Conformance Testing of Variable Driven Automata

Omer Nguena Timo
LABRI - CNRS (UMR 5800)
University of Bordeaux
F-33405 Talence, France
nguena@labri.fr

Antoine Rollet
LABRI - CNRS (UMR 5800)
University of Bordeaux
F-33405 Talence, France
rollet@labri.fr

Abstract

In this paper, we address the conformance testing problem for timed constrained critical systems. We propose a new model adapted to describe such systems. The model is called Variable Driven Timed Automata (VDTA) and is a variant of timed automata in which events are variable assignments and all transitions are urgent. We present a sound and exhaustive on the fly testing algorithm for such systems. As an application of our approach, we propose a case study on a "Bi-manual command" system.

1. Introduction

Testing is an important activity during the development cycle. Since systems are getting more and more complex, it is important to use a proper validation strategy to ensure their correctness. The testing process consists of applying experimentation directly on an Implementation Under Test (IUT). Testing may focus on different topics such as conformance, reliability, interoperability or robustness. The test sequences may be obtained with two major methods : (1) structural testing : the source code is used to extract data for test experimentation (flow graph, etc...); (2) functional testing : the specification (formal or not) is used as a basis for test cases generation. In both cases, the results are observed and compared to the specification.

In this paper we deal with functional testing, and more precisely with formal conformance testing: we define a formal model for specifying a class of timed constrained critical systems. We provide a testing algorithm to ensure the correctness of such systems. The model is called Variable Driven Timed Automata (VDTA). VDTA is a variant of Timed Automata ([1]), specially adapted to describe concisely systems guided by their variable values: in states of variable driven systems, the time elapses continuously or the environment changes the contents of *input variables*. Transitions are fired *as soon as* constraints over input, *output* and *clock* variables are satisfied and contents of output variables may change. We consider that a firing event is a changing value of a variable. In this model all transitions are urgent i.e. the transition is automatically fired if the guard is true. There are no other events in such automata. It permits to handle simultaneous variations of variables. In addition of this contribution, we present a testing approach adapted for this model with a proof of completeness and a case study on a real system: the "Bi-manual command" [2].

The paper is organised as follows. Section 2 recalls the previous major contributions of this domain. Section 3 describes the new model used to specify timed constrained critical systems. Section 4 presents the complete testing framework and a proof of soundness and exhaustiveness. We give a case study of our approach on a real system called "Bi-manual machine" in Section 5 and we finally conclude in Section 6.

This work has been supported by the ANR[1] *Testec* Project.

2. State of the art

Validation of timed systems has been studied quite a long time. Alur and Dill proposed a new model for such systems called timed automata [1] : automata with guards and clocks. Then testing approaches have been proposed in [3], [4], [5] and [6]. [3] and [6] present testing approaches for extensions of timed automata with inputs and outputs based on characterization of states approaches inspired from the Finite State Machines (FSM) testing theory. [4] proposes to generate test cases from event-clock automata, a subset of timed automata with one clock for one action. In [6] and [5] authors use the region graph ([1]) in order to generate test suites.

In the same period, Tretmans end al. presented a new approach for formal testing ([7]) based on asynchronous and non-deterministic specifications with the Labelled Transition Systems model and its I/O extension, and defined conformance relations as preorders. In [8], authors propose to apply the same principle with timed conformance relations on the Timed Extended Finite State Machines (TEFSM) model, a timed extension of FSM with guards on variables and clocks. Labelled Transition Systems (LTS) model has inspired many timed extensions using the LTS semantic ([9], [10]). In [10] authors describe a testing method for timed automata with inputs, outputs

[1] Agence Nationale de la Recherche

and variables. They propose a timed extension of the **ioco** relation ([7]) and a test generation method using special *tick* actions to model time elapsing.

In the previous described works, only events coming from the environment permit to fire transitions. Such models are not adapted for variable driven systems.

The Uppaal project uses timed automata with variables and the possibility of handling urgent events. The associated testing tool [9] generates test cases based on an on the fly algorithm using a symbolic extension of [7]. It is possible to find a Uppaal model of the "Bi-manual" command in [2]. The problem of all these models is that their semantic does not allow to consider simultaneous actions directly in the model. Then it is necessary to add a special event for this case, maybe leading to an important number of transitions.

3. Models of Systems

3.1. Definitions

A simple way to represent timed systems is to use timed transition systems (TTS) over a set of actions. Transitions of TTS are labelled with either an action from Σ or a delay from $\mathbb{R}^+$, the set of non negative real numbers.

Definition 1 A *timed transition system (TTS)* over the alphabet Σ is a transition system $S = \langle Q, \Sigma, q_0, \rightarrow \rangle$, where Q is the set of states, $q_0 \in Q$ is the initial state, and the transition relation $\rightarrow \subseteq Q \times (\Sigma \cup \mathbb{R}^+) \times Q$ consists of time-elapsing transitions $q \xrightarrow{t} q'$ (with $t \in \mathbb{R}^+$), and discrete transitions $q \xrightarrow{a} q'$ (with $a \in \Sigma$).

As usual, we require the following standard properties for TTS:

- TIME-DETERMINISM (if $q \xrightarrow{t} q'$ and $q \xrightarrow{t} q''$ with $t \in \mathbb{R}^+$, then $q' = q''$),
- 0-DELAY :($q \xrightarrow{0} q$),
- ADDITIVITY (if $q \xrightarrow{t} q'$ and $q' \xrightarrow{t'} q''$ with $t,\ t' \in \mathbb{R}^+$, then $q \xrightarrow{t+t'} q''$) and,
- CONTINUITY (if $q \xrightarrow{t} q'$, then for every t' and t'' in $\mathbb{R}^+$ such that $t = t' + t''$, there exists q'' such that $q \xrightarrow{t'} q'' \xrightarrow{t''} q'$).

