iosa}}}))$.
7. Items 3 and 6 imply: $TOL_A^{\mathrm{T_{iosa}}} = RmvSetExp(TOL_{SetExp(A)}^{\mathrm{SE_{iosa}}})$. **QED**

Proposition 4.1 is obtained by replacing $TOL_B^{\mathrm{SE_{iosa}}}$ by $AddTime(OL_B^{\mathrm{SE_{iosa}}})$ in the above Item 7. **QED**

C. Proof of Proposition 5.1

Let S be an input-complete $\mathrm{T_{iosa}}$, and X, Y be defined as follows:
X: $TOL_{\mathcal{IUT}}^{\mathrm{T_{iosa}}} \subseteq TOL_S^{\mathrm{T_{iosa}}}$
Y: $\exists$ $\mathrm{SE_{iosa}}$ $\mathcal{SUT}$ accepting the behavior of **SUT** s.t. $OL_{\mathcal{SUT}}^{\mathrm{SE_{iosa}}} \subseteq OL_{SetExp(S)}^{\mathrm{SE_{iosa}}}$
From Hyp. 3.1, Lemma 3.2 and Def. 4.3, we deduce that the objective is to prove:
$(Tester \preceq SetExp(S)) \Rightarrow (X \Leftrightarrow Y)$.

C.1. Proof of: $X \Rightarrow Y$

Assuming X and $Tester \preceq SetExp(S)$, the aim is to prove Y. Recall that $\mathcal{E}$ denotes a set of Exp actions.

Definition C.1 *The supremal* $\mathrm{SE_{iosa}}$ *of a* $\mathrm{SE_{iosa}}$ B *is denoted* $\mathrm{SupSE_{iosa}}(B)$ *and constructed as follows:*

- *Construct* $Obs(B)$*, the projection of* B *into the observable alphabet, i.e., internal actions are made invisible.*
- *For every internal action* ϵ_x*: add a selfloop labeled* ϵ_x *to every location of* $Obs(B)$.

- *For every internal action ϵ_x and every transition of type 1 (i.e, labeled in the form $\mathcal{E}$) from a location q to a location r: add another transition of type 3 labeled $(\mathcal{E}, \epsilon_x)$ from q to r.*

Note that by construction, $OL_B^{\mathrm{SE_{iosa}}} = OL_{\mathrm{SupSE_{iosa}}(B)}^{\mathrm{SE_{iosa}}}$, $TOL_B^{\mathrm{SE_{iosa}}} = TOL_{\mathrm{SupSE_{iosa}}(B)}^{\mathrm{SE_{iosa}}}$, $TL_B^{\mathrm{SE_{iosa}}} \subseteq TL_{\mathrm{SupSE_{iosa}}(B)}^{\mathrm{SE_{iosa}}}$, and $TL_{\mathrm{SupSE_{iosa}}(B)}^{\mathrm{SE_{iosa}}} = \bigcup_{TOL_{X_i}^{\mathrm{SE_{iosa}}} \subseteq TOL_B^{\mathrm{SE_{iosa}}}} TL_{X_i}^{\mathrm{SE_{iosa}}}$.

1. Let $\mathcal{SUT} = \mathrm{SupSE_{iosa}}(SetExp(S))$, and thus, $OL_{SetExp(S)}^{\mathrm{SE_{iosa}}} = OL_{\mathcal{SUT}}^{\mathrm{SE_{iosa}}}$, $TOL_{SetExp(S)}^{\mathrm{SE_{iosa}}} = TOL_{\mathcal{SUT}}^{\mathrm{SE_{iosa}}}$, $TL_{SetExp(S)}^{\mathrm{SE_{iosa}}} \subseteq TL_{\mathcal{SUT}}^{\mathrm{SE_{iosa}}}$.
2. In Sect. B.4 we have shown that: $TOL_S^{\mathrm{T_{iosa}}} = RmvSetExp(TOL_{SetExp(S)}^{\mathrm{SE_{iosa}}})$.
3. X and Item 2 imply: $TOL_{\mathcal{IUT}}^{\mathrm{T_{iosa}}} \subseteq RmvSetExp(TOL_{SetExp(S)}^{\mathrm{SE_{iosa}}})$.
4. Item 3 and $\mathcal{SUT} = \mathrm{SupSE_{iosa}}(SetExp(S))$ of Item 1 imply: $TL_{\mathcal{IUT}}^{\mathrm{T_{iosa}}} \subseteq RmvSetExp(TL_{\mathcal{SUT}}^{\mathrm{SE_{iosa}}})$.
5. Item 4 and $Tester \preceq SetExp(S)$ imply that $\mathcal{SUT}$ accepts the behavior of **SUT**.

From X and ($Tester \preceq SetExp(S)$), we have determined a $\mathrm{SE_{iosa}}$ $\mathcal{SUT}$ that accepts the behavior of **SUT** and s.t. $OL_{SetExp(S)}^{\mathrm{SE_{iosa}}} = OL_{\mathcal{SUT}}^{\mathrm{SE_{iosa}}}$. Therefore, we have Y.

C.2. Proof of: $Y \Rightarrow X$

Let $RmvTime(x)$ be the operation defined as follows: if $\lambda = (e_1, \tau_1)(e_2, \tau_2)\cdots(e_i, \tau_i)\cdots$, then $RmvTime(\lambda) = e_1 e_2 \cdots e_i \cdots$.

