

Test Generation for Duration Systems

Saddek Bensalem,
Moez Krichen
Verimag Laboratory,
Centre Equation 2,
avenue de Vignate,
38610, Gières, France.
saddek.bensalem@imag.fr,
moez.krichen@imag.fr

Lotfi Majdoub,
Riadh Robbana
LIP2 Laboratory and
Polytechnic School of Tunisia
lotfi.majdoub@ensi.rnu.tn,
riadh.robbana@fst.rnu.tn

Stavros Tripakis
Verimag Laboratory and
Cadence Berkeley Labs,
1995 University avenue,
Suite 460, Berkeley,
CA 94704, USA.
tripakis@cadence.com

Abstract

We are interested in generating tests for duration systems modeling real-time systems. The specification of a duration system is given as a duration graph. Duration graphs are an extension of timed graphs and are suitable for modeling the accumulated times spent by computations within the considered duration system. In this paper, we present a method for generating tests for duration systems based on the so-called approximation method. First, we use the approximation method to extend the specification model into an approximate model. The latter contains the digitization computations of the initial model. Test trees are then extracted from the approximate model. We explain how the obtained digital-test trees can be executed in an analog-fashion.

Keywords: Real-Time, Duration Systems, Testing, Approximation method, Digitization.

1. INTRODUCTION

Duration systems are real-time systems. A duration system encodes both constraints on the delays separating consecutive discrete events as well as constraints on accumulated delays spent by a given computation of the system.

Timed graphs (also called timed automata) constitute a powerful formalism widely adopted for modeling real-time systems [2]. Duration Variable Timed Graphs (DVTG for short) [4] are an extension of the timed graph model. They are used as a formalism to describe duration systems. A given DVTG has a finite set of continuous real variables that can be stopped in some locations ($rate=0$) and resumed in some other locations ($rate=1$). These variables are called *duration variables*. They can model some temporal behaviors of a real-time system such as the accumulated delays spent by some computations at some particular locations.

Testing is an important validation activity with a wide range of goals. In this work, we are interested in conformance testing. Conformance testing aims to check whether a given implementation conforms to its specification. The testing procedure consists in generating test cases by deriving them from the specification and then applying them to the implementation under test (IUT). For the case of real-time systems, conformance is defined with respect to both discrete actions occurrence and time elapsing. That is, for the IUT to be conforming to the specification it must generate correct outputs at correct timings.

Generating exhaustive test remains expensive and in some cases impossible, in particular for real-time systems. Springintveld et al [16] proved that exhaustive testing of deterministic timed automata with dense time is theoretically possible, but highly infeasible. Hence, some works define a criteria for selecting test cases that is generated automatically such as coverage criteria (e.g., transition and location coverage) [8, 9, 11]. Other works try to define purposes of test and generating test cases according to those purposes [14, 12]. An other way to alleviate this problem is the following.

For testing real-time systems, most works [3, 6, 9, 10, 11, 13] use discretization techniques to reduce the infinite state space into a finite (or at least countable) state space then adapt the existing untimed test case generation algorithm [17]. There is a lot of similarities between (model-based) testing and system verification. People working on real-time testing borrow a lot of techniques from the real-time verification field to generate test cases automatically (e.g., discretization techniques based on the so-called *region graph*, model checking techniques, etc.). [8] show how it is possible to represent coverage criteria and test purposes as a reachability property and use the model checking tool UPPAAL [8] to generate test cases with respect to the considered criteria.

It is well known that the verification of real time system is mainly possible thanks to the decidability of the reachability problem for systems represented as timed automata [1]. On the other side, it has been shown that the reachability problem is undecidable for timed graphs extended with one duration variable [5]. Consequently it is less obvious to use the classical verification techniques for testing DVTG.

In this paper, we propose a test generation method for duration systems. We use the approximation method extending a given DVTG-IO specification to another specification called *approximate model*. Our method consists of generating tests from the approximate model. The behavior of a test is derived from the approximate specification model. It is described by a tree, called *test tree*. In order to construct the test tree, we adapt the untimed test generation algorithm of [17] and apply it to the approximate specification model. We also give the way of how the obtained test trees shall be executed.

The rest of the paper is organized as follows. Section 2 introduces the DVTG model. Section 4 gives the approximation method. Section 5 defines the type of tests we consider. Section 6 gives our testing method and shows how test trees are derived from the approximate model. Concluding remarks are presented in section 7.

2. DURATION VARIABLE TIMED GRAPHS WITH INPUTS AND OUTPUTS (DVTG-IO)

In this section, we introduce the formalism we use for describing duration systems, called *duration variable timed graph with inputs and outputs* (DVTG-IO for short).¹ DVTG-IO is an extension of the well-known timed automata model defined in [2]. We give here the formal definition and the operational semantics of this model. Then, we present some terminologies that will be used in our test method. In the last subsection, we illustrate with a simple example.

2.1. The DVTG-IO model

A DVTG-IO is described by a finite set of locations and a transition relation between these locations. In addition, the system has a finite set of duration variables that are constant slope continuous variables, each of them changes continuously with a rate in $\{0, 1\}$ at each location of the system. Transitions between locations are conditioned by arithmetical constraints on the values of the duration variables. When a transition is taken, a subset of duration variables should be reset and an action should be executed, this action can be either an input action, an output action or an unobservable action.

¹A similar model to the DVTG-IO model is introduced in [4].

We consider X a finite set of duration variables. A guard on X is a boolean combination of constraints of the form $x \prec c$ where $x \in X$ and $c \in \mathbb{N}$, $\prec \in \{<, \leq, >, \geq\}$. Let $\Gamma(X)$ be the set of guards on X .

