Electronic Notes in Theoretical Computer Science 130 (2005) 79–100 

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Complete Test Graph Synthesis For Symbolic Real-time Systems

Ahmed Khoumsi [1](#_bookmark1) *,*[2](#_bookmark1)

*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.

# 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 *gen- erating* ) 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](mailto:Ahmed.Khoumsi@USherbrooke.ca)

1571-0661 © 2005 Elsevier B.V. Open access under [CC BY-NC-ND license](http://creativecommons.org/licenses/by-nc-nd/3.0/).

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](#_bookmark41),[24](#_bookmark58),[4](#_bookmark38),[21](#_bookmark55),[9](#_bookmark43),[20](#_bookmark54),[5](#_bookmark39),[6](#_bookmark40),[16](#_bookmark49),[19](#_bookmark50)].

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](#_bookmark57),[22](#_bookmark56),[8](#_bookmark42)]. 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](#_bookmark36)] and diagnosis [[26](#_bookmark60)].

We propose a test synthesis method which combines, and thus extends, real- time testing and symbolic testing. We are motivated by the desire to syn- thesize 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* (Tiosa), that adds time to the IOSTS model of [[23](#_bookmark57)] 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 Tiosa into an automaton called Set-Exp-IOSA (SEiosa). *SetExp* denotes such a transformation, and *SetExp*(*A*) denotes the SEiosa obtained by transformation of a Tiosa *A*. *SetExp* basically adds to the struc- ture of a Tiosa 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](#_bookmark57)].

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](#_bookmark0) describes the Tiosa model used to describe the specification of the IUT. In Sect. [3](#_bookmark5), we define formally the test problem to be solved. Sect. [4](#_bookmark13) introduces the SEiosa model and the transformation “*SetExp* : Tiosa '→SEiosa”. In Sect. [5](#_bookmark18), we propose a test architecture and present a theorem related to *SetExp*. Sect. [6](#_bookmark22) presents a method based on *SetExp* that solves the test problem. And in Sect. [7](#_bookmark33), we conclude the paper.

# Timed IOSA (Tiosa)

In this section, we present timed input output symbolic automata (Tiosa) used to describe the IUT and its specification. Tiosa is a combination of *timed automata* of [[16](#_bookmark49)] and *input output symbolic transition systems* (IOSTS) of [[23](#_bookmark57)].

* 1. *Clocks and related concepts*

A clock *ci* is a real variable whose value can be reset (to 0) with the occur- rence of an action and such that, between two resets, its derivative (w.r.t. time) is equal to 1. Let H = {*c*1*,* ··· *, cN*c } be a set of clocks.

A Clock Guard (CG) is a conjunction of formula(s) in the form “*ci* ∼ *k*”,

where *ci* ∈ H, ∼∈ {*<, >,* ≤*,* ≥*,* =}, and *k* is a nonnegative integer. A CG can be the constant *True* (empty conjunction). Let ΦH be the set of CGs using clocks of H.

A clock reset is a subset of H, and 2H denotes the set of clock resets.

* 1. *Data and related concepts*

A variable is a data whose value can be set with the occurrence of an action.

Let V be a set of variables.

A constant is a data whose value is set once at initial time. Let C be a set of constants.

A (communication) parameter is a data which is transmitted as a param- eter of an action. Let P be a set of parameters.

A Data Guard (DG) is a boolean expression using data of D = V ∪ C ∪ P. Let ΓD denote the set of data guards (we consider that *True* ∈ ΓD).

A Variable Assignment (VA) is a set of assignments *v* := *E*, where *v* ∈V

and *E* is an expression depending on D. Let ΛD be the set of VAs.

The domain of definition of every *x* ∈D is noted *Type*(*x* ).

* 1. *Syntax of* Tiosa

A Tiosa is defined by (L*, l*0*,* H*,* D*,* I*,* Σ*,* T ), where: L is a finite set of locations, *l*0 is the initial location, H is a finite set of clocks, D = V∪ C ∪ P is a finite set of data, I is a boolean expression depending of V∪ C called initial condition, Σ is a finite set of actions, and T is a transition relation. To each *a* ∈ Σ is associated a tuple (*p*1*,* ··· *, pk*) of parameters (possibly empty) denoted *θa* . Signature of *a* is denoted *Sig* (*a*) and defined as follows:

⎧⎨

⟨*Type*(*p1* ) ··· *Type*(*pk* )⟩ if *a* is an input or output

*Sig* (*a*)=

⎩ empty tuple if *a* is an internal action

There are three kinds of actions: the reception of an *input i* containing the tuple *θi* , written ?*i*(*θi* ); the sending of an *output o* containing the tuple *θo*, written !*o*(*θo*); and the occurrence of an *internal action a*, written *ϵa*. *θi* and

*θo* 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 Tiosa is defined by Tr = ⟨*q*; *r*; *σ*; *θσ*; *CG*; *Zσ*; *DG*; *VA*⟩, where: *q* and *r* are origin and destination locations; *σ* is an action in the form

?*i*, !*o* or *ϵa*; *θσ* is the (possibly empty) tuple of parameters associated to *σ*; *CG* and *Zσ* are a clock guard and a clock reset; and *DG* and *VA* are a data guard and a variable assignment defined in V∪ C ∪ *θσ*. [3](#_bookmark3)

The index *σ* in *Zσ* means that the clock reset of a transition depends only on its action. This restriction guarantees determinizability of Tiosa [[16](#_bookmark49)].