Notation: Let $a \in \Sigma$ be an action. we write $q \overset{a}{\not\rightarrow}$ when from q there is no transition labelled with a.

In software engineering, TTS are not adapted to specify directly timed systems; this is because the set of (time-elpasing) transitions can be infinite. But TTS is a natural representation for the semantics of high-level timed models such as timed automata [1], the Uppaal model [9] or the model that we introduce.

We define Variable Driven Timed Automata (VDTA), a new model for specifying an important class of timed systems. VDTA is a variant of timed automata. Compared to timed automata, actions in our models, are assignments of memory-variables and constraints are defined over clocks and memory-variables. Another particularity of our model is that all transitions are urgent, meaning that they must be fired as soon as constraints are satisfied. Justification of this choice is given in Subsection 3.2.

A variable v ranges over a domain $Dom(v)$. Given a set of variables V, we define $Dom(V) = \cup_{v \in V} \mathcal{D}om(v)$, the domain of V. An *action* a over a set of variables V is a total function $a : V \rightarrow \mathcal{D}om(V)$ that assigns a value in $\mathcal{D}om(v)$ to each variable $v \in V$. The symbol Σ_V will denote the set of actions over a set of variables in 2^V.

A *constraint* g over V is a boolean combination of simple constraints of the form $v \bowtie c$ where $\bowtie \in \{<, \leq, >\geq\}$ and c is a constant. The set of constraints over V is denoted by $\Phi(V)$.

In our model, the set of variables are partitioned into a set $\mathcal{H}$ of *clock-variables* and a set $\mathcal{X}$ of *memory-variables*. A *clock-variable* (or a clock for short) h records the elapse of time. A clock is evaluated over $\mathbb{R}^+$, the set of non negative real numbers. A *memory-variable* (or a memory for short) x records a value from a domain. If an action changes a clock, it sets the clock always to zero.

A *clock-constraint* is a constraint defined over a set of clocks and a *memory-constraint* is defined over a set of memory-variables. Constants in constraints are natural numbers.

Definition 2 A *variable driven timed automaton (VDTA)* is a tuple $\mathcal{A} = \langle L, l^0, \Sigma_{\mathcal{X} \cup \mathcal{H}}, \mathcal{X}, \mathcal{X}_c, \mathcal{H}, \Delta \rangle$ where, L is a finite set of locations, l^0 is the initial location, $\mathcal{X}$ is a finite set of memory-variables, $\mathcal{X}_c \subseteq \mathcal{X}$ is the set of *controllable memory-variables*, $\mathcal{H}$ is a finite set of clocks, and $\Delta \subseteq L \times \Phi(\mathcal{X} \cup \mathcal{H}) \times \Sigma_{(\mathcal{X} \setminus \mathcal{X}_c) \cup \mathcal{H}} \times L$ is the transition relation.

Notation: A transition $(\ell, g, a, \ell') \in \Delta$ is often denoted by $\ell \xrightarrow{g,a} \ell'$.

The environment of a system modelled with a VDTA observes all memory-variables. The set of *controllable memory-variables* $\mathcal{X}_c$ represents the variables in which the environment (e.g. the tester) can assign a value. Actions from $\Sigma_{\mathcal{X}}$ that involve variables from $\mathcal{X} \setminus \mathcal{X}_c$ are termed *uncontrollable*; otherwise they are termed *controllable*.

We describe the semantics of VDTA presented above. We need to evaluate variables, to check whether a constraint is true according to a given valuation, and we will need to get new values of variables after an action is performed.

A *snapshot* over a set of variables V is a function $val : V \rightarrow \mathcal{D}om(V)$ defined such that $val(v) \in \mathcal{D}om(v)$. The symbol $\mathcal{V}al(V)$ denotes the set of snapshots over V. Actions can change values of variables. Given an action a and a snapshot val of variables of V, the snapshot $val[a]$ returns values of memories after the execution of the action a; it is defined by $val[a](v) = a(v)$ if the action a is over v; otherwise $val[a](v) = val(v)$.

As the time elapses in real-time systems, we consider a *time elapse* operation over variables. This operation only changes the value of the clock-variables and leaves the memory-variables unchanged. Given a delay $t \in \mathbb{R}^+$, and a snapshot val over $\mathcal{X} \cup \mathcal{H}$, the snapshot $val + t$ is such that $(val + t)(v) = val(v) + t$ if $v \in \mathcal{H}$; otherwise $(val + t)(v) = val(v)$.
A constraint g can be satisfied by a snapshot val; this notion is standard. We write $val \models g$ when g is evaluated to true according to val.

Definition 3 The semantics $[\![\mathcal{A}]\!]$ of a VDTA $\mathcal{A}$ is the TTS $[\![\mathcal{A}]\!] = \langle L \times \mathcal{V}al(\mathcal{H} \cup \mathcal{X}), (l_0, val^0), \Sigma_{\mathcal{H}\cup\mathcal{X}} \cup \mathbb{R}^+, \rightarrow \rangle$, where val^0 is an initial snapshot of variables and the transition relation $\rightarrow$ is such that:

T1: $(l, val) \xrightarrow{a} (l', val[a])$ if and only if there exists $(l', g, a) \in L \times \Phi(\mathcal{X} \cup \mathcal{H}) \times \Sigma_{(\mathcal{X}\setminus\mathcal{X}_c)\cup\mathcal{H}}$ such that $l \xrightarrow{g,a} l'$ and $val \models g$.