A DVTG-IO is a tuple $M = (Q, q_0, E, X, Act, \gamma, \alpha, \delta, \partial)$ where:

- Q is a finite set of locations;
- q_0 is the initial location;
- $E \subseteq Q \times Q$ is a finite set of transitions between locations;
- $Act = In \cup Out \cup \{\tau\}$ is the union of a finite set of input actions ($a?$, $b?$, $c?$, etc.), a finite set of output actions ($a!$, $b!$, $c!$, etc.) and τ the unobservable action τ ;
- $\gamma : E \rightarrow \Gamma(X)$ associates to each transition a guard which should be satisfied by the duration variables whenever the transition is taken;
- $\alpha : E \rightarrow 2^X$ gives for each transition the set of duration variables that should be reset when the transition is taken;
- $\delta : E \rightarrow Act$ gives for each transition the action that should be executed when the transition is taken;
- $\partial : Q \times X \rightarrow \{0, 1\}$ associates with each location q and each duration variable x the rate at which x changes continuously while the computation is at q .

2.2. The state graph of a DVTG-IO

Let R^+ be the set of nonnegative reals and \mathbb{N} the set of nonnegative integers. The semantics of a DVTG-IO are defined in terms of a state graph over states of the form $s = (q, \nu)$ where $q \in Q$ and $\nu : X \rightarrow R^+$. ν is a valuation function that assigns a real value to each duration variable.

Let S_M be the set of states of M . Most of the time, S_M is likely to contain infinitely many states since the values of the duration variables are taken in R^+ . The initial state of a given DVTG-IO is $(q_0, \vec{0})$, where q_0 is the initial location and $\vec{0}$ the valuation assigning 0 to each duration variable.

Given a valuation ν and a guard g , we denote by $\nu \models g$ the fact that valuation of g under the valuation ν is true.

We define two types of transitions between states.

1. Discrete Transitions of the form $(q, \nu) \xrightarrow{a} (q', \nu')$ where:

- $(q, q') \in E$; $\delta(q, q') = a$;
- $\nu \models \gamma(q, q')$;
- $\forall x \in X \setminus \alpha(q, q') : \nu'(x) = \nu(x)$;
- $\forall x \in \alpha(q, q') : \nu'(x) = 0$.

2. Timed transitions of the form $(q, \nu) \xrightarrow{t} (q, \nu')$ such that:

- $t \in R$;
- $\forall x \in X : \nu'(x) = \nu(x) + \partial(q, x) * t$.

The first type of transitions correspond to moves between locations due to the execution of discrete actions from E . The second type of transitions correspond to time progress at some location q .

A *path* of the DVTG-IO is a sequence of transitions of the form

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \cdots \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s_n$$

such that:

- $s_0 = (q_0, \vec{0})$;
- $\forall i = 1, \dots, n : a_i \in Act \cup R^+$;
- $\forall i = 1, \dots, n : s_{i-1} \xrightarrow{a_i} s_i$ is a transition of the DVTG-IO.

A state (q, ν) is called an integer state of S_M if $\nu : X \longrightarrow N$. We denote by $N(S_M)$ the set of integer states of S_M .

2.3. Computation sequences and timed words

We introduce the notion of computation sequences of a DVTG-IO. These sequences are defined as finite sequences of configurations. A configuration is a pair $\langle s, t \rangle$ where s is a state and t is a time value in R^+ . Let C_M be the set of configurations of M .

A configuration $\langle s, t \rangle$ is called an integer configuration if $t \in N$. We denote by $N(C_M)$ the set of integer configurations.

Clearly, we can easily make the link between states and configurations. A configuration can be seen as the state of the DVTG-IO extended with a new duration variable. The new duration variable measures the amount of time which has elapsed since the beginning of the computation. This duration variable is never stopped or reset. We call it the *observation clock* of the DVTG-IO. For instance, consider the path $\lambda = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \cdots \xrightarrow{a_{n-1}} s_{n-1} \xrightarrow{a_n} s_n$. The corresponding computation sequence is $\sigma = \langle s_0, t_0 \rangle \xrightarrow{a_1} \langle s_1, t_1 \rangle \xrightarrow{a_2} \langle s_2, t_2 \rangle \cdots \xrightarrow{a_{n-1}} \langle s_{n-1}, t_{n-1} \rangle \xrightarrow{a_n} \langle s_n, t_n \rangle$ such that $t_0 = 0$ and $\forall i = 1, \dots, n : t_i = \sum_{k=1}^i \text{time}(a_k)$ where $\text{time}(a_k) = 0$ if $a_k \in \text{Act}$; and $\text{time}(a_k) = a_k$ if $a_k \in R^+$.

Each pair $\langle s_i, t_i \rangle$ is called an *extended state*. We call $CS(M)$ be the set of computation sequences of M .