Fig. [1](#_bookmark2) illustrates the definition of Tiosa through an example. Locations are represented by nodes, and a transition Tr = ⟨*q*; *r*; *σ*; *θσ*; *CG*; *Zσ*; *DG*; *VA*⟩ is represented by an arrow linking *q* to *r* and labeled in 3 lines by: *σ*(*θσ*), (*CG*; *Zσ*) and (*DG*; *VA*). The *CG* and *DG True* and the absence of *Zσ* or *VA* are indicated by “-”. Σ = {*φ, α, β, ρ*}, H = {*c*1}, V = {*x*}, C = {*p*}, 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.



*l*3

 (m)

*l*0

( − ; − )

( )

− ; x := m

*l*1



( − ; {c1}) ( − ; − )

*l*2



( c1 > 3 ; − )

( − ; − )

*l*4

Fig. 1. Example of Tiosa

* 1. *Semantics of* Tiosa

At time *τ*0 = 0, the Tiosa *A* = (L*, l*0*,* H*,* D*,* I*,* Σ*,* T ) is at location *l*0 with all clocks equal to 0, and variables and constants taking values such that I evaluates to *True*. A transition Tr =⟨*q*; *r*; *σ*; *θσ*; *CG*; *Zσ*; *DG*; *VA*⟩ 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 *θσ*) can be executed only when Tr is enabled [4](#_bookmark4) ; and after the execution of *σ*: location *r* is reached, the clocks in *Zσ* (if any) are reset, and the assignments in *VA* (if any) are applied.

3 Note that *DG* and *VA* of a transition Tr = ⟨*q*; *r*; *σ*; *θσ*; *CG* ; *Zσ*; *DG*; *VA*⟩ are defined in

V∪ C ∪ *θσ* and *not* in the whole D = V∪ C ∪ P

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

For the example of Fig. [1](#_bookmark2), let *δu,v* be the delay between actions *u* and *v*:

* + - The Tiosa is initially in location *l*0. At the occurrence of *φ*(*m*), location *l*1

is reached and variable *x* is assigned with the value of *m*.

* + - From *l*1, the Tiosa reaches *l*2 at the occurrence of *α*.
    - From *l*2, the Tiosa reaches *l*3 or *l*4 at the occurrence of *β*. *l*3 is reached only if *δα,β <* 3 and *x* ≥ *p*, and *l*4 is reached only if *δα,β >* 2 and *x* ≤ *p*.

We see that there is a nondeterminism when 2 *< δα,β <* 3 and *x* = *p*. *x* is incremented when *l*4 is reached.

* + - From *l*3, the Tiosa executes nothing.
    - From *l*4, the Tiosa reaches *l*1 at the occurrence of *ρ*. We have *δα,ρ >* 3.

The semantics of a Tiosa *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*, τ*1) ··· (*ei, τi*) ·· ·”, where 0 *< τ*1 *<* ··· *< τi <* ·· ·.

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

Acceptance of a timed sequence *λt* = (*e*1*, τ*1)(*e*2*, τ*2) ·· ·, for *e*1*, e*2*,* ··· ∈ Σ. Let *n* be the length of *λt* (*n* can be infinite), and *λt i* = (*e*1*, τ*1) ··· (*ei, τi*) be the prefix of *λt* of length *i*, for 0 ≤ *i* ≤ *n* (*i* is finite). *λt is accepted by A iff λt* is the empty sequence *λt* 0 or *A* has a sequence of length *n* of consecutive transitions *Tr1 Tr2* ··· starting at *l*0 and such that ∀*i* = 1*,* 2*,* ··· *, n*: the action of *Tri* is *ei* and, after the execution of *λti*−1, *Tri* is enabled at time *τi*. Intuitively, *λt* corresponds to an execution of *A*.

Acceptance of a timed trace : Let *µt* = (*e*1*, τ*1)(*e*2*, τ*2) ··· be a timed trace. *µt* is accepted by *A iff µt* is obtained by removing all the timed internal actions of a timed sequence accepted by *A*. Intuitively, *µt* corresponds to the observation of an execution of *A*.

Definition 2.1 The *Timed observable language of a* Tiosa *A* (*TOL*Tiosa ) is the

*A*

set of timed traces accepted by *A*. That is, *TOL*Tiosa

*A*

models the observable

behavior of *A*.

The class of Tiosa that we will consider satisfies the following hypothesis:

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

Remark 2.2 Unlike [[1](#_bookmark35)], 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.

# 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 Tiosa and the notion of test purpose. A test hypothesis is also necessary.