T2: $(l, val) \xrightarrow{a'} (l, val[a'])$ with $a' \in \Sigma_{\mathcal{X}_c}$ if for every $(l', g, a) \in L \times \Phi(\mathcal{X} \cup \mathcal{H}) \times \Sigma_{(\mathcal{X}\setminus\mathcal{X}_c)\cup\mathcal{H}}$ such that $l \xrightarrow{g,a} l'$ we have $val \not\models g$.

T3: $(l, val) \xrightarrow{t} (l, val + t)$ with $t > 0$ if for every $t' < t$, for every $(l', g, a) \in L \times \Phi(\mathcal{X} \cup \mathcal{H}) \times \Sigma_{(\mathcal{X}\setminus\mathcal{X}_c)\cup\mathcal{H}}$, such that $l \xrightarrow{g,a} l'$ we have $val + t' \not\models g$.

The semantics considers two kinds of transitions: *discrete transitions (T1 and T2)* and *delay transitions (T3)*. Delay transitions (T3) represent the elapse of time. Discrete transitions are triggered by actions on variables. There are two sorts of discrete transitions: *internal discrete transitions (T1)* and *external discrete transitions (T2)*. Internal discrete transitions (T1) are fired as soon as constraints are satisfied by the current snapshot. External discrete transitions (T2) allow to change only the contents of controllable memory-variables; they are fired when an action on controllable memory-variable is executed and no constraint is satisfiable. Internal discrete transitions are urgent. External discrete transitions and delay transitions are fired only if there is no urgent transition that can be fired. In the following $(l, val) \xrightarrow{a}_1 (l', val')$ denotes a discrete transition of type T1, $(l, val) \xrightarrow{a}_2 (l', val')$ denotes a discrete transition of type T2 and $(l, val) \xrightarrow{t}_3 (l', val')$ denotes a delay transition. Later we will consider contents of memory-variables in a given state (l, val). Out$((l, val))$ denotes the projection of val on memory-variables in $\mathcal{X}$.

3.2. Relation with Other Models

We consider the example of the "Bi-manual command" system. Using VDTA we present a "natural" representation of that system. Then we discuss the difficulty to represent that system with famous Uppaal model. Here is the description of the "Bi-manual command" system.

Consider the control program of a device designed to start some machine when two buttons (L and R for left and right buttons) are pushed within 0.5 seconds. If only one button is pushed (then L or R are true) and a delay of 0.5 time units delay is performed (time-out has occurred), then the whole process must be started again. The machine starts when the signal s is set to 1. After the machine has started (s=1), it stops as soon as one button is released, and it can start again only after both buttons have been released (L and R are both false).

Figure 1 presents a VDTA model for the "Bi-manual command" system. The model has three memory-variables (L, R and s) and a clock h; L and R are controllable variables. The environment can change the content of L and R; but it can not change the content of s. Actions on s are done by the system.

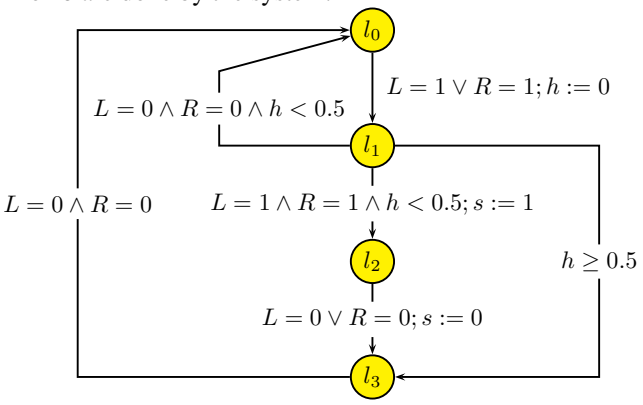

Figure 1. The System Controller

Let us present two important use cases of the model:

1. From the initial location l_0, if L and R are both set to 1, the system must go immediately to l_2 after starting the machine ($s := 1$). This is done by taking in urgency and successively two transitions.
2. If the system reaches l_1 with $L = 0$ and $R = 1$, it can stay in l_1 for at most 0.5 time units, otherwise, the system moves immediately to the state l_3. This is possible since transitions are urgent.

These two use cases are difficult to model with Uppaal because urgent transitions in this tool can not be guarded with constraints on clocks variables and invariants in locations should be constraints of the form $h < n$ or $h \leq n$ (h is a clock and n is a constant).

4. Conformance Testing

4.1. Overview

We consider conformance testing of critical timed systems modelled with VDTA. We define a conformance relation to ensure that an implementation under test (IUT) conforms to its specification. The main idea of this relation is that all behaviors of the implementation have to be allowed by the specification. Especially:

1. The IUT is not allowed to change a memory-variable in a time (too late or too early) when it is not allowed by the specification.

2. The IUT is not allowed to omit to change a memory-variable in a time instant at which the modification is required by the specification.

The model-driven conformance testing activity can be seen as a two player game interaction between a tester (or the environment of the implementation) and the implementation. During a play in the game, the tester provides a conformance verdict of the implementation against the specification. The tester who knows the specification plays in the following way:

1. either, it performs an action on memory-variables and then observes how the implementation reacts. In case of non conformance, the tester returns a *fail* verdict.

2. or, it lets the time elapse for a while; but while the time elapses, it observes eventual actions from the implementation. In case of non conformance, the tester returns a *fail* verdict.

3. or, it stops the play and returns *pass* verdict meaning that up to this point no fault has been detected.

The tester observes behaviours of the implementation through the content of memory-variables. If their content changes, the tester checks whether the new values are expected by the specification. Choice of actions by the tester are guided by the specification.

We assume that the tester can observe all memory-variables of the implementation. The tester is allowed to execute actions that involve only controllable memory-variables (or their copies) of the implementation; it is not allowed to execute actions on non controllable memory-variables. In our model, an output for the tester is an action over controllable memory-variables (or their copies) and an input is an action over observable memory-variables of the implementation.