We introduce now the notion of timed words. A *timed word* is a finite sequence of timed actions. A *timed action* is a pair $\llbracket b, t \rrbracket$, where $b \in \text{Act}$ and $t \in R^+$. The pair $\llbracket b, t \rrbracket$ means that action a took place precisely at time when the observation clock was equal to t . Thus a timed word is of the form $\omega = \llbracket b_1, t_1 \rrbracket \llbracket b_2, t_2 \rrbracket \cdots \llbracket b_n, t_n \rrbracket$ where for $i = 1, \dots, n$, $b_i \in \text{Act}$ and $t_i \in R^+$. If for each $i = 1, \dots, n$, $t_i \in N$ then ω is said to be an integer timed word. Consider the following computation sequence $\sigma = \langle s_0, t_0 \rangle \xrightarrow{a_1} \langle s_1, t_1 \rangle \xrightarrow{a_2} \langle s_2, t_2 \rangle \cdots \xrightarrow{a_{n-1}} \langle s_{n-1}, t_{n-1} \rangle \xrightarrow{a_n} \langle s_n, t_n \rangle$ ² of the DVTG-IO M (i.e., $\sigma \in CS(M)$). Clearly, there exists a unique timed word ω corresponding to the computation sequence σ . The timed word ω is obtained as follows. Let $1 \leq i_1 < \dots < i_N \leq n$ such that for each $j = 1, \dots, N$, $a_{i_j} \in \text{Act}$ and for each $i \notin \{i_1, i_2, \dots, i_N\}$, $a_i \in R^+$. Then, the timed word corresponding to σ is $\omega = \llbracket b_1, t'_1 \rrbracket \llbracket b_2, t'_2 \rrbracket \cdots \llbracket b_N, t'_N \rrbracket$ where for each $j = 1, \dots, N$: $b_j = a_{i_j}$; $t'_i = \sum_{k=1}^i \text{time}(a_k)$.

For simplicity, we may write: $s_0 \xrightarrow{\omega} s_n$. The timed word ω is said to be an *accepted timed word* of the DVTG-IO M . Let $L(M)$ be the set of accepted timed words of M . We suppose that $L(M)$ contains the empty timed word as well.

We need to introduce the notion of *extended timed words* as well. For the computation sequence σ above, if $a_n \in R^+$ then the corresponding extended timed word will be

$$\omega_\diamond = \llbracket b_1, t'_1 \rrbracket \llbracket b_2, t'_2 \rrbracket \cdots \llbracket b_N, t'_N \rrbracket \llbracket \diamond, t'_{N+1} \rrbracket$$

where

$$t'_{N+1} = t_n = \sum_{k=1}^i \text{time}(a_k)$$

and \diamond is a special symbol meaning that no discrete action took place within the interval $(t'_N, t'_{N+1}]$.

We write $s_0 \xrightarrow{\omega_\diamond} s_n$ as well and ω_\diamond is said to be an accepted extended timed word of the DVTG-IO M . We denote $L^\diamond(M)$ the set of accepted timed words and extended timed words of M .

²We make the natural assumption that for each $i = 1, \dots, n$ $\{a_{i-1}, a_i\} \not\subseteq R^+$. That is at least one of each two consecutive labels is a discrete action from Act .

We define the concatenation operator over timed words. For the two timed words ω_1 and ω_2 such that:

$$\omega_1 = \llbracket b_1, t'_1 \rrbracket \llbracket b_2, t'_2 \rrbracket \cdots \llbracket b_n, t'_n \rrbracket$$

and

$$\omega_2 = \llbracket b_{n+1}, t'_{n+1} \rrbracket \llbracket b_{n+2}, t'_{n+2} \rrbracket \cdots \llbracket b_{n+m}, t'_{n+m} \rrbracket$$

we have

$$\omega_1 \cdot \omega_2 = \llbracket b_1, t'_1 \rrbracket \llbracket b_2, t'_2 \rrbracket \cdots \llbracket b_n, t'_n \rrbracket \llbracket b_{n+1}, t'_{n+1} \rrbracket \cdots \llbracket b_{n+m}, t'_{n+m} \rrbracket.$$

By convention, for any $t_n, t_{n+1} \in Act$ and $b_{n+1} \in Act$ we assume that:

$$\llbracket \diamond, t_n \rrbracket \cdot \llbracket b_{n+1}, t_{n+1} \rrbracket = \llbracket b_{n+1}, t_{n+1} \rrbracket.$$

2.4. An example of a DVTG-IO

A simple example of a DVTG-IO M is given in Figure 1. The DVTG-IO has: 5 locations: $\{q_0, q_1, q_2, q_3, q_4\}$; 2 input actions: $\{a?, c?\}$; 2 output actions: $\{b!, d!\}$; 3 duration variables $\{t, x, z\}$.

The initial location of the DVTG-IO is q_0 . The duration variable t is the observation clock of M . It measures time elapsing since the beginning of each computation.

According to the figure, action $a?$ is allowed to happen no later than one time-unit after the beginning of the whole computation. Similarly, actions $c?$ and $d!$ shall happen exactly at $t = 1$ and $t = 2$, respectively.

The guard “ $x = 1$ ”, on the transition between locations q_2 and q_3 , encodes the fact that the input action $c?$ shall be emitted exactly one time-unit after the execution of $a?$ (x is reset as soon as $a?$ is emitted).

The duration variable z is stopped at both locations q_1 and q_3 . Thus, it allows to measure the total accumulated time spent within the other locations (i.e., q_0, q_2 and q_4).

3. THE CONFORMANCE RELATION TIOCO FOR DVTG-IO

3.1. Definition of tioco

In the sequel, for simplicity we consider only DVTG-IO all the actions of which are observable (i.e., No transitions are labeled with τ).

Next, we recall the definition of the *timed input-output conformance* relation *tioco* first introduced in [11] and which is in turn inspired from the “untimed” conformance relation *ioco* of [17].

The conformance relation *tioco* was initially introduced to the case of Timed automata with inputs and outputs. Next, we extend this relation in a straightforward way to the case of DVTG-IO. In order to formally define the conformance relation, we define a number of operators. Given a DVTG-IO M and a timed word $\omega \in L^\diamond(M)$, M after ω is the set of all states of M that can be reached after the execution of the timed word ω .

Formally: M after $\omega = \{s \in S(M) \mid : s_0 \xrightarrow{\omega} s\}$ where s_0 is the initial state of M . Given state $s \in S(M)$, $\text{elapse}(s)$ is the set of all delays which can elapse from s without M making any observable action. Formally: $\text{elapse}(s) = \{t > 0 \mid \exists s' \in S(M) : s \xrightarrow{t} s'\}$.