* 1. *Conformance relation between* Tiosa*, and some related lemmas*

In the following: *I* and *S* denote two Tiosas over the same alphabet Σ, and *o*

is an output action of Σ. We define the following conformance relation:

Definition 3.1 *I* confT

iosa

*S* means, ∀*λ* ∈ *TOL*Tiosa :

(*λ*·(*o, τ* ) ∈ *TOL*Tiosa ) ⇒ (*λ*·(*o, τ* ) ∈ *TOL*Tiosa ).

*S*

*I S*

If the IUT is modeled by *I* , “*I* confTiosa *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 confTiosa , let us first define the *input- completion* of Tiosa. Let Σ? be the set of inputs of the alphabet Σ, and *Univ* be a “universal” Tiosa accepting *all* the timed traces over Σ. That is, *TOL*Tiosa contains every timed trace over Σ. The following definition is inspired from [[11](#_bookmark45),[12](#_bookmark46)].

*Univ*

Definition 3.2 The *input-completion* of a Tiosa *A* = (L*, l*0*,* H*,* D*,* I*,* Σ*,* T ) is a Tiosa *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 Tiosa such that:

Tiosa

*TOL*

*InpComp*(*A*)

= *TOL*Tiosa ∪(

*w*·*a*·*x*).

*w*∈*TOL*Tiosa *,a*∈Σ *,w*·*a*/∈*TOL*Tiosa *,x*∈*TOL*Tiosa

*A*

*A* ? *A*

*Univ*

*A* is said *input-complete iff A* = *InpComp*(*A*). Intuitively, an input-complete

Tiosa accepts every input at any time.

Lemma 3.3 (*I* confTiosa *S* ) ⇔ (*I* confTiosa *InpComp*(*S* ))*.*

Lemma [3.3](#_bookmark7) implies that we can replace a Tiosa *S* by its input-completion before checking whether a Tiosa *I* conforms to it, w.r.t. confTiosa . However, Def. [3.2](#_bookmark6) is not constructive and we do not know how to compute *InpComp*(*S* )) from a Tiosa *S* in the general case. Hence, we will use the following hypothesis:

Hypothesis 3.1 *In “I* confTiosa *S”, we assume S input-complete.*

Note that Lemma [3.3](#_bookmark7) and Hyp. [3.1](#_bookmark8) are inspired from their non-real-time and non-symbolic (i.e., without clocks and data) version in [[11](#_bookmark45)].

Lemma 3.4 *With Hypothesis* [*3.1*](#_bookmark8)*: I* confT

*S* ⇔ *TOL*Tiosa ⊆ *TOL*Tiosa *.*

iosa *I S*

Lemma [3.4](#_bookmark9) means that with Hypothesis [3.1](#_bookmark8), confTiosa is simplified into an inclusion of timed observable languages of Tiosa.

* 1. *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 Tiosa *A* = (L*, l*0*,* H*,* D*,* I*,* Σ*,* T ) is said to be *complete iff* :

∀*l* ∈ L and ∀*e* ∈ Σ, *e* is enabled in *l* for every possible clock value and data value. Intuitively, a complete Tiosa 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 *σ* ∈ Σ, there is a selfloop transition Tr = ⟨*q*; *q*; *σ*; *θσ*; *True*; ∅; *True*; ∅⟩. 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 Tiosa *TP* used to select the behaviors to be tested. By analogy with [[10](#_bookmark44),[23](#_bookmark57),[16](#_bookmark49)], *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](#_bookmark59)]:

Hypothesis 3.2 IUT *can be described by a (possibly unknown) input-complete*

Tiosa IUT *.*

* 1. *Formalization of the test problem*

Given two Tiosas *Spec* and *TP* over the same alphabet, describing the spec- ification 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: IUT confTiosa *Spec*.

We assume *Spec* input-complete (see Hyp. [3.1](#_bookmark8)). *CTG* is an interesting au- tomaton 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., *λ* ∈ *TOL*Tiosa ∩ *TOL*Tiosa

) and such that: a

IUT *SpecTP*

location *R* of *TP* may be reached by *λ*, or no location *A* of *TP* is reachable

after *λ* by *Spec*.

# Transformation of Tiosa into SEiosa

A transformation, called *SetExp*, is presented in [[18](#_bookmark51)] and applied in [[13](#_bookmark47),[16](#_bookmark49),[17](#_bookmark52),[14](#_bookmark48),[15](#_bookmark53)]. *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 Tiosa instead of TA. When apply- ing *SetExp* to Tiosa, 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 ac- count when using (interpreting, processing, ...) the automaton called SEiosa that results from *SetExp*. Our test problem will be solved in Sect. [6](#_bookmark22) by using *SetExp*.

In this Section, we present the SEiosa model and illustrate *SetExp* by an ex- ample (a detailed description of *SetExp* can be found in [[18](#_bookmark51)]). Let *A* be a Tiosa over an alphabet Σ and *SetExp*(*A*) be the SEiosa obtained by transformation.