4.2. Conformance Relation

We provide some definitions involved in the test algorithm. We define the set of reachable states after the execution of a sequence of actions, the computation of outputs and inputs of the specification and the conformance relation **tvco**. These definitions hold on TTS since they are used to define the semantics of systems. They care about urgency of transitions and are not standard.

Let $[\![\mathcal{A}]\!] = \langle Q, q_0, \Sigma_{\mathcal{H}\cup\mathcal{X}} \cup \mathbb{R}^+, \rightarrow\rangle$ be a TTS that represents the semantics of a VDTA $\mathcal{A} = \langle L, l^0, \Sigma_{\mathcal{X}\cup\mathcal{H}}, \mathcal{X}, \mathcal{X}_c, \mathcal{H}, \Delta\rangle$.

A *run* of $\mathcal{A}$ from q_0 is a sequence of the form $q_0 \xrightarrow{t_1}_3 q'_1 \xrightarrow{a_1}_j q_1 \xrightarrow{t_2}_3 q'_1 \xrightarrow{a_2}_j q_2 \cdots \xrightarrow{t_n}_3 q'_n \xrightarrow{a_n}_j q_n$ where $a_i \in \Sigma_{\mathcal{X}\cup\mathcal{H}}, j \in \{1,2\}, t_i \in \mathbb{R}^+$ and it holds that:

- if $q_{i-1} \xrightarrow{t_i}_3 q'_i \xrightarrow{a_i}_j q_i$ and $j = 2$, then there is no $q''_i \in Q$ and no $a \in \Sigma_{\mathcal{X}\cup\mathcal{H}}$ such that $q'_i \xrightarrow{a}_1 q''_i$.

This property of runs is a consequence of the urgency of internal transitions (T1). *Urgent runs* only consider internal transitions. An urgent run is maximal if it ends in a state from which only delay or external transitions can be fired. A *trace* is a sequence of elements from $(\mathbb{R}^+ \times \Sigma_{\mathcal{X}\cup\mathcal{H}})$ and it is an element of $(\mathbb{R}^+ \times \Sigma_{\mathcal{X}\cup\mathcal{H}})^*$. The trace of a run is the ordered sequence of delays and actions on successive transitions of the run.
Let α be an action from $\Sigma_{\mathcal{H}\cup\mathcal{X}}$ or a delay from $\mathbb{R}^+$. We write $q \xrightarrow{\alpha}_i$ iff $q \xrightarrow{\alpha}_i q'$ for some $q' \in Q$ and $i = 1, 2, 3$.
The idea in the definitions of $\Longrightarrow$ below is that if constraints of successive transitions are true, these transitions are fired instantaneously. Let $a \in \Sigma_{\mathcal{H}\cup\mathcal{X}}$ be an action; we write $q \stackrel{a}{\Longrightarrow} q'$ iff there is $q_1, q_2 \in Q, \sigma_1, \sigma_2 \in (\Sigma_{\mathcal{H}\cup\mathcal{X}})^*$ such that $q \xrightarrow{\sigma_1}_1 q_1 \xrightarrow{a}_2 q_2 \xrightarrow{\sigma_2}_1 q'$ and for every $b \in \Sigma_{\mathcal{H}\cup\mathcal{X}}$ with $b \neq a$ it holds that $q_1 \stackrel{b}{\not\rightarrow}_1$ and $q' \stackrel{b}{\not\rightarrow}_1$ (in other words σ_1 and σ_2 are sequences of actions on maximal sequences of urgent transitions).
Recall that successive internal discrete transitions can be fired in urgency and in zero time units and wihtout any variation of controllable variables. Constraints of these transitions should be true. A sequence of internal discrete transitions is *stable* if it is maximal and ends in a state from which no internal discrete transition is firable. After an execution of a stable sequence of internal discrete transitions, the system can let the time elapse or the environment can change the contents of controllable variables. After letting the time elapse, another stable sequence of internal discrete transitions can be executed. In consequence more than one stable sequences of internal discrete transitions can be executed within t time units. For a delay $t \in \mathbb{R}^+$, we write $q \stackrel{t}{\Longrightarrow} q'$ iff $q \xrightarrow{\sigma_1}_1 \xrightarrow{t_1}_3 \xrightarrow{\sigma_2}_1 \xrightarrow{t_2}_3 \xrightarrow{\sigma_3}_1 \ \ldots \ \xrightarrow{\sigma_n}_1 \xrightarrow{t_n}_3 q'$ where each $t = \sum_{i=1..n} t_i$, $\sigma_i \in (\Sigma_{(\mathcal{X}\setminus\mathcal{X}_c)\cup\mathcal{H}})^*$ is a sequence of stable internal discrete transitions and $q' \stackrel{\sigma'}{\not\rightarrow}_1$ for some $\sigma' \in (\Sigma_{(\mathcal{X}\setminus\mathcal{X}_c)\cup\mathcal{H}})^*$).

For a sequence $\sigma = t_0.a_0.t_1.a_1.\ldots.t_n.a_n$ (where $t_i \in \mathbb{R}^+$ and $a_i \in \Sigma_{\mathcal{X}_c\cup\mathcal{H}}$, we write $q \stackrel{\sigma}{\Longrightarrow} q'$ iff $q \stackrel{t_1}{\Longrightarrow}\stackrel{a_1}{\Longrightarrow}\stackrel{t_2}{\Longrightarrow}\stackrel{a_2}{\Longrightarrow} \cdots \stackrel{t_n}{\Longrightarrow}\stackrel{a_n}{\Longrightarrow} q'$. For $\alpha \in \Sigma_{\mathcal{X}\cup\mathcal{H}} \cup \mathbb{R}^+$ we define q *After* $\alpha = \{q' \mid q \stackrel{\alpha}{\Longrightarrow} q'\}$. We define $\text{Trace}(q) = \{\sigma \in (\Sigma_{\mathcal{X}\cup\mathcal{H}} \cup \mathbb{R}^+)^* \mid q \stackrel{\sigma}{\Longrightarrow}\}$ the set of traces from q. For a set of states P, we define P *After* $\alpha = \bigcup_{q\in P} q$ *After* α and $\text{Trace}(P) = \bigcup_{q\in P} \text{Trace}(p)$.