Given state $s \in S(M)$, $\text{out}(s)$ is the set of all observable “events” (outputs or the passage of time) that can occur when the system is at state s . Formally:

$$\text{out}(s) = \{a \in Out \mid \exists s' \in S(M) : s \xrightarrow{a} s'\} \cup \text{elapse}(s).$$

The definition naturally extends to a set of states S : $\text{out}(S) = \bigcup_{s \in S} \text{out}(s)$.

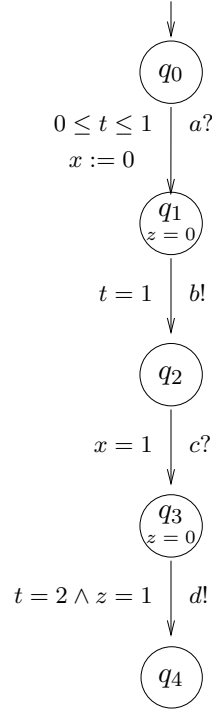


FIGURE 1: An example of a DVTG-IO.

The *timed input-output conformance relation*, denoted tioco , is defined as

$$M_I \text{ tioco } M_S \equiv \forall \omega \in L^\diamond(M_S) : \text{out}(M_I \text{ after } \omega) \subseteq \text{out}(M_S \text{ after } \omega).$$

The relation states that an implementation M_I conforms to a specification M_S if and only if for any observable behavior ω of M_S , the set of observable outputs of M_I after any behavior “matching” ω must be a subset of the set of possible observable outputs of M_S .

Notice that observable outputs are not only observable output actions but also time delays.

3.2. Example

We consider the DVTG-IO M of Figure 1. We consider this DVTG-IO as the specification model. Three possible implementations of M are given in Figure 2, namely M_I^1 , M_I^2 and M_I^3 . Among these three implementations only M_I^3 conforms to M with respect to tioco .

First, $M_I^1 \not\text{tioco } M$ since it produces a wrong output after receiving input $a?$. That is instead of emitting output $b!$ (as stated in the specification) it emits output $d!$. The (extended) timed word that we may use for detecting non-conformance is

$$\omega_\diamond = \llbracket a?, 0 \rrbracket \llbracket \diamond, 1 \rrbracket.$$

In fact we have

$$\text{out}(M \text{ after } \omega_\diamond) = \{b!\};$$

and

$$\text{out}(M_I^1 \text{ after } \omega_\diamond) = \{d!\}.$$

First, $M_I^2 \not\text{tioco } M$ since it does not wait enough time before emitting output $b!$ when it reaches location q_1 as stated in M . The timed word we may use in this case is simply

$$\omega = \llbracket a?, 0 \rrbracket.$$

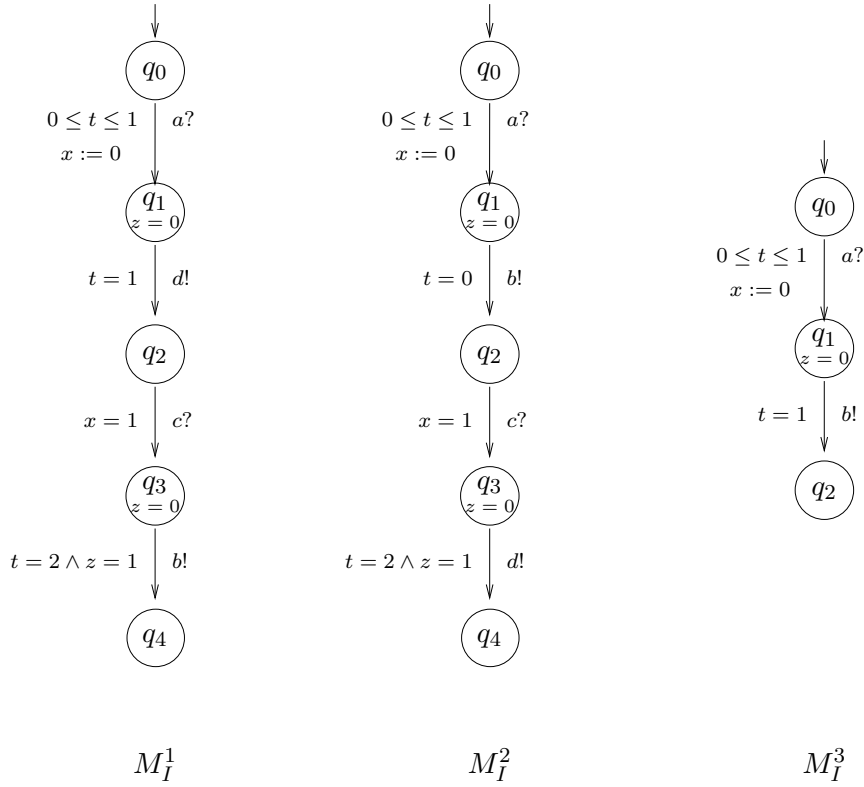


FIGURE 2: Possible implementations of the DVTG-IO given in Figure 1.

It is not difficult to see that

$$\begin{aligned}
 & b! \notin \text{out}(M \text{ after } \omega); \\
 & \quad \text{while} \\
 & b! \in \text{out}(M_I^1 \text{ after } \omega).
 \end{aligned}$$

Finally, we clearly have M_I^3 tioco M since the set of computation sequences of M_I^3 is a subset of the set of computation sequences of M . That is

$$CS(M_I^3) \subseteq CS(M).$$

4. THE APPROXIMATION METHOD

The approximation method we introduce in this section bears some similarity with the method proposed in [15]. This method is used for the verification of reachability properties for duration systems.