* 1. *Actions Set and Exp*

*Set*(*ci , k* ) means: clock *ci* is reset (to 0) and will expire when its value is equal to *k*. More generally, *Set*(*ci , k1 , k2 ,* ··· *, kp*) means that *ci* is reset and will expire several times, when its value is equal to *k*1*, k*2*,* ··· *, kp*, resp. We assume without loss of generality that *k*1 *< k*2 *<* ··· *< kp*.

*Exp*(*ci , k* ) means: clock *ci* expires and its current value is *k*.

Therefore, *Set*(*ci , k* ) is followed (after a delay *k*) by *Exp*(*ci , k* ), and *Set*(*ci , k1 , k2 ,* ··· *, kp*) is followed (after delays *k*1*,* ··· *, kp*) by *Exp*(*ci , k1* ), *Exp*(*ci , k2* )*,* ··· *, Exp*(*ci , kp*). When a *Set*(*ci , m*) occurs, then all *Exp*(*ci ,* ∗) which were expected before this *Set*(*ci , m*) are canceled.

* 1. *Basic principle of SetExp*

In a Tiosa *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 Tiosa *A* is:

1. To replace each clock reset in *A* by the appropriate *Set* action; and then
2. 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.

* 1. *Transitions of* SEiosa

Recall that a transition of Tiosa is defined by ⟨*q*; *r*; *σ*; *θσ*; *CG*; *Zσ*; *DG*; *VA*⟩ and represented in a figure by an arrow linking *q* to *r* and labeled by: *σ*(*θσ*), (*CG*; *Zσ*) and (*DG*; *VA*). Let: *η* denote an action of the alphabet Σ of the Tiosa *A* with its parameters, S (resp. E) denote a set of *Set* (resp. *Exp*) actions, and *occurrence of* S (resp. E) mean the simultaneous occurrences of all the actions in S (resp. E). We categorize transitions of SEiosa *SetExp*(*A*) into three types:

Type 1 : transition labeled by (E) represents the occurrence of E.

Type 2 : transition labeled by (*η*) or (*η,* S), and by a *DG* and a *VA*. (*η*) represents the occurrence of *η*, (*η,* S) represents the simultaneous occur- rences of *η* and S, and *DG* and *VA* have the same semantics as in Tiosa. 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 S (if any) of TR.

Type 3 : transition labeled by (E *, η*) or (E *, η,* S), and by a *DG* and a *VA*. (E *, η*) represents the simultaneous occurrences of E and *η*, and (E *, η,* S) represents the simultaneous occurrences of E, *η* and S. A transition TR of Type 3 in *SetExp*(*A*) corresponds to simultaneous executions of 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 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.

* 1. *Example of transformation SetExp* : Tiosa '→SEiosa

For the Tiosa *A* of Fig. [1](#_bookmark2), we obtain the SEiosa *SetExp*(*A*) of Fig. [2](#_bookmark16), where *Set2,3* is an abbreviation of ?*Set*(*c1 , 2 , 3* ), *Expi* is an abbreviation of !*Exp*(*c1 , 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 *Expi* . 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 *φ*(*m*), (*α, Set2,3* ), *β* or *ρ* in Line 1. Transitions of Type 3 are those labeled (*Expi , β*) in Line 1, and correspond to the simultaneous executions of *Expi* and *β*. We do not indicate whether each action *φ*(*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](#_bookmark8) (input-completeness), be- cause 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*(*c1 , 2* ), *c*1 takes

all the nonnull real values smaller than 2.



( >

; − )

|  |  |
| --- | --- |
|  |  |
|  | ( > ; − ) |

Fig. 2. SEiosa *SetExp*(*A*) obtained from the Tiosa *A* of Fig. [1](#_bookmark2)



Exp2

Exp3

Exp3

Exp2



( − ; − )

Exp3

(m)

( − ; x := m)

  Set2,3

( − ; − )



; x++ )

(



; x++ )

(

* 1. *Syntax of* SEiosa

Let *A* = (L*, l*0*,* H*,* D*,* I*,* Σ*,* T ) be a Tiosa and *B* = *SetExp*(*A*) be the corre- sponding SEiosa. The *syntax* of *B* can be defined by *B* = (Q*, q*0*,* D*,* I*,* Λ*,* Ψ), where: 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 D and I are the same as those used in the definition of *A*. A transition of *B* is syntactically defined by TR = ⟨*q*; *r*; *µ*; *DG*; *VA*⟩, 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](#_bookmark14) and [4.4](#_bookmark15)). Λ is an alphabet consisting of labels of transitions of types 1, 2 and 3 (see Sect. [4.3](#_bookmark14)).