1. We consider $\lambda \in TOL_{\mathcal{IUT}}^{\mathrm{T_{iosa}}}$.
2. In Sect. B.4 we have shown that: $TOL_S^{\mathrm{T_{iosa}}} = RmvSetExp(TOL_{SetExp(S)}^{\mathrm{SE_{iosa}}})$.
3. The existence of $\mathcal{SUT}$ that accepts the behavior of **SUT** implies: $TL_{\mathcal{IUT}}^{\mathrm{T_{iosa}}} \subseteq RmvSetExp(TL_{\mathcal{SUT}}^{\mathrm{SE_{iosa}}})$, and thus, $TOL_{\mathcal{IUT}}^{\mathrm{T_{iosa}}} \subseteq RmvSetExp(TOL_{\mathcal{SUT}}^{\mathrm{SE_{iosa}}})$.
4. Items 1 and 3 imply: $\exists \lambda' \in TOL_{\mathcal{SUT}}^{\mathrm{SE_{iosa}}}$ such that $\lambda = RmvSetExp(\lambda')$.
5. $\lambda' \in TOL_{\mathcal{SUT}}^{\mathrm{SE_{iosa}}}$ in Item 4 implies: $\lambda" = RmvTime(\lambda') \in OL_{\mathcal{SUT}}^{\mathrm{SE_{iosa}}}$.
6. Y and Item 5 imply: $\lambda" \in OL_{SetExp(S)}^{\mathrm{SE_{iosa}}}$.
7. Item 6 and the fact that $\lambda" = RmvTime(\lambda')$ imply: $\lambda' \in TOL_{SetExp(S)}^{\mathrm{SE_{iosa}}}$.
8. Items 2 and 7 imply: $\lambda = RmvSetExp(\lambda') \in TOL_S^{\mathrm{T_{iosa}}}$.

From Y and Item 1, we have deduced Item 8. Therefore, we have X. **QED**

D. Proofs of Lemmas 6.1, 6.2 and 6.3

Let $SpecTP_A$ denote the part of $SpecTP$ (obtained in Step 1) that leads to a location A, and $SpecTP_A^{\mathrm{SE_{iosa}}}$ denote the part of $SpecTP^{\mathrm{SE_{iosa}}}$ (obtained in Step 2) that leads to a state A.

D.1. Proof of Lemma 6.1

1. When **SUT** executes a trace λ that leads to a state $p \in \mathbf{Pass}$, then λ conforms (w.r.t. $\mathbf{conf}_{\mathrm{SE_{iosa}}}$) to $SpecTP_A^{\mathrm{SE_{iosa}}}$.
2. Prop. 5.1 and item 1 imply that when **SUT** executes a trace λ that leads to $p \in \mathbf{Pass}$, then the **IUT** has executed a timed trace μ that conforms (w.r.t. $\mathbf{conf}_{\mathrm{T_{iosa}}}$) to $SpecTP_A$.
3. Item 2 and $TOL_{SpecTP_A}^{\mathrm{T_{iosa}}} \subseteq TOL_{SpecTP}^{\mathrm{T_{iosa}}}$, imply that when **SUT** executes a trace λ that leads to $p \in \mathbf{Pass}$, then the **IUT** has executed a timed trace μ that conforms (w.r.t. $\mathbf{conf}_{\mathrm{T_{iosa}}}$) to $SpecTP$.
4. Item 3 and $TOL_{SpecTP}^{\mathrm{T_{iosa}}} = TOL_{Spec}^{\mathrm{T_{iosa}}}$, imply that when **SUT** executes a trace λ that leads to $p \in \mathbf{Pass}$, then the **IUT** has executed a timed trace μ that conforms (w.r.t. $\mathbf{conf}_{\mathrm{T_{iosa}}}$) to $Spec$.
5. Item 2 implies that when **SUT** executes a trace λ that leads to $p \in \mathbf{Pass}$, then the **IUT** has executed a timed trace μ that leads to location A of TP.
6. Items 4 and 5 imply Lemma 6.1. **QED**

D.2. Proof of Lemma 6.2

1. When **SUT** executes a trace λ that leads to the state **fail**, then λ does not conform (w.r.t. $\mathbf{conf}_{\mathrm{SE_{iosa}}}$) to $SpecTP^{\mathrm{SE_{iosa}}}$.
2. Prop. 5.1 and item 1 imply that when **SUT** executes a trace λ that leads to $fail$, then the **IUT** has executed a timed trace μ that does not conform (w.r.t. $\mathbf{conf}_{\mathrm{T_{iosa}}}$) to $SpecTP$.
3. Item 2 and $TOL_{SpecTP}^{\mathrm{T_{iosa}}} = TOL_{Spec}^{\mathrm{T_{iosa}}}$, imply that when **SUT** executes a trace λ that leads to $fail$, then the **IUT** has executed a timed trace μ that does not conform (w.r.t. $\mathbf{conf}_{\mathrm{T_{iosa}}}$) to $Spec$. **QED**

D.3. PROOF OF LEMMA 6.3

1. When **SUT** executes a trace λ that leads to a state $x \in \mathbf{Inconc}$, then λ conforms to $SpecTP^{\mathrm{SE}_{\mathrm{iosa}}}$ but no state A can be reached after λ.

2. Prop. 5.1 and item 1 imply that when **SUT** executes a trace λ that leads to $x \in \mathbf{Inconc}$, then the **IUT** has executed a timed trace μ that conforms (w.r.t. $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$) to $SpecTP$ but no location A can be reached after μ.

3. Item 2 and $TOL^{\mathrm{T}_{\mathrm{iosa}}}_{SpecTP} = TOL^{\mathrm{T}_{\mathrm{iosa}}}_{Spec}$, imply that when **SUT** executes a trace λ that leads to $x \in \mathbf{Inconc}$, then the **IUT** has executed a timed trace μ that conforms (w.r.t. $\mathbf{conf}_{\mathrm{T}_{\mathrm{iosa}}}$) to $Spec$ but no location A can be reached after μ. **QED**