Next, we describe the method and we show how we use it for testing real-time systems modeled as DVTG-IO. The approximation method we present is mainly based on the so-called discretization technique. In [7], the latter is called digitization technique instead. It consists in associating a discrete computation to each possible analog computation of the specification. We show with an example that the obtained discrete computations may not belong to the specification model. Thus, we propose an approximate model that contains all discrete computations. The approximate model is generated from the specification by the approximation method.

4.1. The digitization technique

We present the notion of digitization introduced in [7]. We adapt it to the case of the DVTG-IO model. We first introduce some definitions and notations. Let $t \in \mathbb{R}^+$ and $n \in \mathbb{N}$. If $t \in [n, n + 1)$

then we use the following notation: $\lfloor t \rfloor = n$; and $\lceil t \rceil = n + 1$. For instance we have: $\lfloor 819.3 \rfloor = 819$ and $\lceil 819.3 \rceil = 820$.

For $t \in \mathbb{R}^+$ and $\xi \in [0, 1)$, if $t \leq \lfloor t \rfloor + \xi$ then $\lfloor t \rfloor_\xi = \lfloor t \rfloor$. Otherwise, $\lfloor t \rfloor_\xi = \lceil t \rceil$.

For instance for $t = 819.3$:

- For $\xi = 0.2$, $\lfloor 819.3 \rfloor_{0.2} = 820$;
- For $\xi = 0.3$, $\lfloor 819.3 \rfloor_{0.3} = 819$;
- For $\xi = 0.4$, $\lfloor 819.3 \rfloor_{0.4} = 819$.

The real value ξ is called the *digitization quantum*. Clearly for $\xi = 0$, we have for any $t \in \mathbb{R}^+$: $\lfloor t \rfloor_0 = \lfloor t \rfloor$. Thus for any $t \in \mathbb{R}^+$ and $\xi \in [0, 1)$, we have: $t \in (\lfloor t \rfloor_\xi + \xi - 1, \lfloor t \rfloor_\xi + \xi]$ which is equivalent to: $-1 + \xi < t - \lfloor t \rfloor_\xi \leq \xi$.

Next we extend the notion of digitization above to the case of computation sequences. For that purpose, consider the following computation sequence

$$\sigma = \langle (q_0, \nu_0), t_0 \rangle \xrightarrow{a_1} \langle (q_1, \nu_1), t_1 \rangle \cdots \xrightarrow{a_n} \langle (q_n, \nu_n), t_n \rangle.$$

It is not difficult to see that for each duration variable x of the DVTG-IO and each $k = 1, \dots, n$, we have:

$$\nu_k(x) = \sum_{i=j_k+1}^{k-1} \partial(q_i, x) * (t_{i+1} - t_i)$$

such that

$$j_k = \max(\{-1\} \cup \{j \mid j \leq k \wedge x \in \alpha(q_j, q_{j+1})\}).^3$$

Given a digitization quantum $\xi \in [0, 1)$, the digitization of the computation sequence σ is the *integer computation sequence*

$$[\sigma]_\xi = \langle (q_0, \nu_0^\xi), [t_0]_\xi \rangle \xrightarrow{a_1^\xi} \langle (q_1, \nu_1^\xi), [t_1]_\xi \rangle \cdots \xrightarrow{a_n^\xi} \langle (q_n, \nu_n^\xi), [t_n]_\xi \rangle.$$

where for each $k = 1, \dots, n$ and for each duration variable x we have: $\nu_k^\xi(x) = [\nu_k(x)]_\xi$. We denote $[(q_k, \nu_k)]_\xi = (q_k, \nu_k^\xi)$.

The labels a_k^ξ are defined as follows. If $a_k \in Act$ then $a_k^\xi = a_k$. Otherwise, $a_k^\xi = [t_k]_\xi - [t_{k-1}]_\xi$.

We denote $Digit(CS(M))$ the set of digitizations of all the real computation sequences of the DVTG-IO M . Notice that $Digit(CS(M))$ is a countable set while $CS(M)$ is not.

We next extend the digitization technique to the case of timed words. Consider the following timed word

$$\omega = \llbracket b_1, t_1 \rrbracket \llbracket b_2, t_2 \rrbracket \cdots \llbracket b_n, t_n \rrbracket.$$

For the digitization quantum $\xi \in [0, 1)$, the integer timed word corresponding to ω is

$$[w]_\xi = \llbracket b_1, [t_1]_\xi \rrbracket \llbracket b_2, [t_2]_\xi \rrbracket \cdots \llbracket b_n, [t_n]_\xi \rrbracket.$$

We denote $Digit(L(M))$ the set of digitizations of all the timed words accepted by the DVTG-IO M . Clearly, $Digit(L(M))$ is a countable set.

It is possible to make the general observation that the digitization technique allows to transform an uncountable set X to a countable one $Digit(X)$.

³The integer value j_k corresponds to the last position at which x was reset before reaching the k^{th} position.

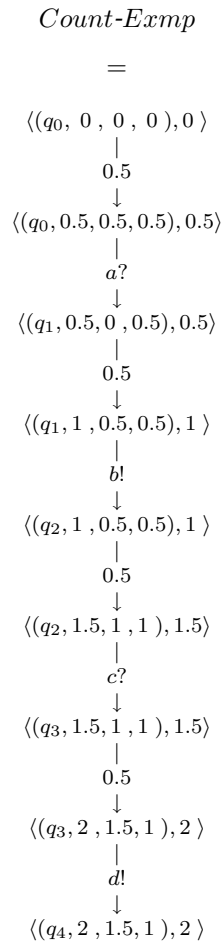
A legitimate question one may ask is whether $Digit(X) \subseteq X$ or not. In our case given a DVTG-IO M , the question is

“ $Digit(CS(M)) \subseteq CS(M)$ or not?”.