* 1. *Semantics of* SEiosa

Let us define the *semantics* of a SEiosa *B* = (Q*, q*0*,* D*,* I*,* Λ*,* Ψ). Initially, *B* is at state *q*0 with all clocks of H equal to 0, and variables and constants taking values such that I evaluates to *true*. The transition TR =⟨*q*; *r*; *µ*; *DG*; *VA*⟩ 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* SEiosa denote a sequence “*E*1*E*2 ·· ·”, where *E*1*, E*2*,* ··· *,* ∈ Λ; and let a *trace of* SEiosa be obtained from a sequence of SEiosa by re- moving all its internal actions. Let us define the semantics of a SEiosa *B* = (Q*, q*0*,* D*,* I*,* Λ*,* Ψ) by the set of sequences and traces accepted by *B*:

Acceptance of a (finite or infinite) sequence *λ* = *E*1*E*2 ·· ·, for

*E*1*, E*2*,* ··· ∈ Λ. Let *n* be the length of *λ* (*n* can be infinite), and *λi* = *E*1*E*2 ··· *Ei* be the prefix of *λ* of length *i*, for 0 ≤ *i* ≤ *n* (*i* is finite). *λ is accepted by B iff λ* is the empty sequence *λ*0 or:

there exists a sequence of transitions *Tr1 Tr2* ··· of *B* of length *n* such that

∀*i* = 1*,* 2*,* ··· *, n*: *Tri* is labeled by *Ei* and, after the execution of *λi*−1, *Tri* 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 SEiosa:

Definition 4.3 The observable language of a SEiosa *B* (*OL*SEiosa ) is the set of traces accepted by *B*. That is, *OL*SEiosa models the observable behavior of *B*.

*B*

*B*

Given a SEiosa *B*, one can remark that *OL*SEiosa

*B*

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*SEiosa )) = *TOL*Tiosa *.*

*SetExp*(*A*)

*A*

Intuitively, Theorem [4.4](#_bookmark17) 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.

# Test architecture, and a theorem

Given two Tiosas *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 Tiosas *Spec* and *TP* , but rather on a SEiosa computed from the two Tiosas. In order to make the link between *CTG* and the IUT, we use a particular test architecture [[16](#_bookmark49)] that we now present. It is represented in Fig. [3](#_bookmark20):

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 confSEiosa which is simply an inclusion of observable languages of SEiosa:

Definition 5.1 Let *I* ' and *S* ' be two SEiosas over the same alphabet:

(*I* ' conf *S* ') ≡ (*OL*SEiosa ⊆ *OL*SEiosa ).

SEiosa

*I* ' *S* '

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* Tiosa*. If Test-Controller generates Set actions only when they are accepted by SetExp*(*S* )*, then:*

(IUT confTiosa *S* ) ⇔

(∃ SEiosa SUT *accepting behavior of* SUT *such that* SUT confSEiosa *SetExp*(*S* ))*.*

The above theorem implies that we can check “SUT confSEiosa *SetExp*(*S* )” instead of “IUT confTiosa *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](#_bookmark57)].



Test−Controller

Set(c,k)

Exp(c,k)

Clock−Handler

input

output

SUT

IUT

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](#_bookmark21) means that there is no timing constraint relatively to internal actions.

# Method of test generation

Our test method combines, and thus extends, two complementary test meth- ods presented in [[23](#_bookmark57)] and [[16](#_bookmark49)], respectively. It consists of four steps outlined in Fig. [4](#_bookmark23) and described in subsections [6.1](#_bookmark25) to [6.4](#_bookmark30). Its inputs are *Spec* (input- complete, from Lemma [3.3](#_bookmark7) and Hyp. [3.1](#_bookmark8)) and *TP* (complete, from Def. [3.7](#_bookmark11)). In a first step, we compute a Tiosa *SpecTP* which accepts (all and only) the timed sequences of *Spec* and indicates the locations corresponding to the loca- tions *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 exe- cuted on the IUT in order to determine whether: IUT confTiosa *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](#_bookmark10) and

[3.3](#_bookmark12) 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](#_bookmark57)]. [5](#_bookmark24)

*Spec*

*TP*

Step 1

Step 2

Synchronized *SpecTP* Transformation *SpecTPSEA*

Step 3

Extraction of visible behaviour

*SpecTPSEA*

Step 4

*VIS*

Computation of

product

SetExp

complete test graph

*CTG*

Fig. 4. Steps of the test method

Figure [5](#_bookmark26) represents *Spec* and *TP* over the alphabet Σ = {?*φ,* ?*σ,* !*ρ, ϵa, ϵb*} used to illustrate the steps of the test method. Data of *Spec* are H1 = {*c*1}, V1 = {*x*}, C1 = {*p*}, P1 = {*m*}, where *x, p, m* are integers. Data of *TP* are H2 = V2 = C2 = ∅, P2 = {*n*}, where *n* is integer. /= *x* means any action of Σ different from *x*, and ?∗ means any input ∈ Σ (i.e., ?*φ* or ?*σ*). *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](#_bookmark8)). 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 !*ρ* 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



*l*3

(m)

*l*0

( − ; − )

( )

− ; x := m

*l*1



( − ; {c1}) ( − ; − )

 



*l*

2

(n)









A





(m)

*l*4

R 



TP

Spec

*TL*





( c1 > 3 ; − )

( − ; − )

a

b

* 1. *Step 1 : Synchronous product of Spec and TP*

The aim is to compute a Tiosa *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 Tiosas.