Let us define the timed variable-change conformance relation (**tvco**) between two TTS.

Definition 4 Let $P = \langle P, p_0, \Sigma_{\mathcal{H}\cup\mathcal{X}}\cup\mathbb{R}^+, \rightarrow_P\rangle$ and $Q = \langle Q, q_0, \Sigma_{\mathcal{H}\cup\mathcal{X}}\cup\mathbb{R}^+, \rightarrow_Q\rangle$ be the semantics of two VDTA.

$$P \textbf{ tvco } Q$$

iff

$$\forall\sigma \in \text{Trace}(q_0), \text{Out}(p_0 \textit{ After } \sigma) \subseteq \text{Out}(q_0 \textit{ After } \sigma)$$

where in the context of our model, Out(p) is a set of actions and each action corresponds to the values of the variables in a state from $p' \in p$ *After* σ. Recall that a state is a tuple made of a location and a snapshot over variables.

Consider the model in Figure 1 and let $q = (l_0, (0, 0, 0, 0.2))$ be a state where $(0, 0, 0, 0.2)$ are the values of the tuple (L, R, s, h). If σ is the sequence with a single action $a = \{L = 1, R = 1\}$, then q *After* $\sigma = \{q'\}$ where $q' = (l_2, 1, 1, 1, 0)$ and Out(q') = Out$(q$ *After* $\sigma) = \{L := 1, R := 1, s := 1\}$. Actions in Out$(q)$ are defined over memory-variables.

Definition 5 Consider a specification $\mathcal{A} = \langle L^{\mathcal{A}}, l_0^{\mathcal{A}}, \Sigma_{\mathcal{H}^{\mathcal{A}} \cup \mathcal{X}^{\mathcal{A}}}, \mathcal{X}^{\mathcal{A}}, \mathcal{X}_c^{\mathcal{A}}, \mathcal{H}^{\mathcal{A}}, \Delta^{\mathcal{A}} \rangle$ and an IUT $\mathcal{I} = \langle L^{\mathcal{I}}, l_0^{\mathcal{I}}, \Sigma_{\mathcal{H}^{\mathcal{I}} \cup \mathcal{X}^{\mathcal{I}}}, \mathcal{X}^{\mathcal{I}}, \mathcal{X}_c^{\mathcal{I}}, \mathcal{H}^{\mathcal{I}}, \Delta^{\mathcal{I}} \rangle$. Let $val_0^{\mathcal{A}}$ be a snapshot that assigns 0 to clocks and variables of $\mathcal{A}$ and $val_0^{\mathcal{I}}$ be a snapshot that assigns 0 to clocks and variables of $\mathcal{I}$. $\mathcal{I}$ conforms to $\mathcal{A}$ and we write $\mathcal{I}$ **tvco** $\mathcal{A}$ iff $[\![\mathcal{I}]\!]$ **tvco** $[\![\mathcal{A}]\!]$.

We remark that in the definition above, the IUT and the specification neither share their memory-variables nor their clocks variables. But, we will assume that there is a one to one correspondence between memory-variables of the IUT and memory-variables of the specification. Then, we say that an action over memory-variables of the IUT is equal to an action over memory-variables of the specification if and only if by replacing memory-variables by their correspondent we get the same action. Under that assumptions, we can compare actions from $\Sigma_{\mathcal{X}^{\mathcal{I}}}$ with actions from $\Sigma_{\mathcal{X}^{\mathcal{A}}}$ and we can check if a subset of $\Sigma_{\mathcal{X}^{\mathcal{I}}}$ is included in a subset of $\Sigma_{\mathcal{X}^{\mathcal{A}}}$.

4.3. Conformance Testing Algorithm

We propose Algorithm 1 that is an on the fly conformance testing algorithm for systems modeled with VDTA. Algorithm 1 follows the principle presented in Subsection 4.1.

An *observable memory-variable action* is an action over $\mathcal{X}$ that changes the value of at least one memory-variable. The tester chooses controllable actions over $\mathcal{X}_c^{\mathcal{A}}$ that enable to fire a discrete transition from a set of states Z. These actions belong to the set: $EnvOut(Z) = \{a \in \Sigma_{\mathcal{X}_c^{\mathcal{A}}} \mid \exists (l, val) \in Z. \exists l \xrightarrow{g, a'} l' \text{ s.t } val[a] \models g\}$. Observe that $EnvOut(Z)$ is empty if for every state in $q \in Z$, the constraint on each transition from the location in q is not satisfied by the snapshot in q. In example consider the specification in Figure 1. If $Z = \{(l_0, (0, 0, 0, 2))\}$, the unique transition from locations in Z is the transition $l_0 \xrightarrow{g, a} l_1$ where g is the constraint $L = 1 \vee R = 1$ and a is a *silent* action over $\emptyset$. The set $EnvOut(Z) = \{\{L := 1\}, \{R := 1\}, \{(L := 1), (R := 1)\}\}$ contains three actions; one action is the union of the two others. The execution of every action in $EnvOut(Z)$ makes the constraint $L = 1 \vee R = 1$ become true. As L and R are controllable memory-variables, the tester can execute any action in $\{\{L := 1\}, \{R := 1\}, \{(L := 1), (R := 1)\}\}$.