Next we prove, by means of an example, that the statement above is not always true.

A counter example: We consider the DVTG-IO M given in Figure 1 . We give a computation sequence of this DVTG-IO such that for any digitization quantum the corresponding integer computation sequence is not a element of $CS(M)$.

The computation sequence we consider is the following:



Note that each of the extended states above is of the form $\langle (q_i, t, x, z), t \rangle$. That is t is both a duration variable of the DVTG-IO and its observation clock.

Depending on the value of the digitization quantum ξ , the digitization of the computation sequence *Count-Exmp* may lead to the one of the following integer computation sequences:

$0 \leq \xi \leq 0.5$	$0.5 < \xi < 1$
$[Count-Exmp]_\xi$	$[Count-Exmp]_\xi$
=	=
$\langle\langle q_0, 0, 0, 0 \rangle, 0\rangle$	$\langle\langle q_0, 0, 0, 0 \rangle, 0\rangle$
 1	 0
$\langle\langle q_0, 1, 1, 1 \rangle, 1\rangle$	$\langle\langle q_0, 0, 0, 0 \rangle, 0\rangle$
 <i>a</i> ?	 <i>a</i> ?
$\langle\langle q_1, 1, 0, 1 \rangle, 1\rangle$	$\langle\langle q_1, 0, 0, 0 \rangle, 0\rangle$
 0	 1
$\langle\langle q_1, 1, 1, 1 \rangle, 1\rangle$	$\langle\langle q_1, 1, 0, 0 \rangle, 1\rangle$
 <i>b</i> !	 <i>b</i> !
$\langle\langle q_2, 1, 1, 1 \rangle, 1\rangle$	$\langle\langle q_2, 1, 0, 0 \rangle, 1\rangle$
 1	 0
$\langle\langle q_2, 2, 1, 1 \rangle, 2\rangle$	$\langle\langle q_2, 1, 1, 1 \rangle, 1\rangle$
 <i>c</i> ?	 <i>c</i> ?
$\langle\langle q_3, 2, 1, 1 \rangle, 2\rangle$	$\langle\langle q_3, 1, 1, 1 \rangle, 1\rangle$
 0	 1
$\langle\langle q_3, 2, 2, 1 \rangle, 2\rangle$	$\langle\langle q_3, 2, 1, 1 \rangle, 2\rangle$
 <i>d</i> !	 <i>d</i> !
$\langle\langle q_4, 2, 2, 1 \rangle, 2\rangle$	$\langle\langle q_4, 2, 1, 1 \rangle, 2\rangle$

It is not difficult to check that none of the two sequences above is an element of $CS(M)$.

4.2. Digital approximate model

Next, we define an abstraction of a given DVTG-IO M . We call it the *digital approximate model* of the DVTG-IO. The intuition is that we want to build an automaton which accepts the set of integer computation sequences of M (or at least a subset of it).

We denote $Dig-Approx(M)$ the digital approximate model of the DVTG-IO M . $Dig-Approx(M)$ is defined as follows. Each node of $Dig-Approx(M)$ is a nonempty set of integer states of M . The initial node of $Dig-Approx(M)$ is $\{s_0\}$ where $s_0 = (q_0, \vec{0})$ and q_0 is the initial location of M .

The edges between nodes of $Dig-Approx(M)$ are labeled with actions from $Act \cup \{1\}$. Let S be a set of integer states of M . For $a \in Act \cup \{1\}$, we define S' the set of (integer) states that can be reached after executing a and starting from a state in S as follows:

$$S' = \{s' \in N(S_M) \mid \exists s \in S : s \xrightarrow{a} s'\}$$

where $s \xrightarrow{a} s'$ is true if there exist $p, p' \in S_M$, $n, n' \in N$ and a digitization quantum $\xi \in [0, 1)$ such that: $s = [p]_\xi$, $s' = [p']_\xi$ and

$$\langle p, n \rangle \xrightarrow{a} \langle p', n' \rangle.$$

If S is already a node of $Dig-Approx(M)$ then we add the edge $S \xrightarrow{a} S'$ to it.

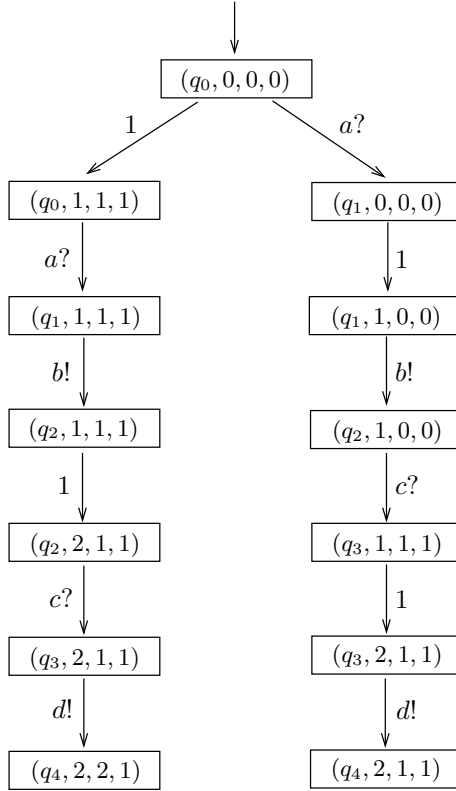


FIGURE 3: The (digital) approximate model of the DVTG-IO of Figure 1.