Let *Ai* = (L*i, li ,* H*i,* D*i,* I*i,* Σ*i,* T *i*) where D*i* = V*i* ∪ C*i* ∪ P*i*, for *i* = 1*,* 2, be two Tiosas. The synchronized product of *A*1 and *A*2, written *A1* ⊗ *A2* , is inspired (but different) from the synchronized product of TA [[2](#_bookmark37)] and the synchronized product of IOSTS [[23](#_bookmark57)]. *A1* ⊗ *A2* is defined *iff* the following four conditions are satisfied:

0

1. Σ1 = Σ2. The common alphabet will then be denoted Σ. This condition can be easily relaxed [[23](#_bookmark57)], but we will keep it for simplicity.
2. H1 ∩ H2 = ∅ [[2](#_bookmark37)].
3. (V1 ∪P1) ∩(V2 ∪P2)= ∅, (C1 ∪P1) ∩P2 = ∅, and (C2 ∪P2) ∩P1 = ∅ [[23](#_bookmark57)].
4. Each action *a* ∈ Σ has the same signature in *A*1 and *A*2 [[23](#_bookmark57)].

Assuming the above four conditions satisfied, *A1* ⊗ *A2* is defined by (L*, l*0*,* H*,* D*,* I*,* Σ*,* T ) such that: L = L1 × L2, *l*0 = (*l*1*, l*2), H = H1 ∪ H2,

0 0

D = V∪ C ∪ P, V = V1 ∪ V2, C = (C1 ∪ C2) \ V, P = P1 ∪ P2, I = (I1 ∧ I2),

and the set of transitions T is defined as follows:

For each pair of transitions (⟨*qi*; *ri*; *σ*; *θ i*; *CGi*; *Z i*; *DGi*; *VAi*⟩∈ T *i*, *i* = 1*,* 2:

*σ σ*

If *θ* 1 and *θ* 2 are the empty tuple *ϵ* : then there is a transition

*σ σ*

⟨(*q 1* ; *q 2* ); (*r 1* ; *r 2* ); *σ*; *ϵ*; *CG*1 ∧ *CG*2; *Z* 1 ∪ *Z* 2; *DG*1 ∧ *DG*2; *VA*1 ∪ *VA*2⟩∈ T .

*σ σ*

If *θ* 1 and *θ* 2 are not empty : let *DG*1*,*2 (resp. *VA*1*,*2) denote the expres-

*σ σ*

sion obtained by replacing in *DG*2 (resp. *VA*2) each parameter from *θ* 2

*σ*

by the corresponding, same-position parameter from *θ* 1; then there is a transition ⟨(*q 1* ; *q 2* ); (*r 1* ; *r 2* ); *σ*; *θ* 1; *CG*1 ∧ *CG*2; *Z* 1 ∪ *Z* 2; *DG*1 ∧ *DG*1*,*2; *VA*1 ∪

*σ*

*σ σ σ*

*VA*1*,*2⟩∈ T . This procedure is inspired from [[23](#_bookmark57)].

In Step 1, we compute *SpecTP* = *Spec* ⊗ *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

= *TOL*

equivalent (i.e., *TOL*Tiosa

*Spec*

Tiosa

*SpecTP*

). Completeness of *TP* and input-

completeness of *Spec* imply that *SpecTP* is input-complete. The effect of

*Spec* ⊗ *TP* is to determine in *Spec* all the executions that correspond to loca- tions *A* and *R*, respectively.

For *Spec* and *TP* of Fig. [5](#_bookmark26), we obtain the *SpecTP* of Fig. [6](#_bookmark27). 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 corre- sponds 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*.



*A*

*L* 



( − ; {c1})

( − ; − )





*A*

*A*

(m)







( c1 > 3 ; − )

( − ; − )







( c1 > 3 ; − )

( − ; − )

(m)

*A*



*R A*

 

Ignored part

(m) ( − ; − )

( − ; x := m)



( − ; {c1}) ( − ; − )

Fig. 6. Step 1: *SpecTP* obtained from *Spec* and *TP* of Fig. [5](#_bookmark26)

* 1. *Step 2 : Transforming the* Tiosa *SpecTP into* SEiosa

We transform the problem into a non-real-time form by computing *SpecTP* SEiosa = *SetExp*(*SpecTP* ). For the *SpecTP* of Fig. [6](#_bookmark27), we obtain the *SpecTP* SEiosa of Fig. [7](#_bookmark29). *Set2,3* denotes ?*Set*(*c1 , 2 , 3* ), and *Expi* denotes

!*Exp*(*c1 , i* ). States of *SpecTP* SEiosa that correspond to location *A* (resp. *R*) of *SpecTP* are indicated by *A* (resp. *R*), (Expi*,* Σ) means the simultaneous occurrences of Expi and any *x* ∈ Σ, and nodes linked by a broken line indi- cate the same state [6](#_bookmark28) . State *A*1 of Fig. [7](#_bookmark29) 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](#_bookmark29) and subsequent figures, if *DG* evaluates to *true* and *VA* is empty in a transition, then (*DG*; *VA*) is not represented.





R

Exp2 

Exp2



R

Exp3  

Exp3



R

?\*

?\*

?\*

Exp3

Exp2

Exp3

Exp3



A1 .....