When the tester delays, the amount of the delay is not hazardous. It must belong to $Delay(Z)$, the set of time instants at which a constraint on a transition from a location in Z is satisfied. During the delay, the tester observes outputs from the implementation and checks their conformance with those expected from currents states of the specification. Let Z be the set of current states of the specification. Expected outputs in the specification belong to the set $ImpOut(Z)$ defined as follows: $ImpOut(Z) = \{a \in \Sigma_{\mathcal{X}^{\mathcal{A}}} \mid \exists (l, val) \in Z. \text{ s.t } \forall v \in \mathcal{X}. val[a](v) = a(v)\}$.

Algorithm 1 On-The-Fly Testing Algorithm for VDTA

Require: $\mathcal{A} = \langle L, l^0, \Sigma_{\mathcal{X} \cup \mathcal{H}}, \mathcal{X}, \mathcal{X}_c, \mathcal{H}, \Delta \rangle$
Require: IUT
1: $Z = \{(l^0, val^0)\}$
2: **while** TRUE **do**
3: operation = switch-randomly (action,delay,pass)
4: **if** operation == action **then**
5: **if** $EnvOut(Z) \neq \emptyset$ **then**
6: randomly choose $a \in EnvOut(Z)$
7: $Z \leftarrow Z$ After a
8: execute the action a on the IUT and let o be an observed action over $X^{\mathcal{I}}$.
9: **if** $o \notin ImpOut(Z)$ **then**
10: RETURN fail
11: **end if**
12: **end if**
13: **else if** operation == delay **then**
14: randomly choose $t \in Delays(Z)$
15: sleep for t time units and wake up on an observable variable action o.
16: **if** o occurs at $t' \leq t$ **then**
17: $Z \leftarrow Z$ *After* t'
18: **if** $o \notin ImpOut(Z)$ **then**
19: RETURN fail
20: **else**
21: $Z \leftarrow Z$ *After* o
22: **end if**
23: **else**
24: $Z \leftarrow Z\, After\, t$
25: **end if**
26: **else if** operation == pass **then**
27: RETURN pass
28: **end if**
29: **end while**

In Line 9 (Algorithm 1), we require a copy of each memory variables. One copy for the IUT and one copy for the specification $\mathcal{A}$. This allows to compare contents of memory-variables of the implementation with contents of their copies in the specification.

Each execution of Algorithm 1 constructs a sequence of delays and actions. Actions in sequences come either from the tester or the IUT. Actions from the tester are

executed on the IUT and actions from the IUT are observed by the tester. Sets of such sequences of delays and actions are *test cases*.

A test case can be represented by a TTS having the following properties:

- finiteness: it ensures that a testing experiment lasts for a finite time and obviously, *terminal states* in test cases are labelled with *fail* or *pass*.
- determinism: it ensures that in a time instant, a single action can be observed. Moreover from a state of the test case, there must be no choice between actions and delays; nor between input (action from the tester) and output (action from the IUT).

We remark that every state of a test case can have many outgoing transitions labelled with actions from the IUT (or outputs). This is because the tester is allowed to observe all possible actions from the IUT.
Algorithm 1 can be executed infinitely many times providing a (infinite) set of test cases. A set of test cases is called a *test suite*. A test case is executed on an implementation. An implementation may behave in a non deterministic way. Then different test runs of a test case with the same implementation may lead to different terminal states of the test case and hence to different verdicts.

Definition 6 A IUT **passes** a test case if and only if every test run leads to a final state of the test case labelled with the verdict *pass*.

4.4. Completeness

A test suite checks IUT conformance with respect to the specification and the relation **tvco**. Ideally, an implementation should pass the test suite if and only if it conforms. In this case the test suite is called *complete*. But, it may happen that one restricts to *sound* test suites. A test suite is *sound* if it never considers a correct IUT as faulty. On the other hand, a test suite which can ensure conformance but may reject conforming IUT is called *exhaustive*.

Definition 7 Let $\mathcal{A}$ be a specification and T be a test suite; then for the relation **tvco**:

T is complete means $\forall \mathcal{I}$:IUT, $\mathcal{I}$ **tvco** $\mathcal{A}$ iff $\mathcal{I}$ **passes** T.

T is sound means $\forall \mathcal{I}$:IUT, $\mathcal{I}$ **tvco** $\mathcal{A}$ implies $\mathcal{I}$ **passes** T.

T is exhaustive means $\forall \mathcal{I} \in$IUT, $\mathcal{I}$ **tvco** $\mathcal{A}$ if $\mathcal{I}$ **passes** T.

We provide a proof of the following proposition:

Proposition 8 Algorithm 1 is complete for $\mathcal{A}$ with respect to all possible test cases and the relation **tvco**.

We show that the algorithm is sound and exhaustive.

Soundness We would like to show that $\forall \mathcal{I}, \mathcal{I}$ **tvco** $\mathcal{A}$ implies $\mathcal{I}$ $passes$ T. We use a contradiction argumentation.
Suppose that it is not true that $\mathcal{I}$ $passes$ T. Then, there is an execution of the algorithm that returns (at step 3) the verdict $Fail$. That execution exhibits a trace $\sigma = \sigma'.a$ where σ' is a sequence of actions and a is an action. As the verdict is returned in Line 27 of the algorithm, a is necessarily an output of the implementation that is not expected by the specification. Consequently the **tvco** relation is violated. This is a contradiction the hypothesis.