By the generation method, the size of $Dig-Approx(M)$ may increase infinitely. A possible way for tackling that is to fix some maximal depth during the generation of the graph. We can also put a limit on the number of nodes of the graph as well. Similarly, many other criteria may be considered in order to guarantee the finiteness of $Dig-Approx(M)$.

Example: An example of an approximate model (for the DVTG-IO M of Figure 1) is given in Figure 3. For pedagogical reasons, the example of approximate model, we propose, is simply deduced from the two integer computation sequences of M given so far in Section 4.1. Notice that all the nodes of $Dig-Approx(M)$ are singletons of integer states.

5. DIGITAL TIMED TESTS

Digital timed tests can be represented as either total functions or as *labeled transition systems with inputs and outputs* (LTS-IO for short).⁴ An *untimed test* for a specification $Spec$ over the set of actions $Act = In \cup Out$ is a total function

$$T : (Act \cup N)^* \rightarrow In \cup \{\text{wait, pass, fail}\}.$$

$T(\omega)$ specifies the action the tester must take once it observes ω . If $T(\omega) = a? \in In$ then the tester emits input $a?$. If $T(\omega) = \text{wait}$ then the tester waits (lets time elapse). If $T(\omega) \in \{\text{pass, fail}\}$ then the tester produces a verdict (and stops).

⁴An LTS-IO is the untimed version of the DVTG-IO model (i.e., an LTS-IO = a DVTG-IO with no duration variables). See [17] for the formal definition of an LTS-IO.

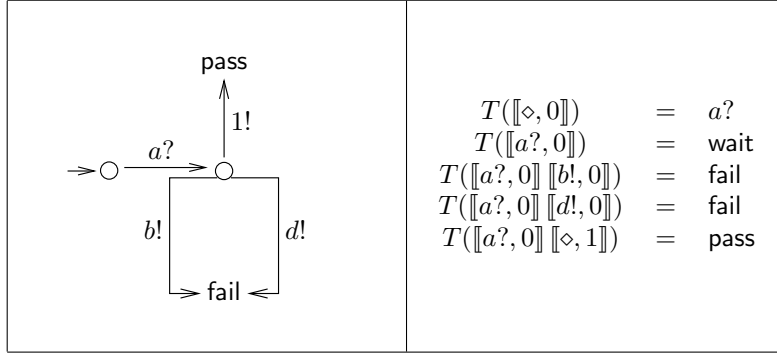


FIGURE 4: A digital timed test represented as a LTS-IO or a function.

To represent a valid test, T must satisfy a number of conditions:

$$\exists n \in \mathbb{N}, \forall \omega \in (Act \cup N)^* : |\omega| > n \implies T(\omega) \in \{\text{pass}, \text{fail}\} \quad (1)$$

$$\forall \omega \in (Act \cup N)^* :$$

$$T(\omega) \in \{\text{pass}, \text{fail}\} \implies \forall \omega' \in (Act \cup N)^* : T(\omega \cdot \omega') = T(\omega) \quad (2)$$

where “ $|\omega|$ ” is the length of “ ω ” and “ $\omega \cdot \omega'$ ” is the timed word obtained by concatenating the two timed words.

Condition (1) states that the test reaches a verdict after the execution of bounded number n of discrete actions. Condition (2) is a “suffix-closure” property ensuring that the test does not recall a verdict.

The LTS-IO corresponding to T is defined as follows. The states of the LTS-IO are sequences $\omega \in (Act \cup N)^*$. The initial state is $\llbracket \diamond, 0 \rrbracket$ (the empty timed word). For every $a! \in Out \cup \{1\}$ and every timed word $\omega = \llbracket a_1, t_1 \rrbracket \cdots \llbracket a_n, t_n \rrbracket$ there is a transition $\omega \xrightarrow{a!} \omega \cdot \llbracket a?, t_n \rrbracket$. If $T(\omega) = a? \in In$ then there is a transition $\omega \xrightarrow{a?} \omega \cdot \llbracket a?, t_n \rrbracket$. As a convention, all states ω such that $T(\omega) = \text{pass}$ are “collapsed” into a single sink state pass , and similarly with fail .

Example: A possible digital timed test for the DVTG-IO of Figure 1 is given in Figure 4. The test defined in the right part of the figure can be equivalently represented by the DVTG-IO shown in the left part. Function T is partially defined in the figure. The remaining cases are covered by the suffix-closure property of pass/fail – Condition (2). For instance, $T(a? b! d!) = \text{fail}$, because $T(a? b!) = \text{fail}$.

The execution of a digital timed test: A digital timed test T represented as an LTS-IO has two types of nodes, namely input- and output-nodes. An *input-node* has only one outgoing edge which is labeled with an input action. An *output-node* has as many outgoing edges as the number of elements of $Out \cup \{1\}$. Each outgoing edge is labeled with a distinct a from $Out \cup \{1\}$. While executing T the tester emits an input to the IUT if it is currently occupying an input-node. The input-action is the label of the outgoing edge from the current node. It must be sent to the IUT before the next “tick” of the digital clock happens.

If the current node is an output-one then the tester only waits for the next output action the IUT may send or for the next tick to happen. As soon as action $a \in Out \cup \{1\}$ is received the tester declares fail and stops the test if a is not accepted by the specification. Otherwise, the tester follows the corresponding edge (i.e., the one labeled with a) and moves to the next node. The tester keeps repeating that until reaching either pass or fail .