S1

  Set2,3

Exp2

Exp3

?\*



(m)

?\*

?\*

?\*

?\*

R



R



R Exp2 

 Exp

R



Exp3  

Exp

R



R



Exp3  

Exp

3

2

3

R



(m)

( − ; x := m)

b

; x++ )

(

b

Fig. 7. Step 2: *SpecTP* SEiosa obtained from *SpecTP* of Fig. [6](#_bookmark27)

a

>

a

>

* 1. *Step 3 : Extracting the visible behavior of SpecTP* SEiosa

We construct the visible behavior of *SpecTP* SEiosa in three substeps:

Substep 3a : Internal actions are eliminated by projection into the observ- able alphabet. For that purpose, we can *adapt* a procedure proposed in [[23](#_bookmark57)]. The result is denoted *Vis*(*SpecTP* SEiosa ). The adaptation consists of a pre- liminary step where internal actions in transitions of Type 3 are simply erased. After that, we can use the procedure of [[23](#_bookmark57)] 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* SEiosa ) is determinized by using a heuristic proposed in [[23](#_bookmark57)]. The result is denoted *Det*(*Vis*(*SpecTP* SEiosa )).

Substep 3c : Note that every state of *Det*(*Vis*(*SpecTP* SEiosa )) corresponds to one or several states of *SpecTP* SEiosa . States *R* and *A* of

*Det*(*Vis*(*SpecTP* SEiosa )) are selected as follows:

* + - We call *R* every state corresponding to at least one state *R* of *SpecTP* SEiosa . 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* SEiosa . Intuitively, we accept every execution which:

1) cannot correspond to a sequence not to be tested, and 2) can corre- spond 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* SEiosa . In- tuitively, we accept an execution only when we are sure that it corresponds to a sequence to be tested.

The result is denoted *SpecTP* SEiosa .

*VIS*

For the *SpecTP* SEiosa of Fig. [7](#_bookmark29), we obtain *SpecTP* SEiosa of Fig. [8](#_bookmark31).



 Set2,3

Exp2

Exp3



x p

A



 (m)

?\*

?\*

?\*

R

R



Exp3 ,o Exp3

o



o

R

o

Exp 2, o Exp2

R

o

R



o

(m)

( − ; x := m)

*VIS*

Fig. 8. Step 3: *SpecTP* SEiosa obtained from *SpecTP* SEiosa of Fig. [7](#_bookmark29)

*VIS*

* 1. *Step 4 : Computing a Complete Test Graph (CTG)*

Recall that a transition of SEiosa is labeled in the form (S) or in one of the following four forms: (*σ*), (*σ,* £), (S *, σ*), (S *, σ,* £), in addition to (*DG*; *VA*). Let “output transition of SEiosa” denote any transition labeled in one of the four forms such that *σ* is an output of the IUT. By analogy with [[10](#_bookmark44),[16](#_bookmark49),[23](#_bookmark57)], we construct a Complete Test Graph (*CTG*) as follows:

* + - Let *L2A* be the set of states of *SpecTP* SEiosa from which a *A* is accessible.

*VIS*

* + - Let Pass denote the set of states *A* of *SpecTP* SEiosa .

*VIS*

* + - Let Fail = {fail} consist of a new state that is reached by every non- specified output transition of *SpecTP* SEiosa executable from *L2A*.

*VIS*

* + - Let Inconc be the set of states of *SpecTP* SEiosa that are not in *L2A* ∪ Pass

*VIS*

and are accessible from *L2A* by a single output transition of *SpecTP* SEiosa .

*VIS*

* + - We then obtain *CTG* from *SpecTP* SEiosa by:

*VIS*

* + - * adding (implicitly) state Fail and its incoming transitions,
      * removing every state /∈ *L2A* ∪ Pass ∪ Inconc ∪ Fail, and
      * removing outgoing transitions of every state ∈ Pass ∪ 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 !*δ* whose occurrence means: no observable action occurs during a period equal to *T* . We then proceed as follows:

* + - we define a new state inconc*δ* ∈ Inconc, and
    - to every state /∈ Pass∪Inconc∪Fail without outgoing Exp-Trans (of type 1 or 3), we add a transition labeled !*δ* and leading to inconc*δ*.

The use of !*δ* and inconc*δ* 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* SEiosa

*VIS*

of Fig. [8](#_bookmark31), we obtain the *CTG* of Fig. [9](#_bookmark32).

In *CTG*, every input (resp. output) action must be interpreted as an out-

put (resp. input) action of the tester (i.e., Test-Controller). Transition !*δ* in State 0 (resp. 1) is irrelevant, because it can be preempted by the only possible other transition labeled (?*φ*(*m*); −; *x* := *m*) (resp. (?*σ, Set2,3* )) which is under the control of Test-Controller. Transition !*δ* 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 !*δ* can be eas- ily implemented by using ?*Set*(*c0 , T* ) and !*Exp*(*c0 , 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* SEiosa of Fig. [8](#_bookmark31))



(m)

0 ( − ; x := m)

1



 Set2,3

2

Exp2

3

Exp3

4



x p

pass



inconc



*VIS*

Let *SpecTP A* denote the part of *SpecTP* (obtained in Step 1) that leads to

a location *A*, and *SpecTP* SEiosa denote the part of *SpecTP* SEiosa (obtained in

*A*

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. confSE

iosa

) to *SpecTP* SEiosa . From

Theor. [5.2](#_bookmark19), the IUT has executed a timed trace *µ* that conforms (w.r.t.