Exhaustivity We show that $\forall \mathcal{I}, \mathcal{I}$ $passes$ T implies $\mathcal{I}$ **tvco** $\mathcal{A}$. We use a contradiction argument. We assume that is is not true that $\mathcal{I}$ **tvco** $\mathcal{A}$ and it is true that $\mathcal{I}$ $passes$ T. By definition $\neg(\mathcal{I}$ **tvco** $\mathcal{A})$ implies there is a trace σ, such that $Out(\mathcal{I}$ *After* $\sigma) \not\subseteq Out(\mathcal{A}$ *After* $\sigma)$. Equivalently, there is an action a such that $a \in Out(\mathcal{I}$ *After* $\sigma)$ and $a \notin Out(\mathcal{A}$ *After* $\sigma)$. In a first step we show that there is a run of our algorithm that reaches a step where a test case tc^{σ}, having σ as an observable trace, is constructed. We recall that by definition, a test case is not necessarily a sequence. The construction of tc^{σ} is done by induction on the length of σ; it works as follows:

- If $\sigma = \epsilon$ is an empty sequence, then according to Line 27 of the algorithm, tc^{σ} is the test case having a single state labelled with *pass*. There is no transition from that state.
- If σ is of the form $b.\beta$ and b is an output from the tester (or the environment), then according to Line 4 of the algorithm, tc^{σ} is the test case having a state q as initial state and q has an outgoing transition labelled with a. The target of that transition is the root of the test case tc^{β}. This is an inductive step.
- If σ is of the form $d.\beta$ and $d \in \mathbb{R}^{+}$ is a delay, then according to Line 13, tc^{σ} is the test case having a state q as initial state and q has an outgoing transition labelled with d. The target of that transition is the root of the test case tc^{β}.
- If σ is of the form $o.\beta$ and o is an output from the implementation. Then tc^{σ} is the test case having a state q as initial state. Outgoing transitions from q are labelled with all possible outputs from the implementation. If o is one of that outputs, there is an o-labelled transition from q to the root of tc^{β} if o is expected by the specification; otherwise there is an o-labelled transition from q to *fail*. In particular, according to the hypothesis, there will be an a-labelled transition from q to *fail*.

It should be clear that there exists at least one test run of tc^{σ} that leads to *fail*. Consequently it is not

true that I passes T. This is a contradiction with the hypothesis.

5. A Case Study

Now, we present some runs of Algorithm 1 when the IUT is the same as the specification, and when the IUT is a mutation of the specification. We consider the "Bimanual" control command system.

5.1. The IUT is the Specification

A state is represented by a tuple $(l, (c_1, c_2, c_3, c_4))$ where l is a location, and (c_1, c_2, c_3, c_4) are values of L, R, s and h respectively. Locations of implementations can not be observed by the tester. Only contents of memory-variables of the IUT are observable.
Initially, clock-variables and memory-variables are equal to zero. The algorithm starts with $Z = \{(l_0, (0,0,0,0))\}$. In Table 1 and Table 2 we present two scenarios starting with a same action a_1 and a same delay $t = 0.1$. The column $\frac{observation}{Z}$ presents observations of the tester (above the line) and states of the specification (below the line). Current location and values of clocks in the implementation are not observed; but they are presented in tables.

op	**EnvOut(Z)**	Choice	$\frac{Observation}{Z}$
action	$a_1 := \{L := 1\}$ $a_2 := \{R := 1\}$ $a_3 := \{a_1, a_2\}$	$a = a_1$	$\frac{(l_1,(1,0,0,0))}{\{(l_1,(1,0,0,0))\}}$
delay		$t = 0.1$	$\frac{(l_1,(1,0,0,0.1))}{\{(l_1,(1,0,0,0.1))\}}$
Sub-scenario 1			
delay		$t = 0.2$	$\frac{(l_1,(1,0,0,0.3))}{\{(l_1,(1,0,0,0.3))\}}$
action	$a_1 := \{R := 1\}$	$a = a_1$	$\frac{(l_2(1,1,1,0.3))}{\{(l_2,(1,1,1,0.3))\}}$
delay		$d = 1.2$	$\frac{(l_2,(1,1,1,1.5)}{\{(l_2,(1,1,1,1.5))\}}$
action	$a_1 := \{L := 0\}$ $a_2 := \{R := 0\}$ $a_3 := \{a_1, a_2\}$	$a = a_3$	$\frac{(l_0,(0,0,0,1.5))}{\{(l_0,(0,0,0,1.5))\}}\}$
Sub-scenario 2			
delay		$t = 0.5$	$\frac{(l_3,(1,0,0,0.6))}{\{(l_3,(1,0,0,0.6))\}}$
action	$a_1 := \{R := 1\}$	$a = a_1$	$\frac{(l_3,(1,1,0,0.6)}{\{(l_3,(1,1,0,0.6))\}}$
delay		$t = 1.2$	$\frac{(l_3,(1,1,0,1.8))}{\{(l_3,(1,1,0,1.8))\}}$
action	$a_1 := \{L := 0\}$ $a_2 := (R := 0)$ $a_3 := \{a_1, a_2\}$	$a = a_3$	$\frac{(l_0,(0,0,0,1.8)}{\{(l_0,(0,0,0,1.8))\}}$

Table 1. Simulation of Algorithm 1

We give some comments on Sub-scenario 1 in Table 1; they can be adapted easily on Sub-scenario 2. The tester executes the sequence of choices a_1, $t = 0.1$, $t = 0.2$, a_1, $t = 1.2$ a_3. When it executes a_1 for the first time, it does not observe any change on uncontrollable memory-variables of the implementation; this conforms to the specification. This is also true when it executes $t = 0.1$, $t = 0.2$. But after the execution of a_1 for the second time, the tester observes that the value of s has changed to 1; this conforms to the specification. After the execution of a_3 the value of s changes to 0 in accordance to the specification.

5.2. The IUT is a Mutant of the Specification

Figure 2 presents a model of the IUT that is obtained from the specification (see Figure 1) by changing the constraint on the clock t. The new constraint compares t with 0.8.