6. TEST GENERATION

We recall the untimed test generation algorithm of [17]. Roughly speaking, the algorithm builds a test in the form of a tree. A node in the tree is a set of states S of the specification and represents the “knowledge” of the tester at the current test state. The algorithm extends the test by adding successors to a leaf node. For all *illegal* outputs ν_i (outputs which cannot occur from any state in S) the test leads to fail. For each legal output ν_i , the test proceeds to node S_i , which is the set of states the specification can be in after emitting ν_i . At each step, the test may decide to emit an input μ_j of the specification. At any node, the algorithm may decide to stop the test and label this node as pass.

Untimed test generation is performed by Algorithm 1. The algorithm uses the following notation. Given a nonempty set X , $\text{pick}(X)$ chooses randomly an element in X . Given a set of states P and an action μ , $\text{succ}(P, \mu)$ is defined as the set of states which can be reached by some state in P after executing action μ . That is:

$$\text{succ}(P, \mu) = \{q' \mid \exists q \in P : q \xrightarrow{\mu} q'\}.$$

The algorithm is extended in a straightforward way to specifications given as digital approximate models. The set of outputs to be used is $Out \cup \{1\}$. The set of inputs is simply In .

```

1   $S \leftarrow \{q_0\}$ ;
2   $T \leftarrow$  the one-node tree with root  $S$ ;
3  while(true)
4    foreach(leaf  $S$  of  $T$  distinct from pass and fail)
5       $i \leftarrow \text{pick}(\{0, 1, 2\})$ ;
6      case( $i = 0$ ) :
7         $\mu \leftarrow \text{pick}(In)$ ;
8         $S' \leftarrow \text{succ}(S, \mu)$ ;
9        append edge  $S \xrightarrow{\mu} S'$  to  $T$ ;
10     case( $i = 1$ ) :
11       foreach( $\nu \in Out$ )
12          $S' \leftarrow \text{succ}(S, \nu)$ ;
13         if ( $S' \neq \emptyset$ ) append edge  $S \xrightarrow{\nu} S'$  to  $T$ ;
14         else append edge  $S \xrightarrow{\nu} \text{fail}$  to  $T$ ;
15         endif;
16       endforeach;
17     case( $i = 2$ ) : replace  $S$  with pass in  $T$ ;
18   endforeach;
19 endwhile;
```

Algorithm 1: Untimed test generation.

Example: A possible test tree that may be extracted from the (digital) approximate model given in Figure 3 is the one given in Figure 4. The algorithm “decides” to make the root of the test an input-node. According to Figure 4, the approximate model allows only input $a?$ from its initial location. Thus, the outgoing edge from the root of the test is labeled with $a?$. The approximate model states that one tick shall be observed before the output action $b!$ happens. Hence, both outputs $b!$ and $d!$ lead to fail within the test tree. The test generation procedure is topped at depth 2. This test can be clearly extended to a longer one.

7. CONCLUSION

We introduced a method for testing duration systems. We proposed a framework for modelling real-time systems based on Duration Variable Timed Graphs with Inputs and Outputs (DVTG-IO) which extend the timed automaton model. We gave an approximation method based on digitization techniques. A given DVTG-IO is abstracted into an approximate model. Test trees are then extracted from this approximate model.

REFERENCES

- [1] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *LICS*, pages 414–425, 1990. 2
- [2] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994. 1, 2
- [3] Henrik C. Bohnenkamp and Axel Belinfante. Timed testing with torx. In *FM*, pages 173–188, 2005. 2
- [4] Ahmed Bouajjani, Rachid Echahed, and Riadh Robbana. Verifying invariance properties of timed systems with duration variables. In *FTRTFT*, pages 193–210, 1994. 1, 2
- [5] Karlis Cerans. Decidability of bisimulation equivalences for parallel timer processes. In *CAV*, pages 302–315, 1992. 2
- [6] Abdeslam En-Nouaary, Rachida Dssouli, Ferhat Khendek, and A. Elqortobi. Timed test cases generation based on state characterization technique. In *IEEE Real-Time Systems Symposium*, pages 220–, 1998. 2
- [7] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In *ICALP*, pages 545–558, 1992. 7
- [8] Anders Hessel, Kim Guldstrand Larsen, Brian Nielsen, Paul Pettersson, and Arne Skou. Time-optimal real-time test case generation using uppaal. In *FATES*, pages 114–130, 2003. 2
- [9] Anders Hessel and Paul Pettersson. A test case generation algorithm for real-time systems. In *QSIC*, pages 268–273, 2004. 2
- [10] A. Khoumsi. A method for testing the conformance of real time systems. In *FTRTFT'02*, volume 2469 of *LNCS*. Springer, 2002. 2
- [11] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. In *11th International SPIN Workshop on Model Checking of Software (SPIN'04)*, volume 2989 of *LNCS*. Springer, 2004. 2, 5
- [12] Lotfi Majdoub and Riadh Robbana. Test purpose of duration systems. In *MSVVEIS*, pages 67–75, 2006. 2
- [13] M. Mikucionis, K G. Larsen, and B. Nielsen. Online on-the-fly testing of realtime systems. In *Basic Research in Computer Science, BRICS Report Series RS-03-49*, December 2003. 2
- [14] P. Morel. *Une algorithmique efficace pour la génération automatique de tests de conformité*. PhD thesis, Université de Rennes, 2000. 2
- [15] Riadh Robbana. Verification of duration systems using an approximation approach. *J. Comput. Sci. Technol.*, 18(2):153–162, 2003. 7
- [16] Jan Springintveld, Frits W. Vaandrager, and Pedro R. D'Argenio. Testing timed automata. *Theor. Comput. Sci.*, 254(1-2):225–257, 2001. 2
- [17] Jan Tretmans. Testing concurrent systems: A formal approach. In *CONCUR*, pages 46–65, 1999. 2, 5, 11, 13