*A*

confTiosa ) to *SpecTP A*, and thus, to *SpecTP* . Therefore, *µ* conforms (w.r.t.

confTiosa ) to *Spec* (because *Spec* and *SpecTP* are observationally equivalent).

*Fail* means that *λ* does not conform (w.r.t. confSE ) to *SpecTP* SEiosa . From Theor. [5.2](#_bookmark19), the IUT has executed a timed trace *µ* that does not conform (w.r.t. confTiosa ) to *SpecTP* , and thus, nor to *Spec* (because *Spec* and *SpecTP* are observationally equivalent).

iosa

*Inconclusive* means that we cannot determine whether *λ* conforms (w.r.t.

*A*

confSE

iosa

) to *SpecTP* SEiosa . From Theor. [5.2](#_bookmark19), we cannot determine whether

the IUT has executed a timed trace *µ* that conforms (w.r.t. confTiosa ) to

*SpecTP A*.

# Contribution and future Work

Real-time test consists of testing systems which must guarantee timing con- straints. 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](#_bookmark57)] and the method of real-time testing of [[16](#_bookmark49)]. Since the test method in [[16](#_bookmark49)] is a rigorous generalization of TGV [[10](#_bookmark44)] to the real-time case, we can say that our method is a rigorous generalization of STG and TGV [7](#_bookmark34) 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 dur- ing 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](#_bookmark26)).

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](#_bookmark57)]) 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](#_bookmark21)). 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 Tiosa, in order to model actions that *must* occur (instead of being only *permitted* to occur) when they are enabled.
    - Def. [3.2](#_bookmark6) is not constructive and we do not know how to compute *InpComp*(*S* )) from a Tiosa *A* in the general case. We intend to determine a class of (nondeterministic) Tiosas 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 J´eron 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.

1. Alur, R. and D. Dill, *A theory of timed automata*, Theoretical Computer Science 126 (1994),

pp. 183–235.

1. 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.
2. Braberman, V., M. Felder and M. Mass´e, *Testing timing behaviors of real time software*, in:

*Proc. Quality Week 1997*, San Francisco, USA, 1997, pp. 143–155.

1. Cardell-Oliver, R., *Conformance testing of real-time systems with timed automata*, Formal Aspects of Computing 12 (2000), pp. 350–371.
2. Cardell-Oliver, R., *Conformance testing of real-time systems with timed automata*, in: *Nordic* *Workshop on Programming Theory*, 2000.
3. Clarke, D., “Testing real-time constraints,” Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania, USA (1996).
4. Clarke, D., T. J´eron, 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.
5. 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.
6. Jard, C. and T. J´eron, *TGV : theory, principles and algorithms*, in: *Proc. 6th World Conf. on Integ. Design and Process Technol. (IDPT)*, Pasadena, California, USA, 2002.
7. Jard, C., T. J´eron, L. Tanguy and C. Viho, *Remote testing can be as powerful as local testing*, in: *Proc. PSTV/FORTE*, Beijing, China, 1999.
8. J´eron, 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.
9. 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.](http://www.gel.usherb.ca/khoumsi/Research/Public/FTRTFT02.ps)

1. 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.](http://www.gel.usherb.ca/khoumsi/Research/Public/WODES02.ps)

1. 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.](http://www.gel.usherb.ca/khoumsi/Research/Public/WODES04.ps)
2. Khoumsi, A., T. J´eron 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.](http://www.gel.usherb.ca/khoumsi/Research/Public/FATES03.ps)
3. 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.](http://www.gel.usherb.ca/khoumsi/Research/Public/RTCSA02-Cont.ps)
4. 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.](http://www.gel.usherb.ca/khoumsi/Research/Public/SBMF04-SetExp.pdf)
5. 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).
6. 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).
7. 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.
8. Rusu, V., *Veriﬁcation using test generation techniques*, in: *Formal Methods Europe (FME)*

(2002), pp. 252–271.

1. Rusu, V., L. du Bousquet and T. J´eron, *An approach to symbolic test generation*, in: *Int. Conf. on Integrating Formal Methods (IFM)* (2000), pp. 338–357.
2. Springintveld, J., F. Vaandrager and P. D’Argenio, *Testing timed automata*, Technical Report CTIT97-17, University of Twente, Amsterdam, The Netherlands (1997).
3. Tretmans, J., “A Formal Approach to Conformance Testing,” Ph.D. thesis, University of Twente, The Netherlands (1992).
4. Tripakis, S., *Fault diagnosis for timed automata*, in: *Proc. Form. Technique in Real-Time and Fault-Toler. Syst. (FTRTFT), LNCS 2469* (2002).