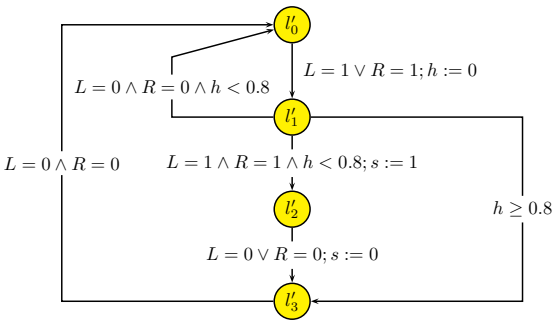

Figure 2. A VDTA model for a mutant IUT

Table 1 presents two executions of Algorithm 1 against an implementation of the model in Figure 2 and the specification (see Figure 1).

Sub-scenario 1 conforms to the specification while Sub-scenario 2 does not : the algorithm returns the fail verdict. The non-conformance occurs because the value of s has changed to 1 after 0.6 time units after the left button has been pushed. But this is not allowed by the specification in Figure 1.

op	**EnvOut(Z)**	Choice	$\frac{Observation}{Z}$
action	$a_1 := \{L := 1\}$ $a_2 := \{R := 1\}$ $a_3 := \{a_1, a_2\}$	$a = a_1$	$\frac{(L'_1,(1,0,0,0))}{\{(l_1,(1,0,0,0)\}}$
delay		$t = 0.1$	$\frac{(l'_1,(1,0,0,0.1))}{\{(l_1,(1,0,0,0.1))\}}$
Sub-scenario 1			
delay		$t = 0.2$	$\frac{(l'_1,(1,0,0,0.3))}{\{(l_1,(1,0,0,0.3))\}}$
action	$a_1 := \{R := 1\}$	$a = a_1$	$\frac{(l'_2,(1,1,1,0.3))}{\{(l_2,(1,1,1,0.3))\}}$
delay		$t = 1.2$	$\frac{(l'_2,(1,1,1,1.5))}{\{(l_2,(1,1,1,1.5))\}}$
action	$a_1 := (L := 0)$ $a_2 := (R := 0)$ $a_3 := \{a_1, a_2\}$	$a = a_3$	$\frac{(l'_0,(0,0,0,1.5))}{\{(l_0,(0,0,0,1.5))\}}$
Sub-scenario 2			
delay		$t = 0.5$	$\frac{(l'_1,(1,0,0,0.6))}{\{(l_1,(1,0,0,0.6))\}}$
action	$a_1 := \{R := 1\}$	$a = a_1$	$\frac{(l'_2,(1,1,1,0.6))}{\{(l_1,(1,1,0,0.6))\}}$
Verdict:			FAIL

Table 2. Simulation of Algorithm 1

6. Concluding Remarks

In this paper, we presented a test generation algorithm for a new model adapted to describe systems with only urgent transitions and driven by variables: only variable changing are considered as events and transitions are fired as soon as constraints become true. The test generation algorithm is proved to be sound and exhaustive. Then we have presented a case study to explain our approach.

As a future work, we intend to add formal verification approaches (such as symbolic model checking) to increase the testing efficiency. Indeed, at this step our method does not provide a way to focus on a particular part of the system. We are currently working on new approaches to lead the test session into particular states.

References

[1] R. Alur and D. Dill, "A theory of timed automata", *Theoretical Comput. Sci.*, vol. 126, pp. 183–235, 1994.

[2] H. B. Mokadem, B. Bérard, P. Bouyer, and F. Laroussinie, "A New Modality for Almost Everywhere Properties in Timed Automata", in *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005*, volume 3653 of *Lecture Notes in Computer Science*, 2005, pp. 110–124.

[3] R. Cardell-Oliver, "Conformance Testing of Real-Time Systems with Timed Automata Specifications", *Formal Aspects of Computing Journal*, vol. 12, no. 5, pp. 350–371, 2000.

[4] B. Nielsen and A. Skou, "Automated Test Generation from Timed Automata", in *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001*, volume 2031 of *Lecture Notes in Computer Science*, 2001, pp. 343–357.

[5] J. Springintveld, F. Vaandrager, and P. R. D'Argenio, "Timed Testing Automata", *Theoretical Comput. Sci.*, vol. 254, no. 254, pp. 225–257, 2001.

[6] A. EnNouaary, R. Dssouli, and F. Khendek, "Timed Wp-Method: Testing Real-Time Systems", *IEEE Transactions on Software Engineering (TSE)*, vol. 28, no. 11, pp. 1023–1038, 2002.

[7] J. Tretmans, "Test generation with Inputs, Outputs, and Repetitive Quiescence", vol. 17, pp. 103–120, 1996.

[8] M. Núñez and I. Rodríguez, "Conformance Testing Relations for Timed Systems", in *Formal Approaches to Software Testing, 5th International Workshop, FATES 2005, Edinburgh, UK, July 11, 2005, Revised Selected Papers*, volume 3997 of *Lecture Notes in Computer Science*, 2005, pp. 103–117. Springer.

[9] M. Mikucionis, K. G. Larsen, and B. Nielsen, "T-UPPAAL: Online Model-based Testing of Real-Time Systems", in *19th IEEE International Conference on Automated Software Engineering (ASE 2004), 20-25 September 2004, Linz, Austria*, 2004, pp. 396–397. IEEE Computer Society.

[10] M. Krichen and S. Tripakis, "Black-Box Conformance Testing for Real-Time Systems", in *Model Checking Software, 11th International SPIN Workshop, Barcelona, Spain, April 1-3, 2004*, volume 2989 of *Lecture Notes in Computer Science*, 2004, pp. 109–126. Springer.